Summer 2016 Internship: Mapping with MetPy

Alex Haberlie

7/29/2016

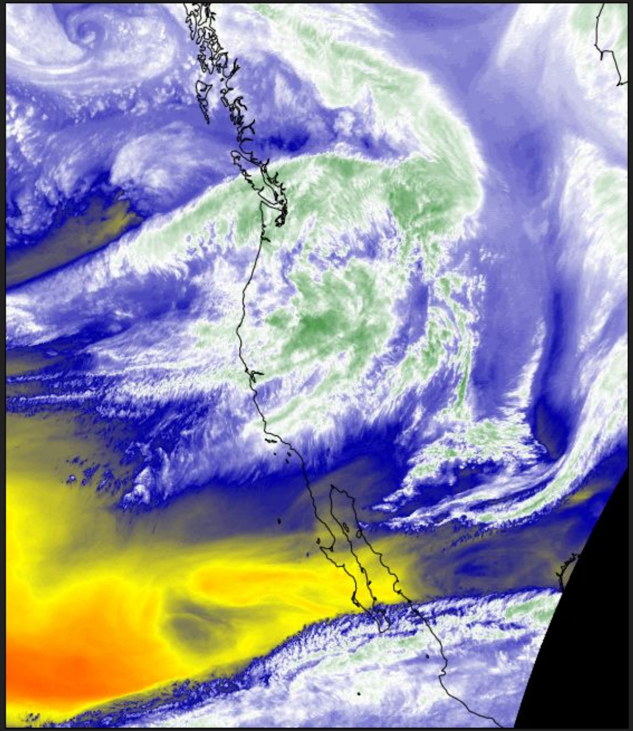
UCAR COMMUNITY PROGRAMS
unidata
Data Services and Tools for Geoscience

MetPy refresher

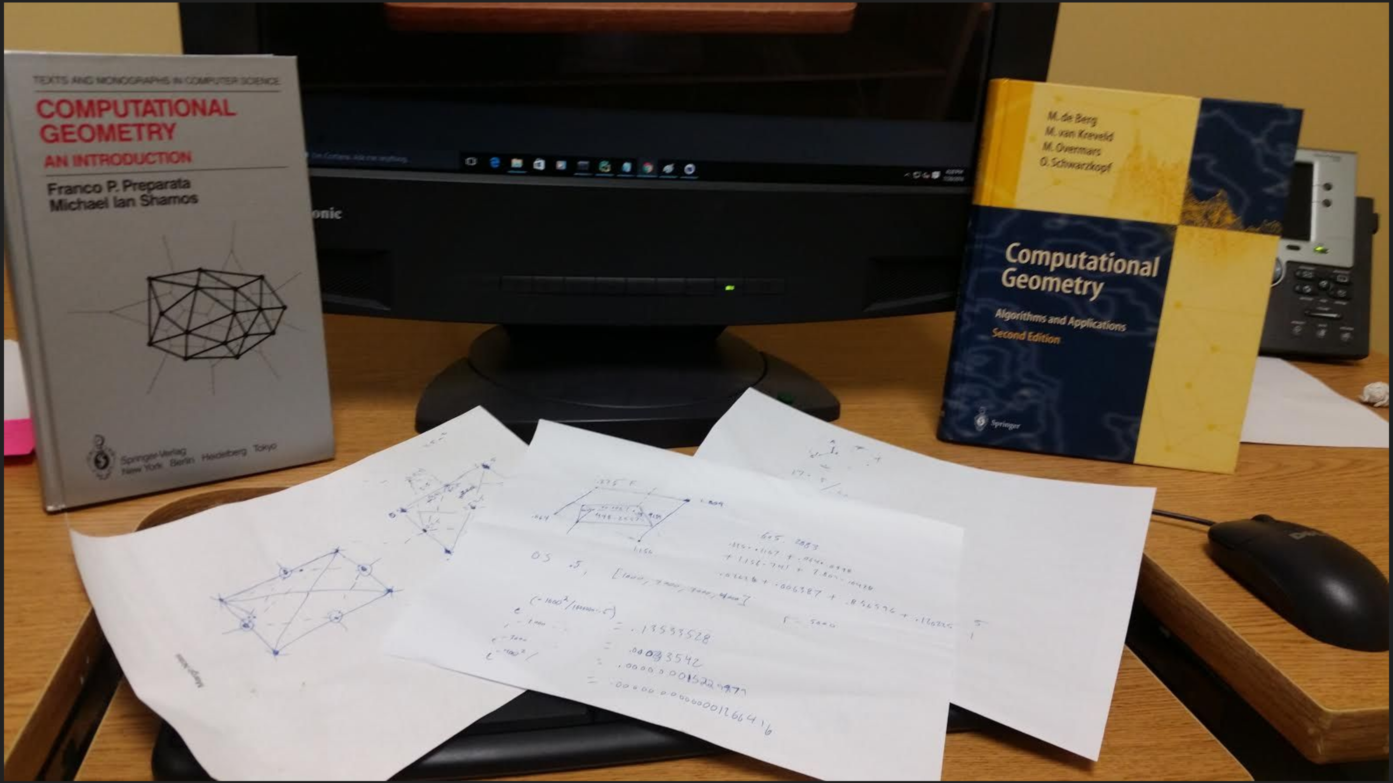
- “Collection of tools in Python for reading, visualizing and performing calculations with weather data.”
- “The space MetPy aims for is GEMPAK (and maybe NCL)-like functionality, in a way that plugs easily into the existing scientific Python ecosystem (numpy, scipy, matplotlib).”
- Something missing from MetPy was some built in mapping functionality, including:
  - Interpolation
  - GEMPAK-like mapping

Goal: Simplify mapping meteorological data in Python

- Wrap existing functions to reduce steps
  - Example, scipy.griddata requires:
    - “Zipped” coordinates
    - Observation values
    - fishnet/meshgrid for x and y dimensions
    - Interpolation type name
- Implement unavailable interpolation schemes
  - Barnes
  - Cressman
  - Natural Neighbor
  - More?
- Create a mapping class and process that mimics GEMPAK functionality
  - More on this later…



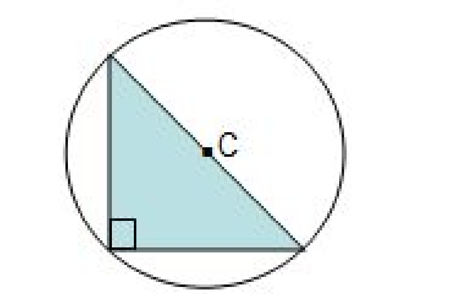
Step 1 of 4: Implement Natural Neighbor



Thanks for letting us borrow the books Julien!

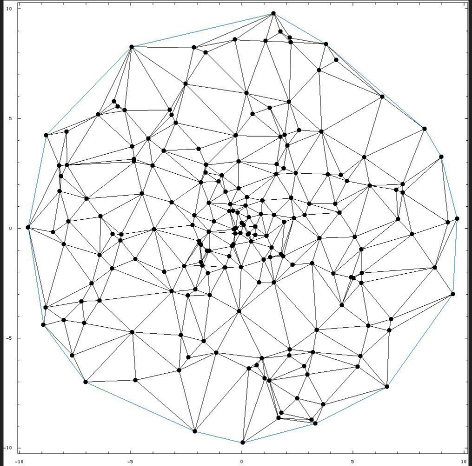
Natural Neighbor Interpolation Terms

Circumcircle of a triangle: A circle where the three vertices of a given triangle are on the perimeter.



Circumcenter of a triangle: The center of the circumcircle of a triangle.

Circumradius of a triangle: Radius of the circumcircle. Also the distance each given triangle vertex is from the circumcenter.



