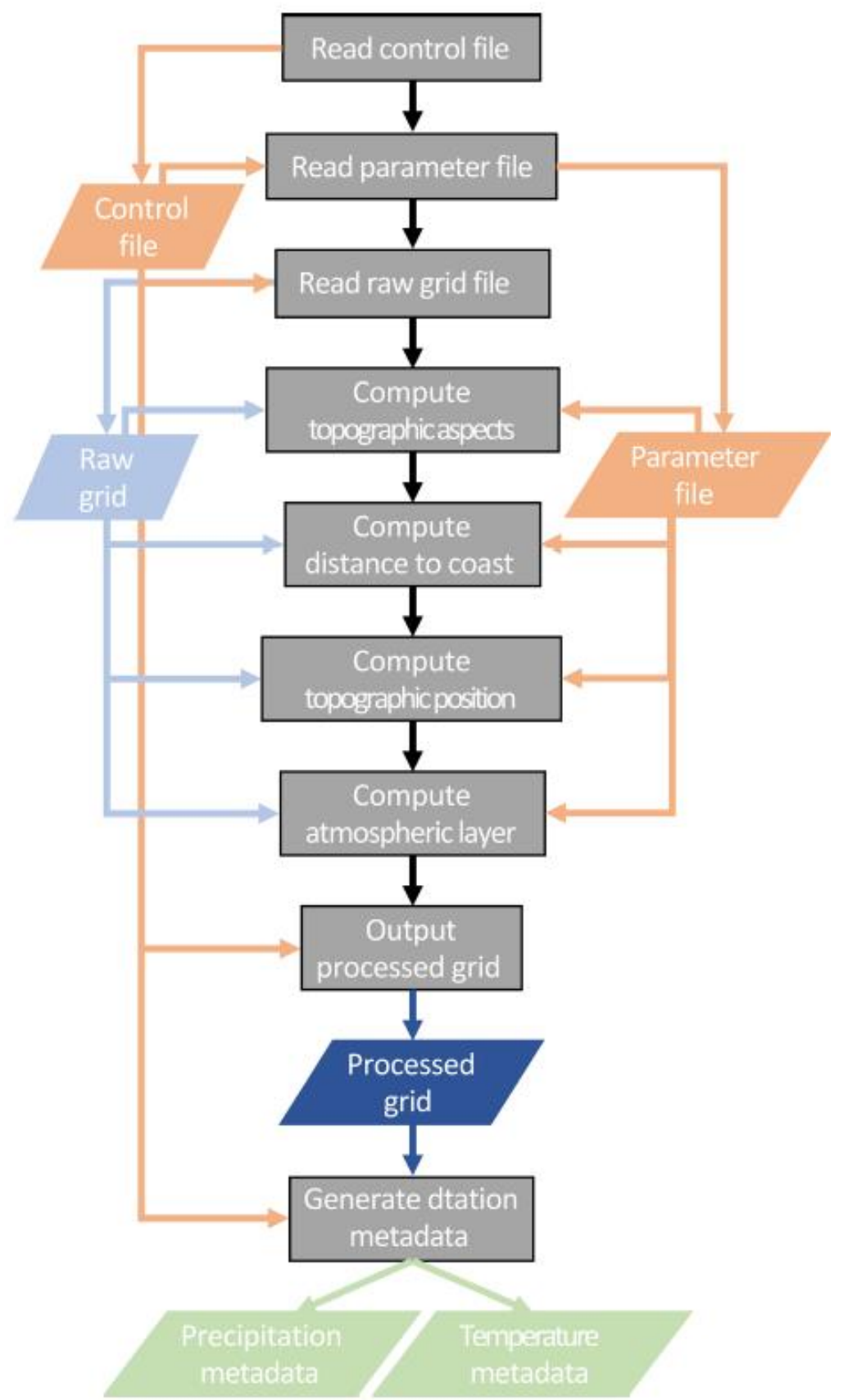
pyTIER: Development of a Python knowledge-based weather station interpolation algorithm

Mozhgan A. Farahani
University of Colorado Denver, National Center for Atmospheric Research

TIER Model Concept

Converting point-based observations from irregularly placed weather stations to values on a uniform high-resolution grid is challenging, especially in areas of complex terrain, and many different methods have been developed to create these data products.