Delaunay Triangulation: A triangulation where none of the input points (coordinates) are within the circumcircle of any triangle.

Natural Neighbors: For a given point, a triangle is a natural neighbor if the point falls within that triangle’s circumcircle.

Natural Neighbor Pseudocode

For each grid point

  Find its natural neighbors, extract “edge” vertices and order them counter clockwise

  For each edge vertex (which is associated with an observation value)

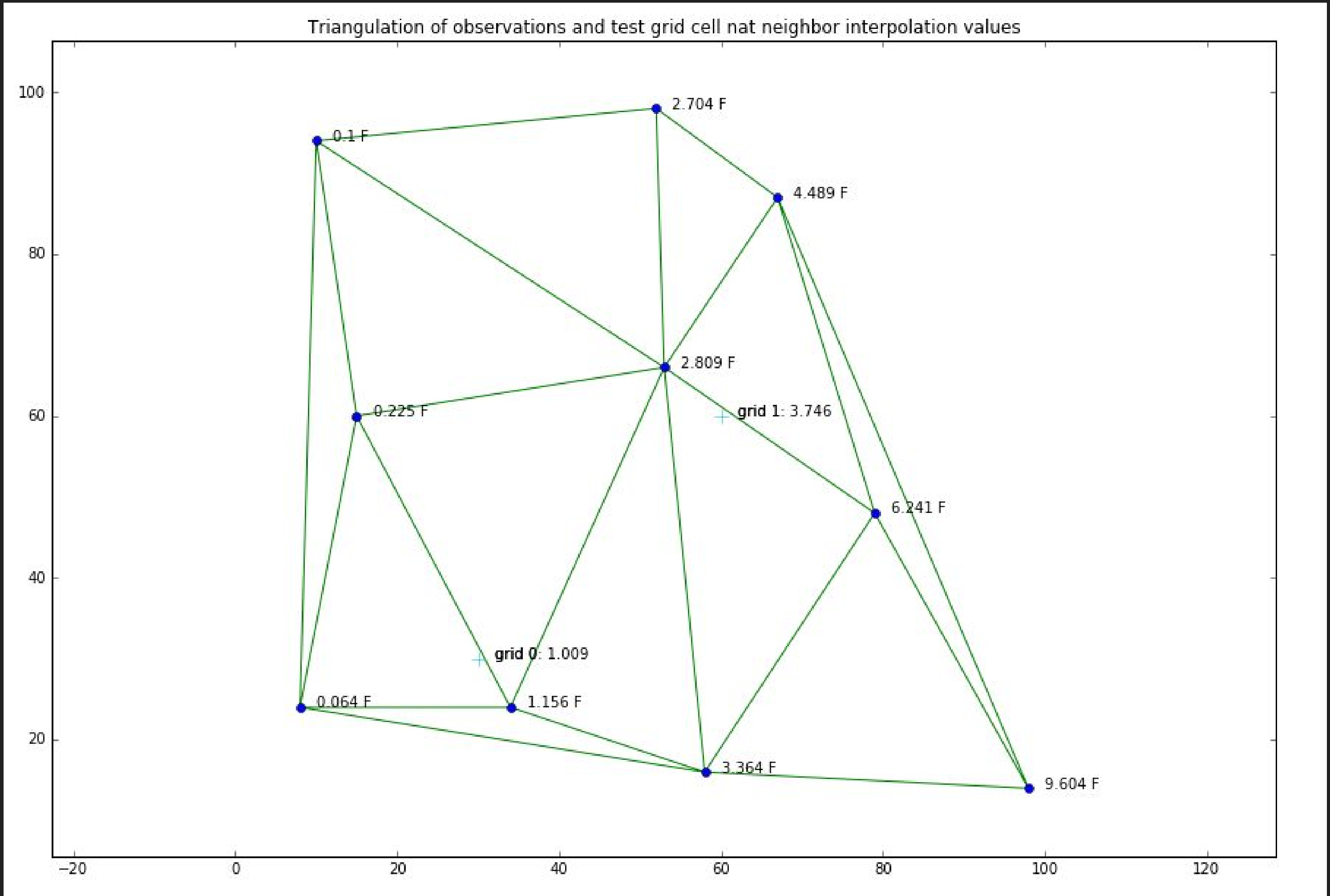
    Get the circumcenters for each triangle in which the edge vertex resides

    Find circumcenter of the current edge vertex, the grid point, the edge vertex “before”; repeat for “after” edge vertex

    Generate a polygon from these points, calculate its area, and repeat until each edge vertex is visited

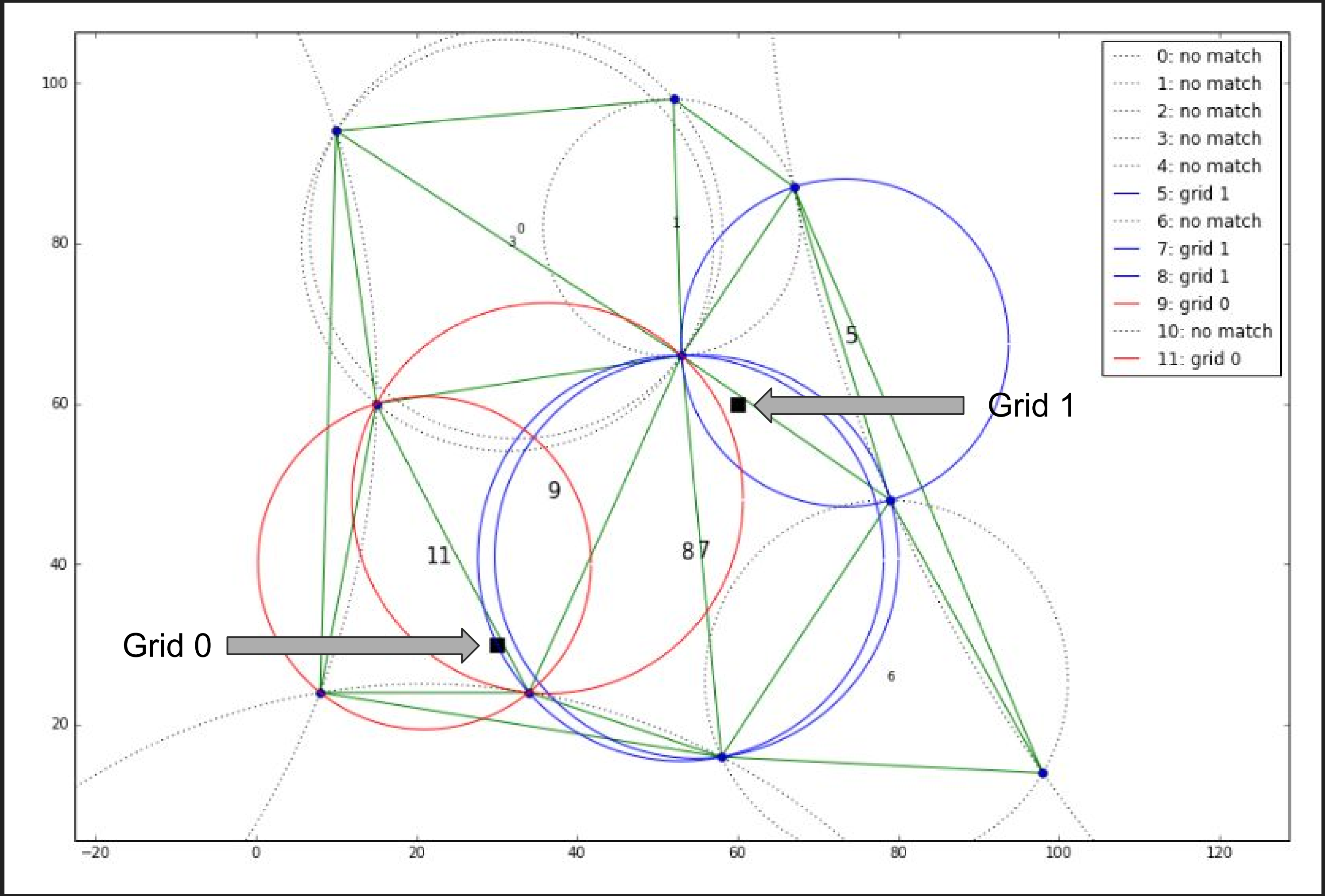
  The observation values are weighted by dividing each affiliated polygon area by the total area of all polygons

  Return the sum of the weighted observation values



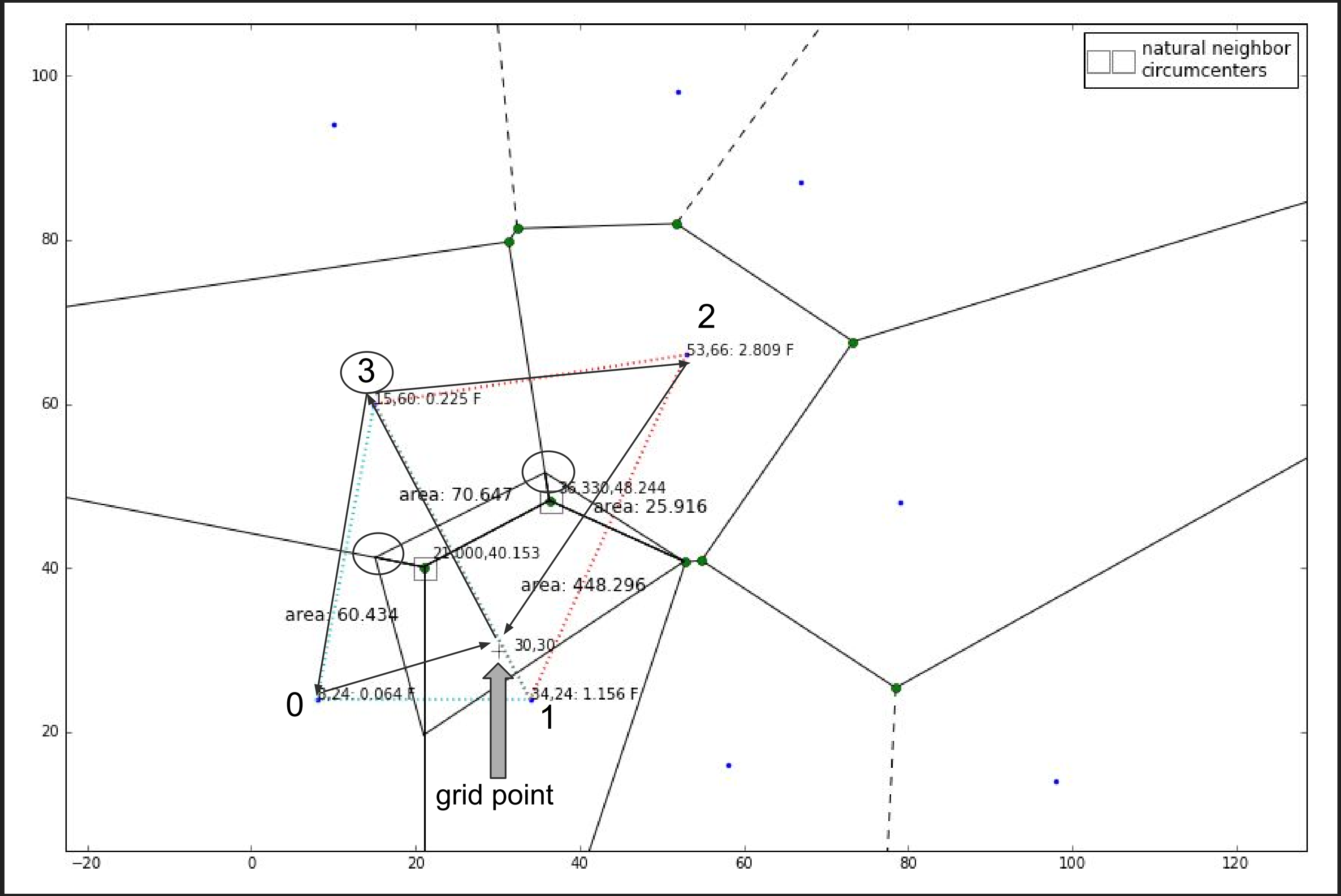
Random points

The “temperature” is equal to x*x / 1000



Preprocessing:

Associate each grid with its natural neighbor triangles and their circumcenters

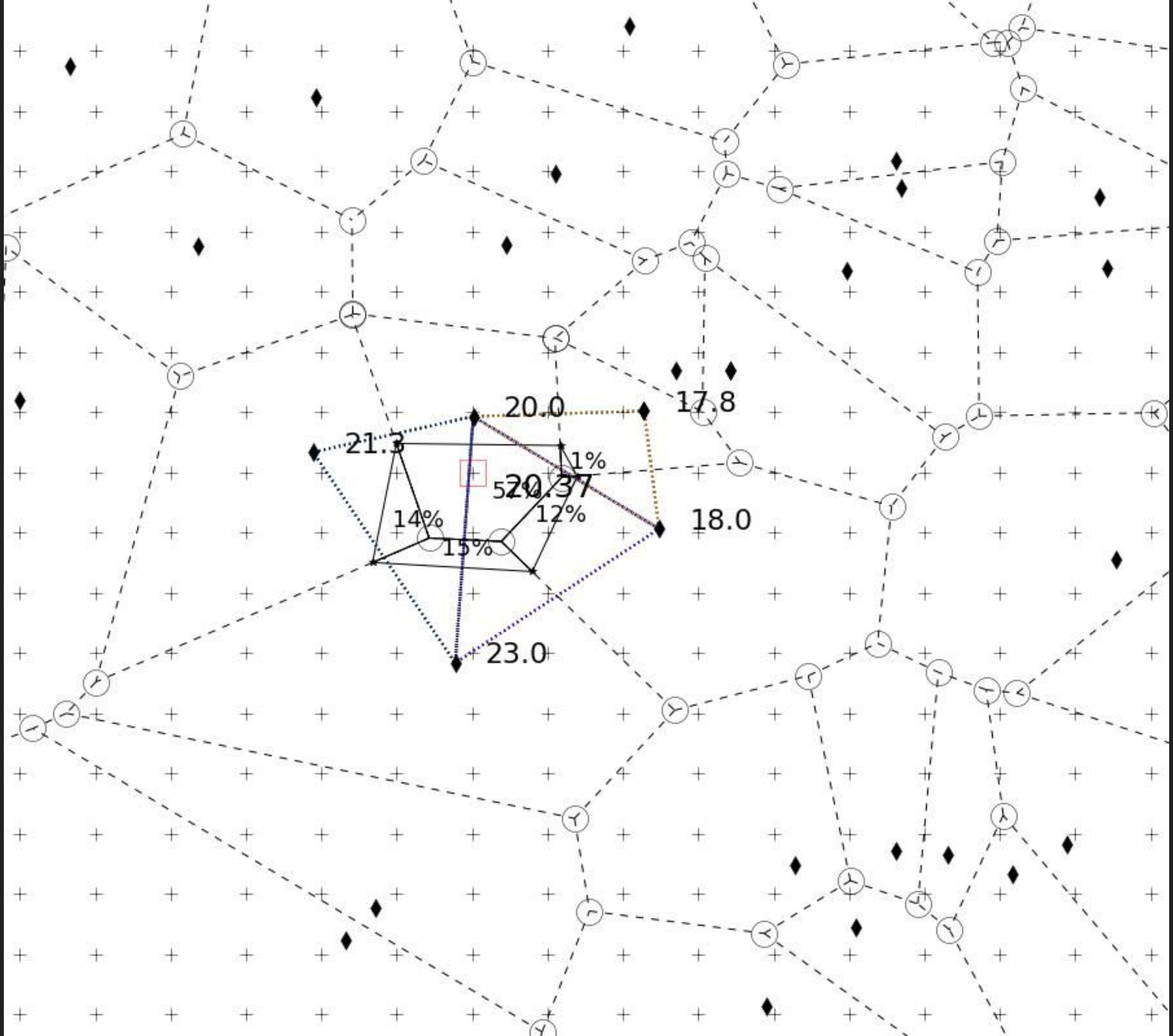


Edge vertex 3 is a part of the right and the left natural neighbor triangles, so its polygon will have 4 vertices:

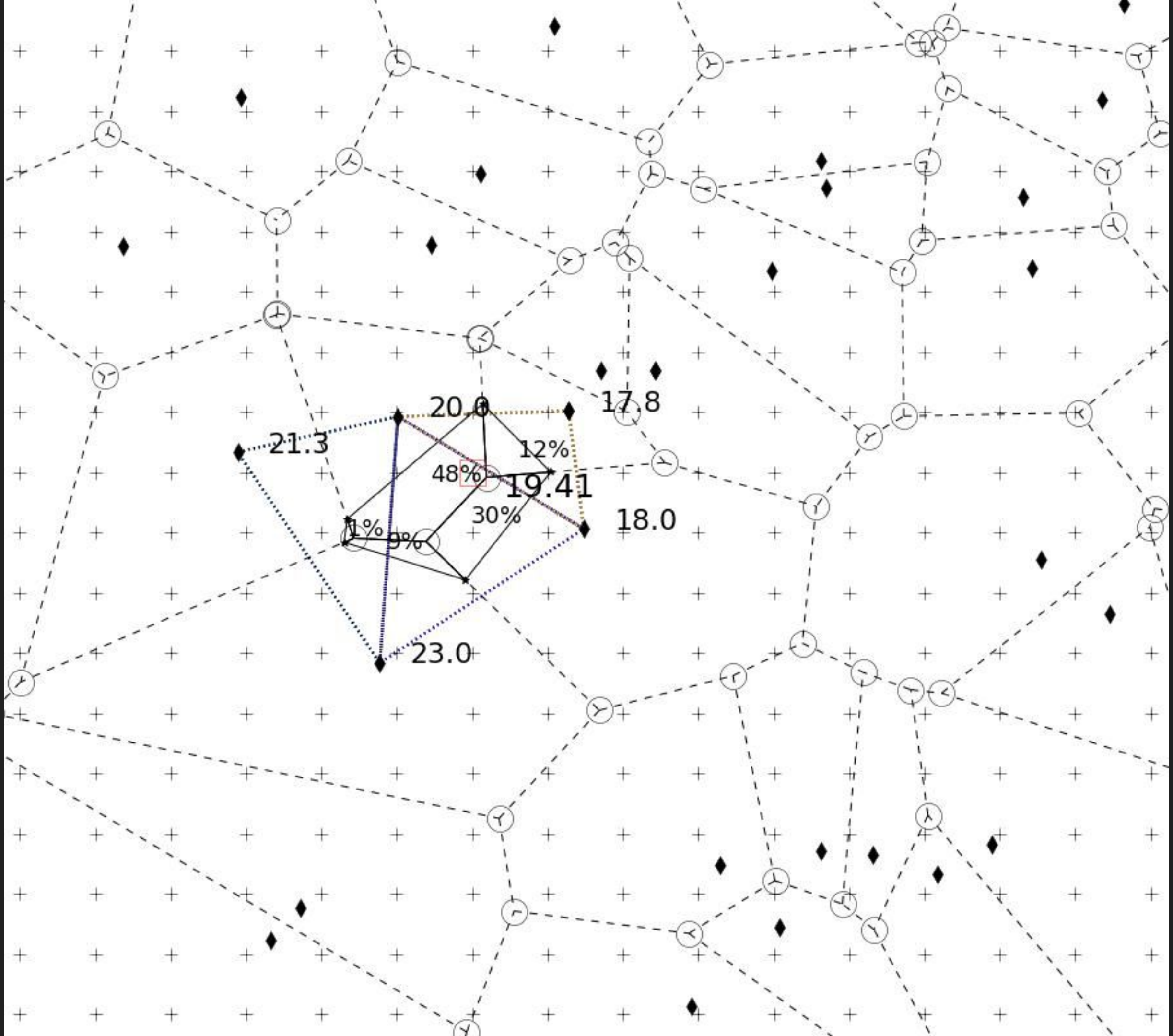
2 vertices will be the circumcenters of those two triangles

The other two will be:
1) the circumcenter of vertex 1, vertex 0, and the grid point at 30, 30

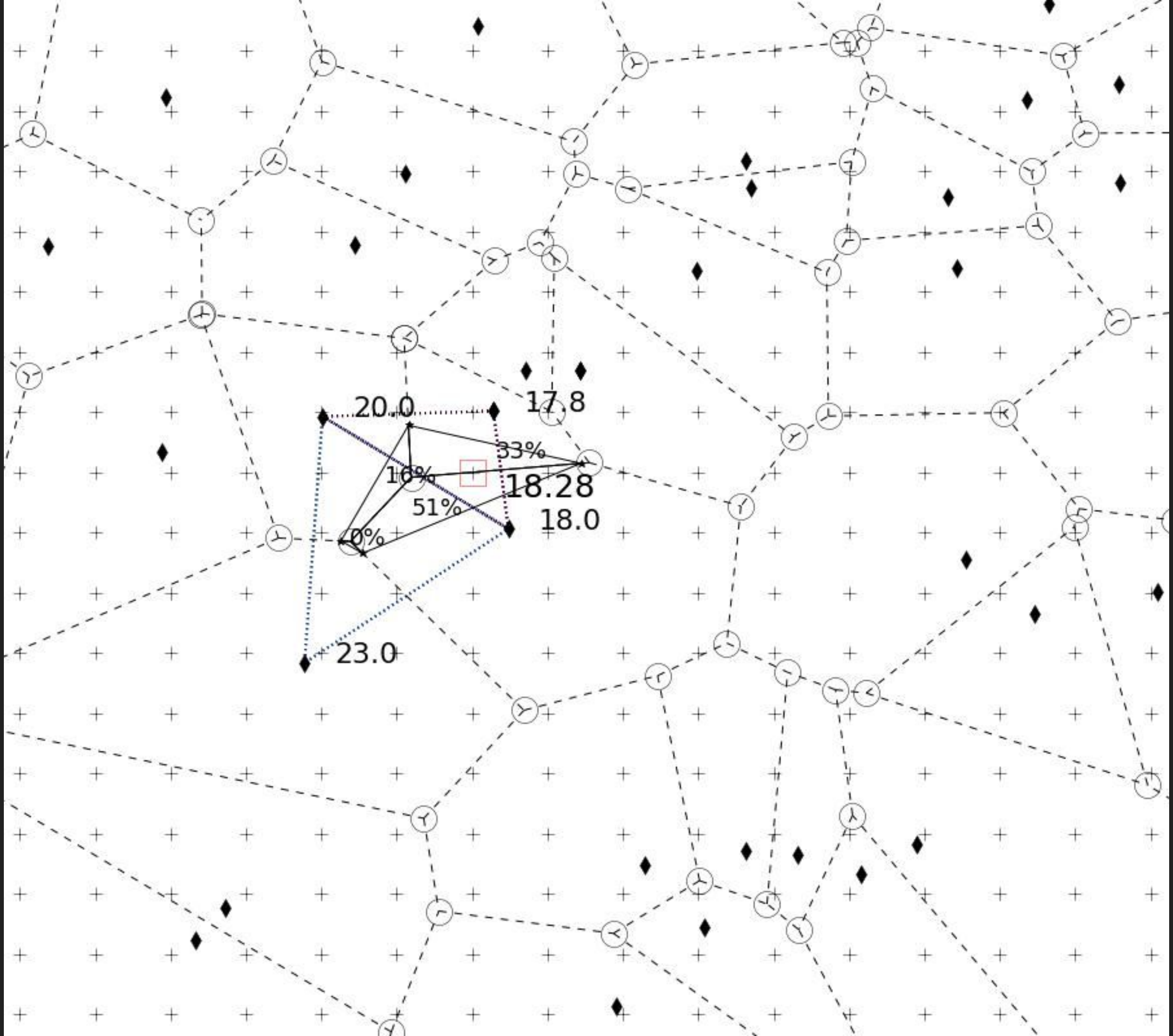
2) the circumcenter of vertex 1, vertex 2, and the grid point at 30, 30