Gridded precipitation and temperature products are used widely across the Earth Sciences as input data to process models, model verification datasets, and analysis datasets for features of interest (e.g., changes in temperature over time). The **Topographically InformEd Regression (TIER)** method, written in MATLAB®, implements a well-known, yet proprietary, knowledge-based interpolation system in an open-source repository.



Motivation

Why Conversion to Python?

- **Technical**: General-purpose programming
- **Freedom**: Without being locked-in with a given provider
- **Social**: Python community
- **Financial**: Free and have access to many free open-source packages

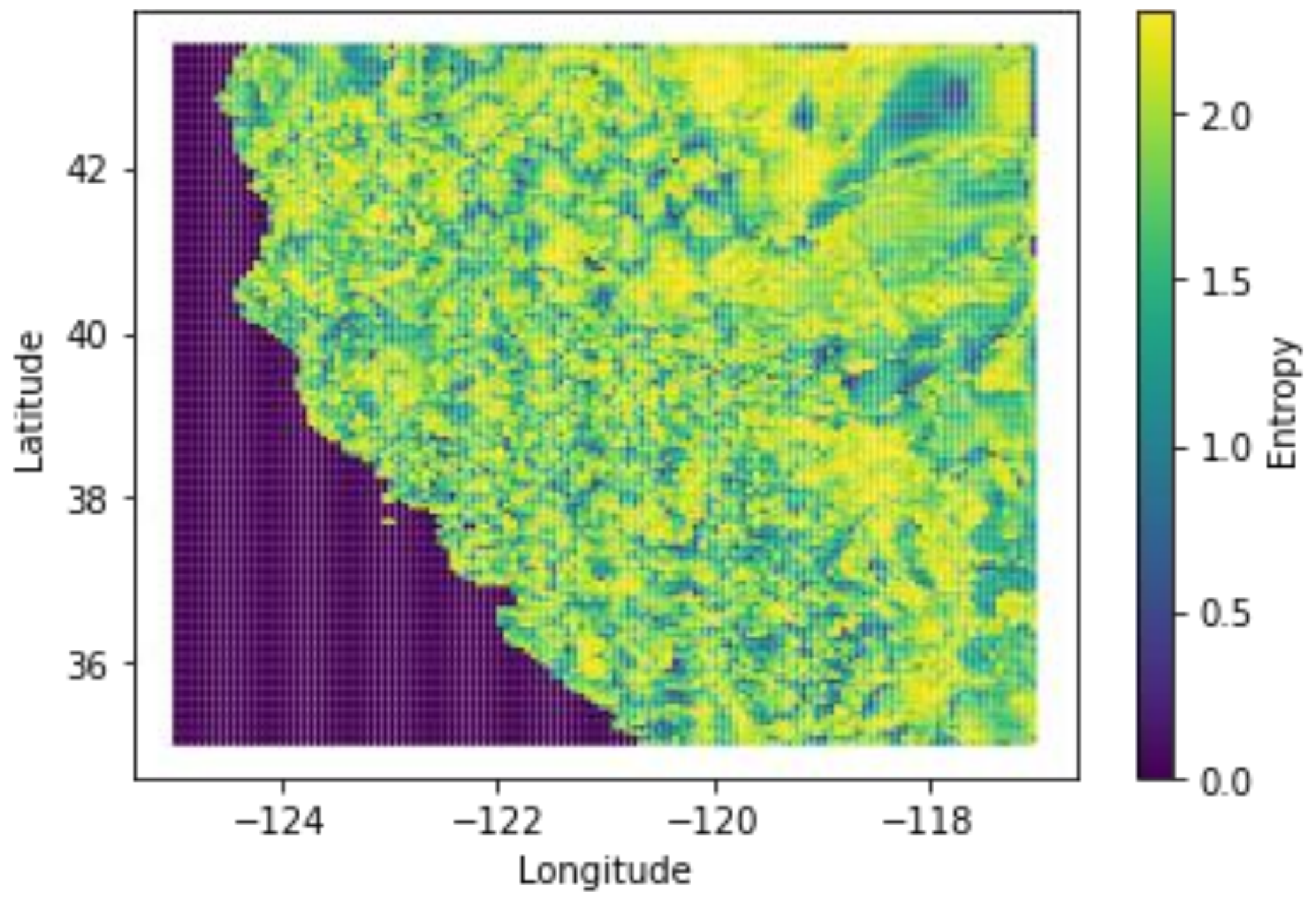
Apply Information Theory to Model

The **information** content of single variable is estimated using the concept of entropy (Shannon 1948), described as:

$$H(X) = -\sum_{i=1}^{m} P(x_i) \log P(x_i)$$

Entropy is a measure of the uncertainty associated with the occurrence of a certain event.

- How much information each station provides to each grid cell?
- How many stations do we need to consider?
- Quantify and prioritize the amount of information contained in a single station and shared among 2 or more stations?

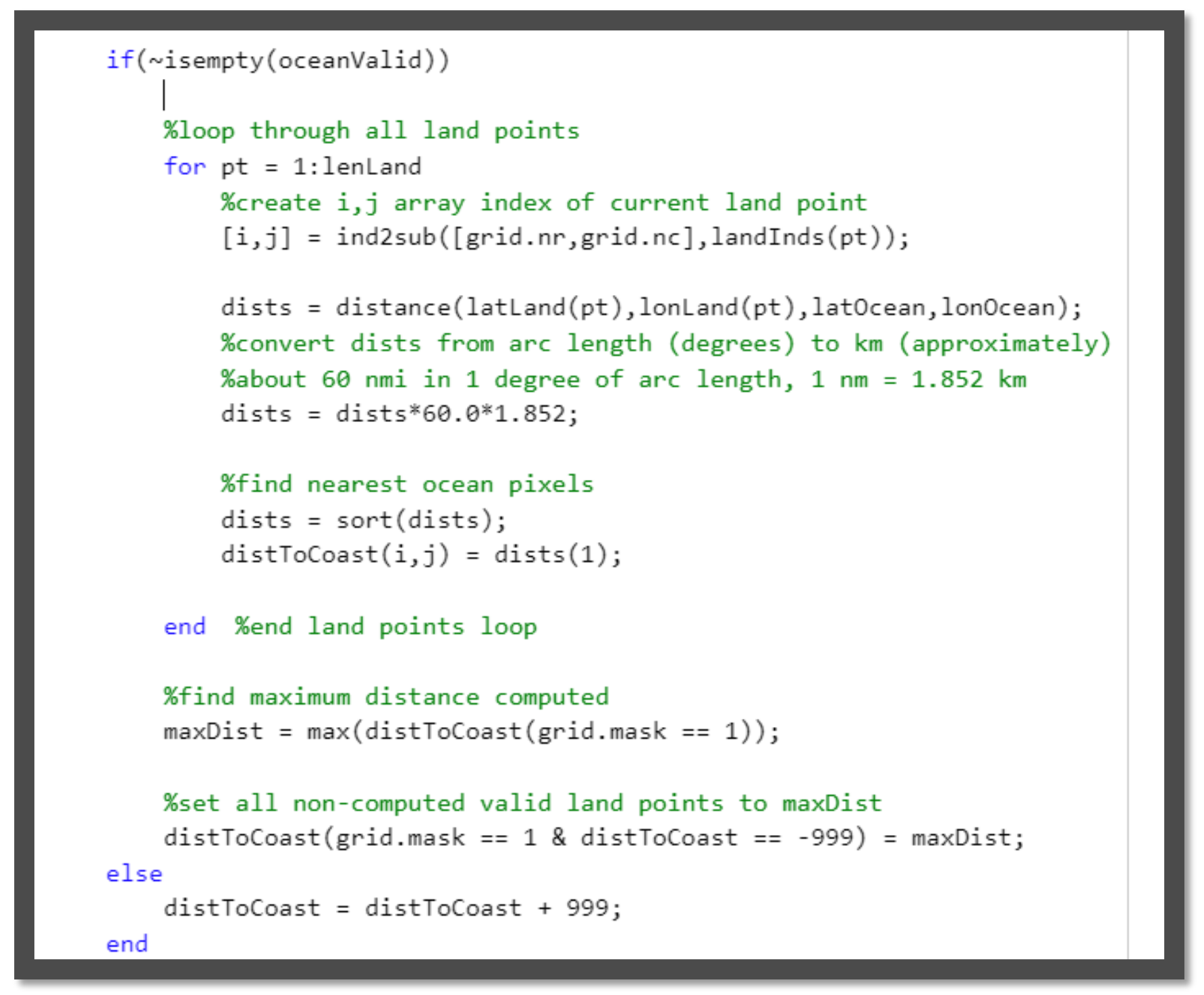


Challenges

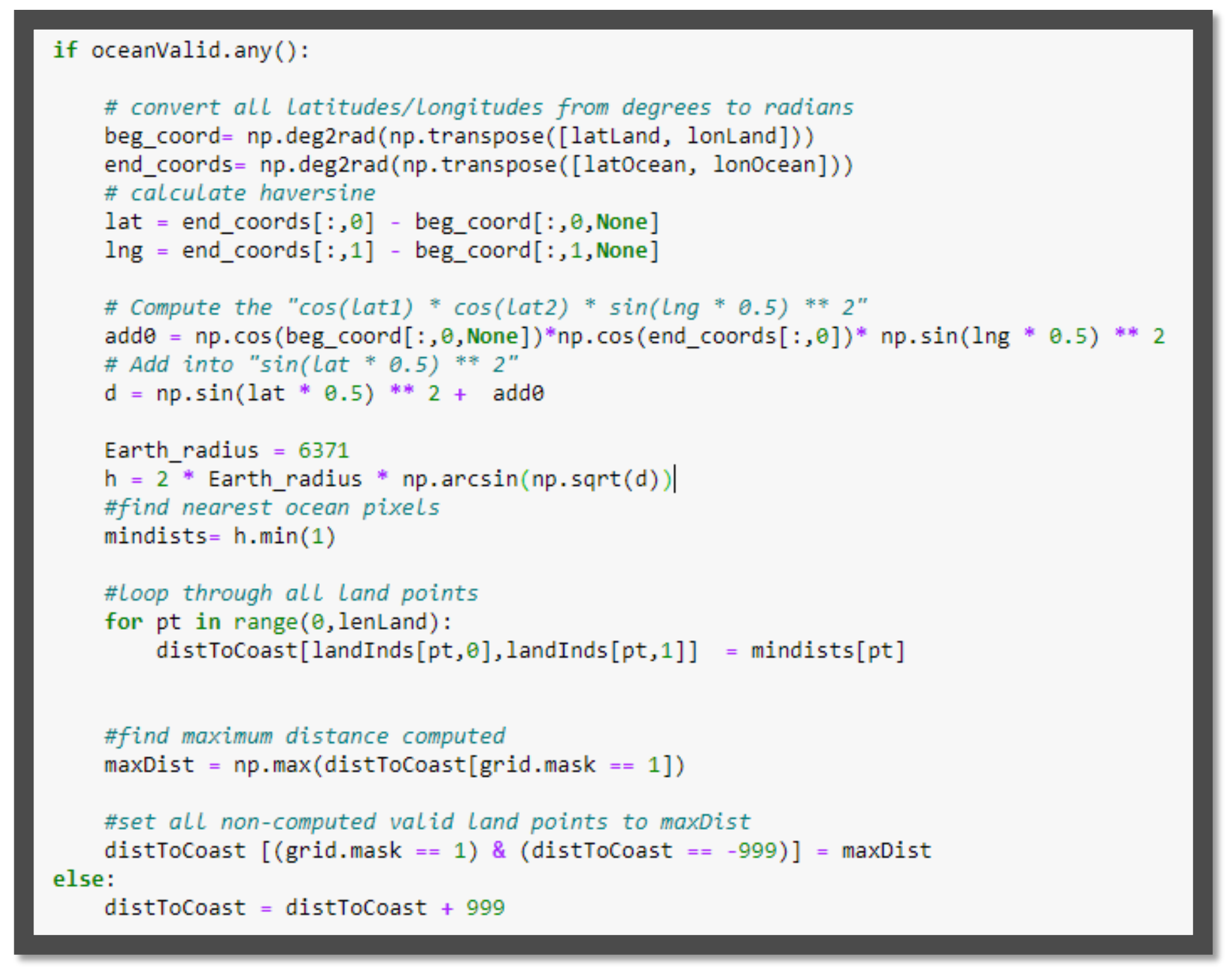
Matlab distance Function:
Distance between points on sphere or ellipsoid