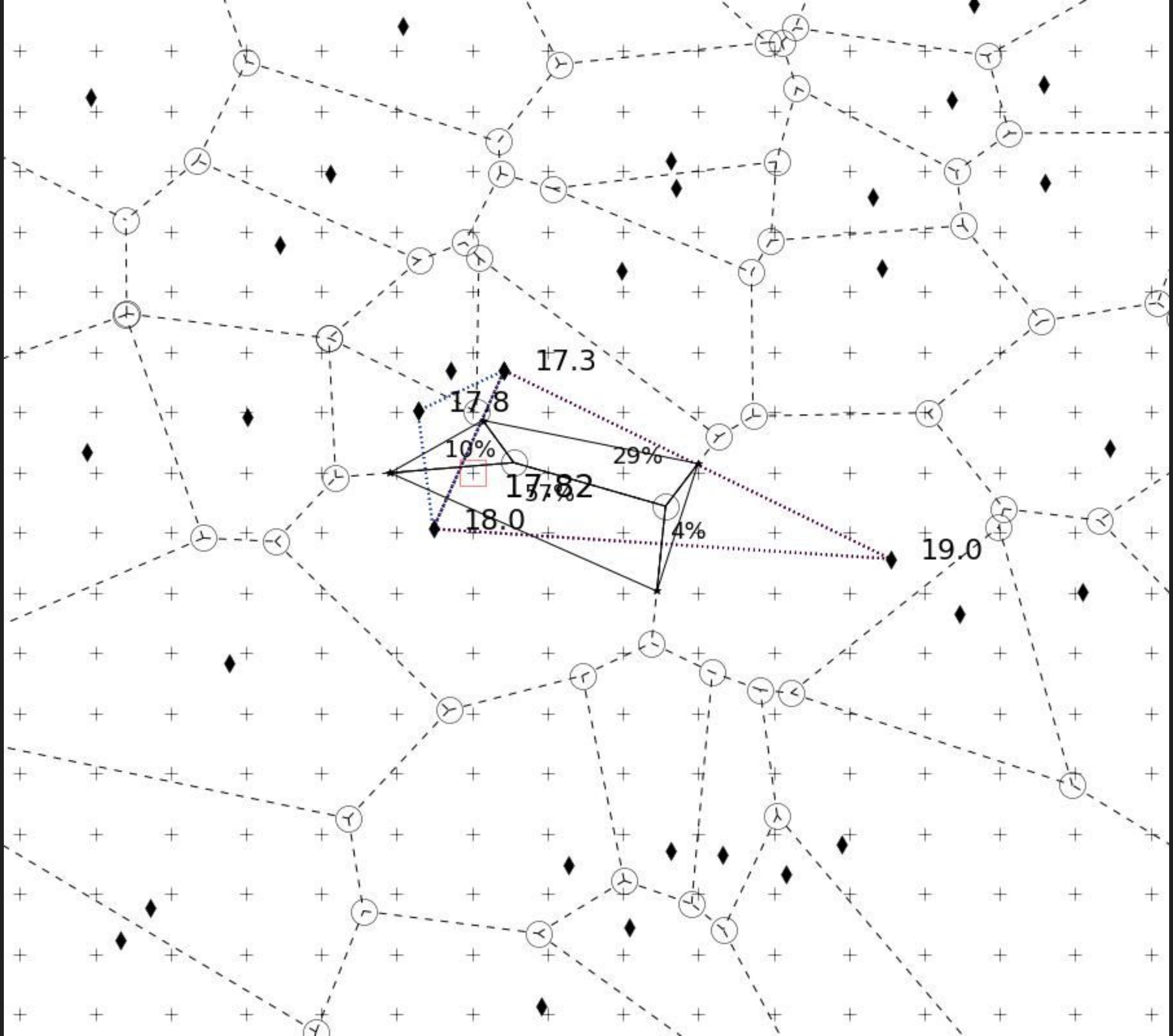
20.0
17.8
21.3
1%
20.37
52%
14%
12%
18.0
15%
23.0

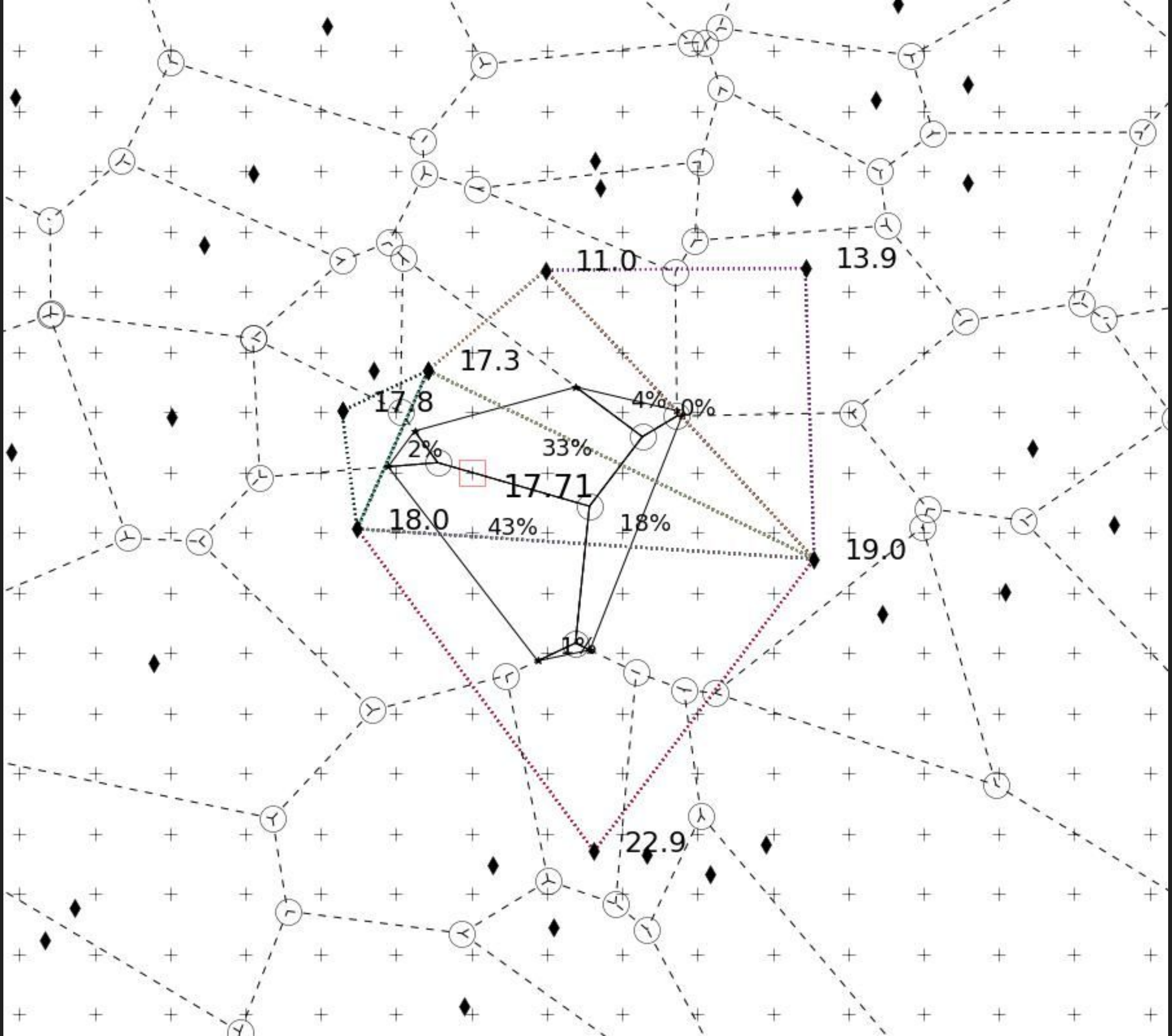


20.0
17.8
21.3
12%
48%
19.41
30%
18.0
1%
9%
23.0

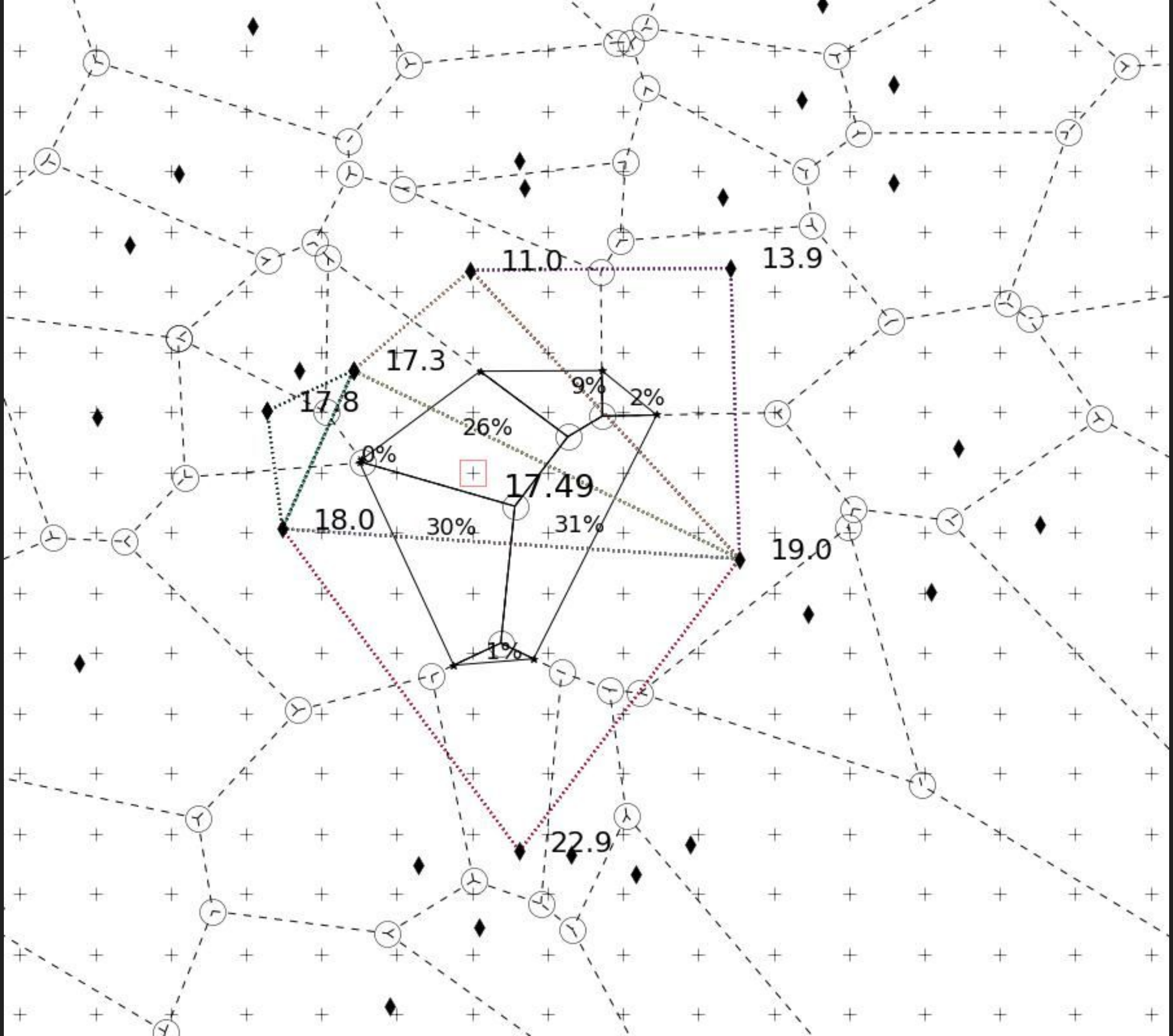


20.0
17.8
33%
16%
18.28
51%
18.0
0%
23.0


17.3
17.8
10%
29%
17.82
37%
18.0
4%
19.0

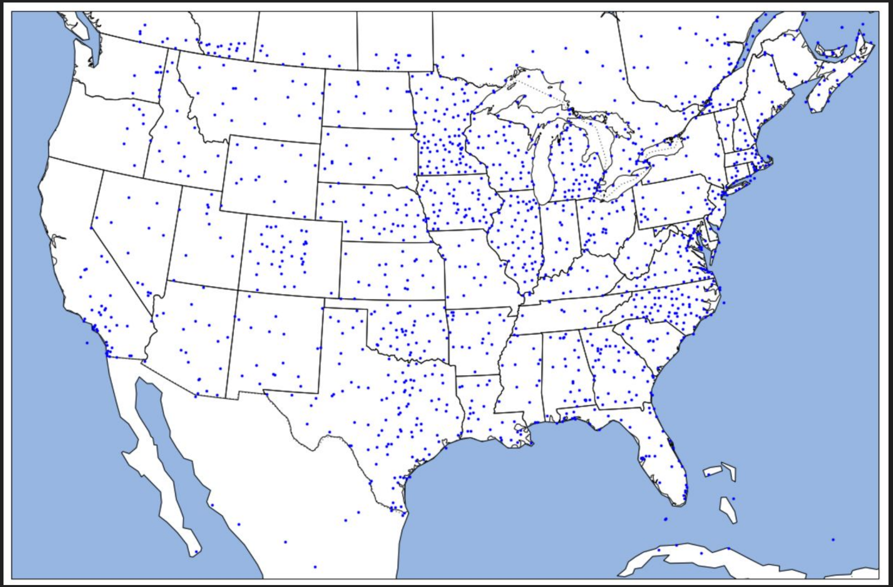


11.0
13.9
17.3
17.8
4%
0%
2%
33%
17.71
18.0
43%
18%
19.0
1%
22.9



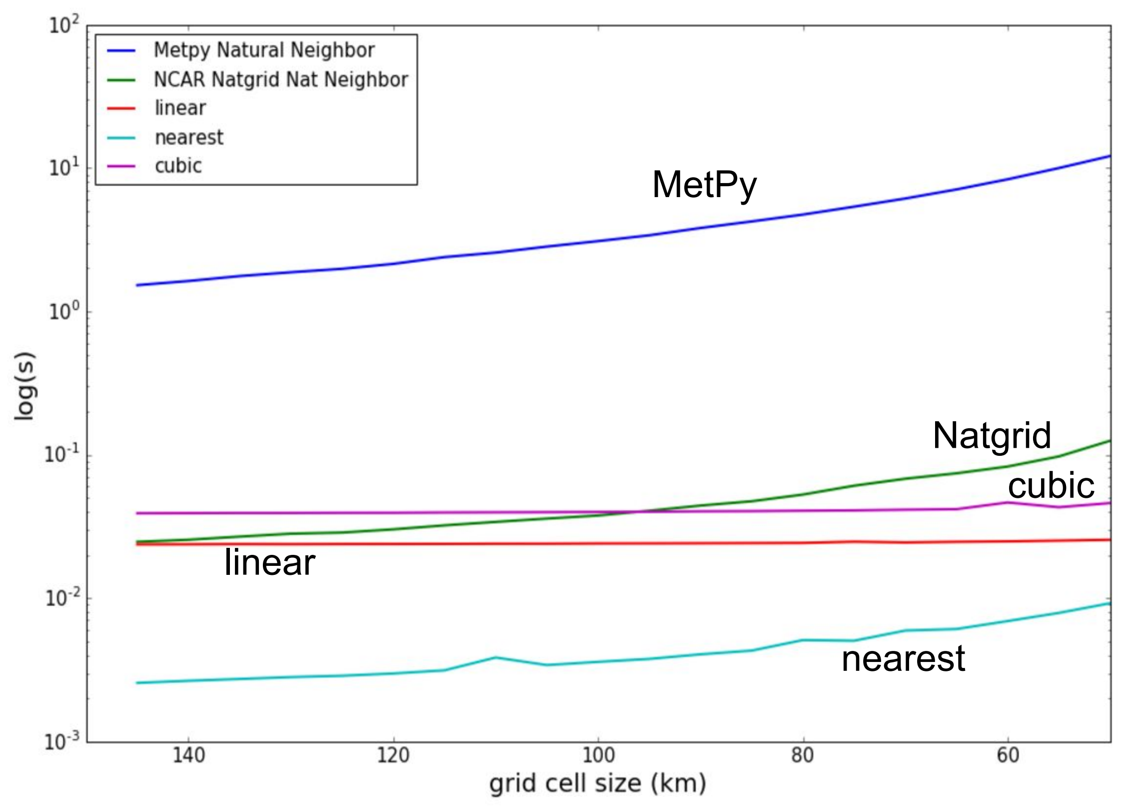
11.0
13.9
17.3
17.8
9%
2%
26%
0%
17.49
18.0
30%