[arclen,az] = distance(lat1,lon1,lat2,lon2)

- Here's a vectorized method leveraging the NumPy ufuncs (haversine func) to replace math-module funcs
- we are enabling to operate on entire arrays in one go



```
if(~isempty(oceanValid))
    |
    %loop through all land points
    for pt = 1:lenLand
        %create i,j array index of current land point
        [i,j] = ind2sub([grid.nr,grid.nc],landInds(pt));

        dists = distance(latLand(pt),lonLand(pt),latOcean,lonOcean);
        %convert dists from arc length (degrees) to km (approximately)
        %about 60 nmi in 1 degree of arc length, 1 nm = 1.852 km
        dists = dists*60.0*1.852;

        %find nearest ocean pixels
        dists = sort(dists);
        distToCoast(i,j) = dists(1);

    end  %end land points loop

    %find maximum distance computed
    maxDist = max(distToCoast(grid.mask == 1));

    %set all non-computed valid land points to maxDist
    distToCoast(grid.mask == 1 & distToCoast == -999) = maxDist;
else
    distToCoast = distToCoast + 999;
end
```



```
if oceanValid.any():

    # convert all latitudes/longitudes from degrees to radians
    beg_coord= np.deg2rad(np.transpose([latLand, lonLand]))
    end_coords= np.deg2rad(np.transpose([latOcean, lonOcean]))
    # calculate haversine
    lat = end_coords[:,0] - beg_coord[:,0,None]
    lng = end_coords[:,1] - beg_coord[:,1,None]

    # Compute the "cos(lat1) * cos(lat2) * sin(lng * 0.5) ** 2"
    add0 = np.cos(beg_coord[:,0,None])*np.cos(end_coords[:,0])* np.sin(lng * 0.5) ** 2
    # Add into "sin(lat * 0.5) ** 2"
    d = np.sin(lat * 0.5) ** 2 +  add0

    Earth_radius = 6371
    h = 2 * Earth_radius * np.arcsin(np.sqrt(d))
    #find nearest ocean pixels
    mindists= h.min(1)

    #loop through all land points
    for pt in range(0,lenLand):
        distToCoast[landInds[pt,0],landInds[pt,1]]  = mindists[pt]


    #find maximum distance computed
    maxDist = np.max(distToCoast[grid.mask == 1])

    #set all non-computed valid land points to maxDist
    distToCoast [(grid.mask == 1) & (distToCoast == -999)] = maxDist
else:
    distToCoast = distToCoast + 999
```

Strategies for Converting Matlab to Python

Either do it manually or take the help of some tools

1. SMOP (Small Matlab and Octave to Python compiler)
2. matlab2python
3. OMPC (Open-Source Matlab-To-Python Compiler)

Useful for short scripts and find specific functions

How to know which module I should use? The SciPy stack

NumPy pandas SciPy matplotlib

- 11 Preprocessing Matlab functions converted to one python script
- 22 Main Matlab functions converted to one python script

Conclusion and Future Work

Developed a python version of the TIER weather station interpolation algorithm

- Enable users to better contribute to the TIER codebase
- Using a more popular, and free programming language
- make TIER more efficient for large spatiotemporal domain processing
- This project enables future developers to easily add parallelization for use on clusters and other high performance computing environments.
- Add additional interpolation methodologies in a user-friendly manner
- Potential applications of Information Theory to further improve our methodological understanding of these types of algorithm.

pyTIER GitHub repository:
https://github.com/NCAR/pyTIER/tree/develop

SCAN ME

Acknowledgment

Thanks to Andrew Newman, Seth McGinnis, Rachel McCrary for their mentorship. Special thanks to SIParCS team!

Mozhgan.askarzadeh@ucdenver.edu