31%
19.0
1%
22.9

“Real World” test example



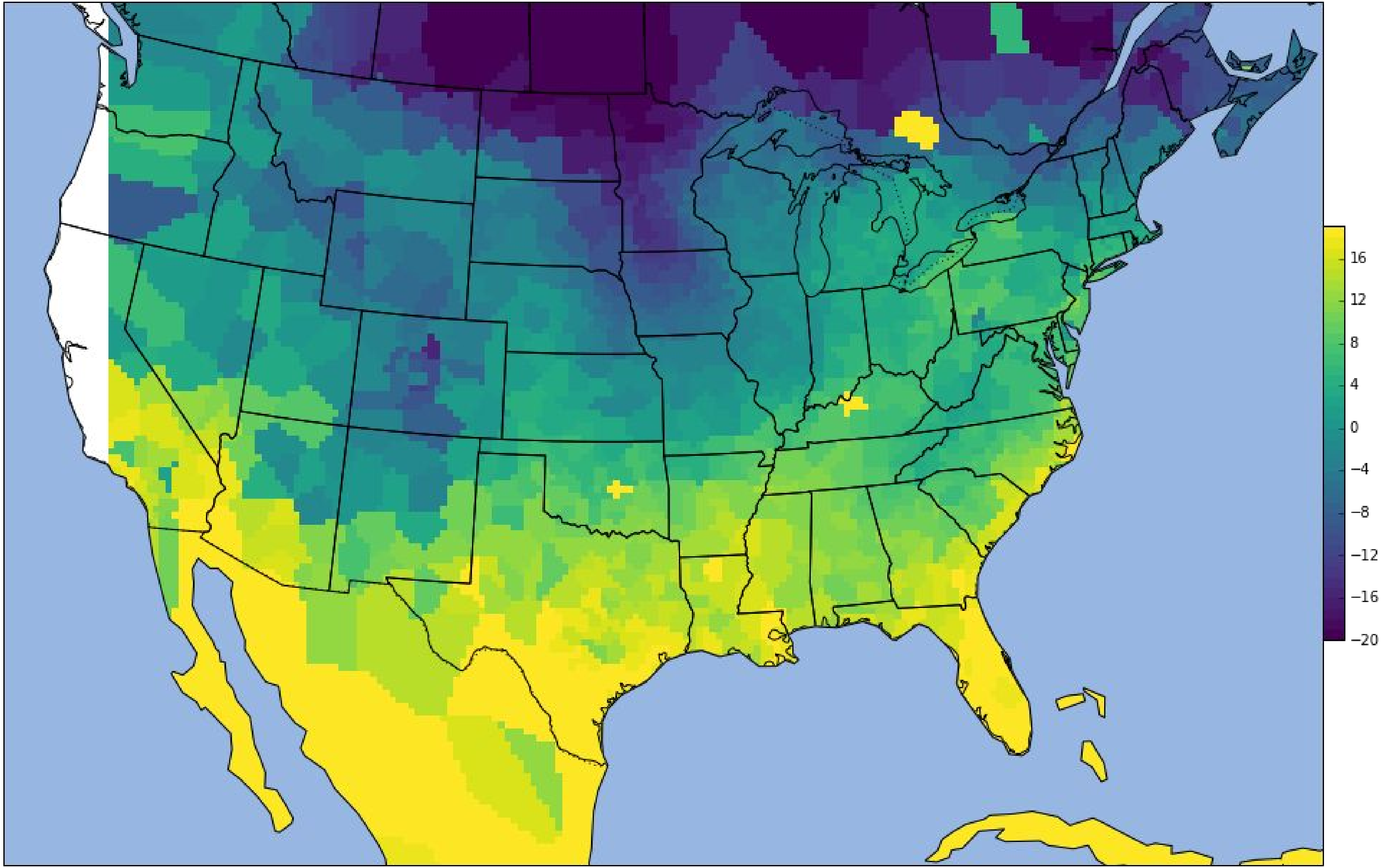
- ~1500 station observations from 00 UTC (evening) January 16th, 2016
- Sparse / Uneven distribution
- Plot the variable ‘air_temperature’
  - Minimum -24.1
  - Maximum 26.0
- Compare processing times between approaches
  - Grid sizes between 50 km (7350 grids) and 150 km (825 grids)
- Compare visualizations between approaches

Timing results

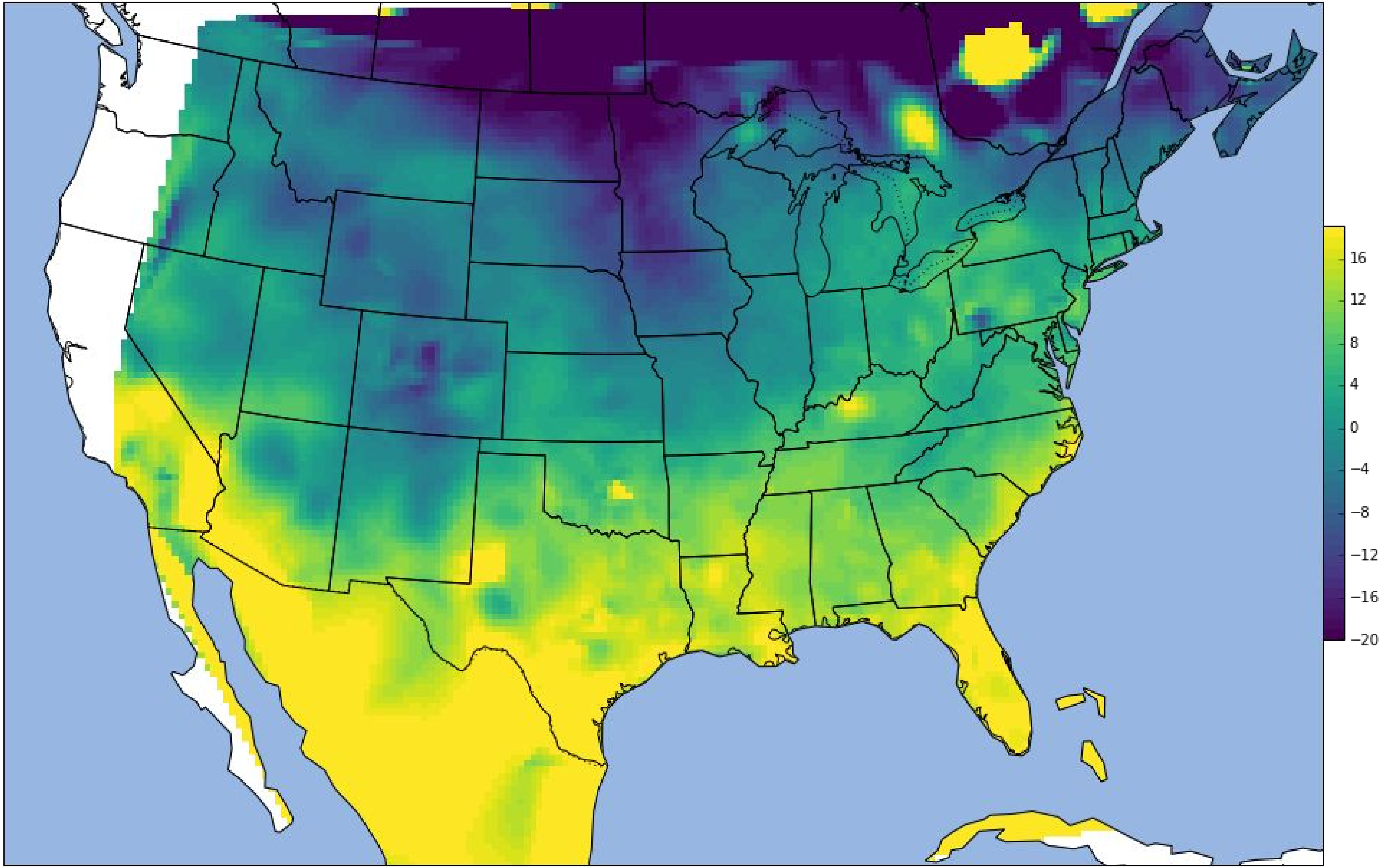


- MetPy version is much slower
  - As expected!
- MetPy.natural_neighbor is a pure python implementation
- SciPy uses Cython, C, and C++ on the backend
- The python package ‘Natgrid’ is basically a C wrapper for copyrighted and licensed code

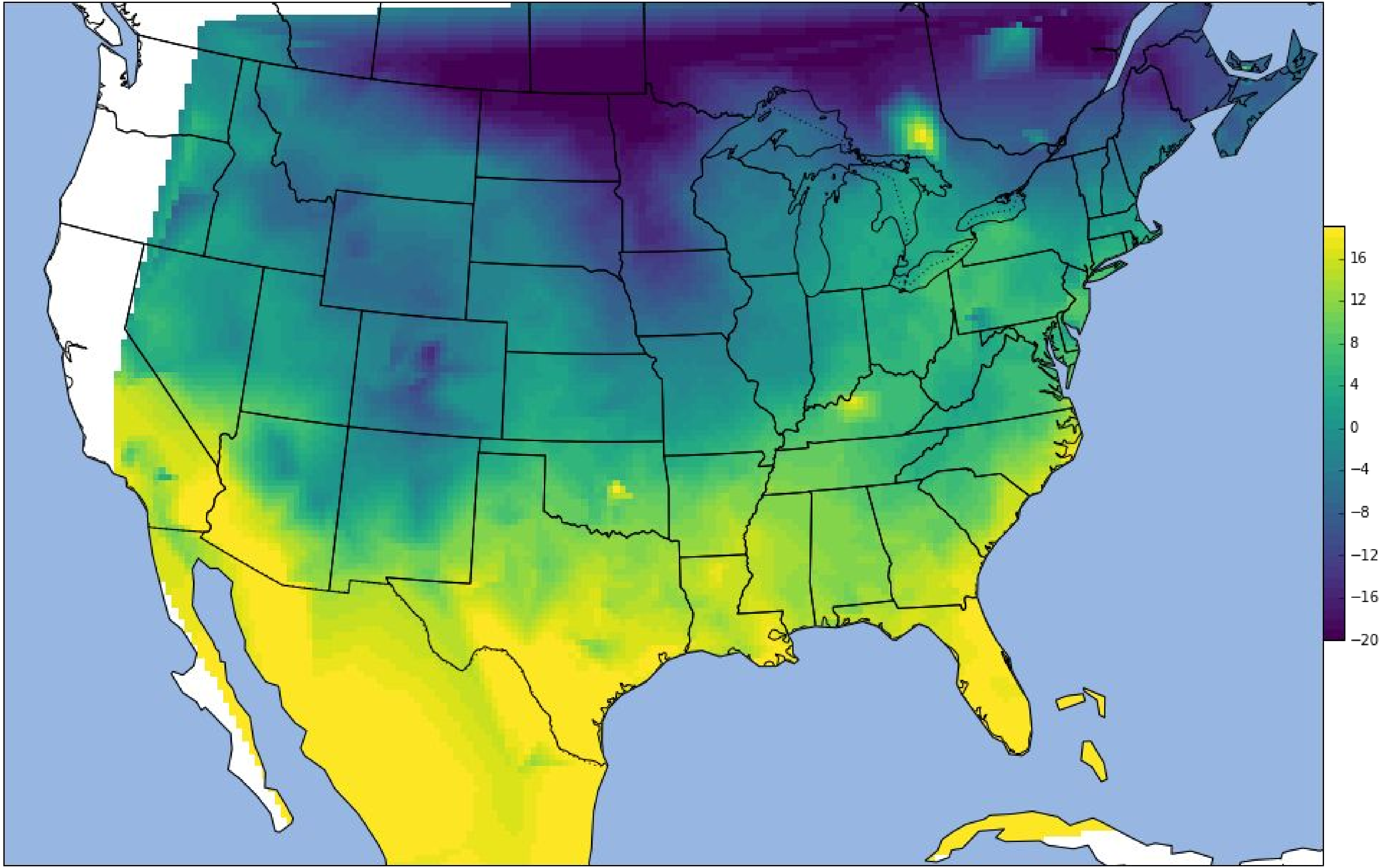
nearest



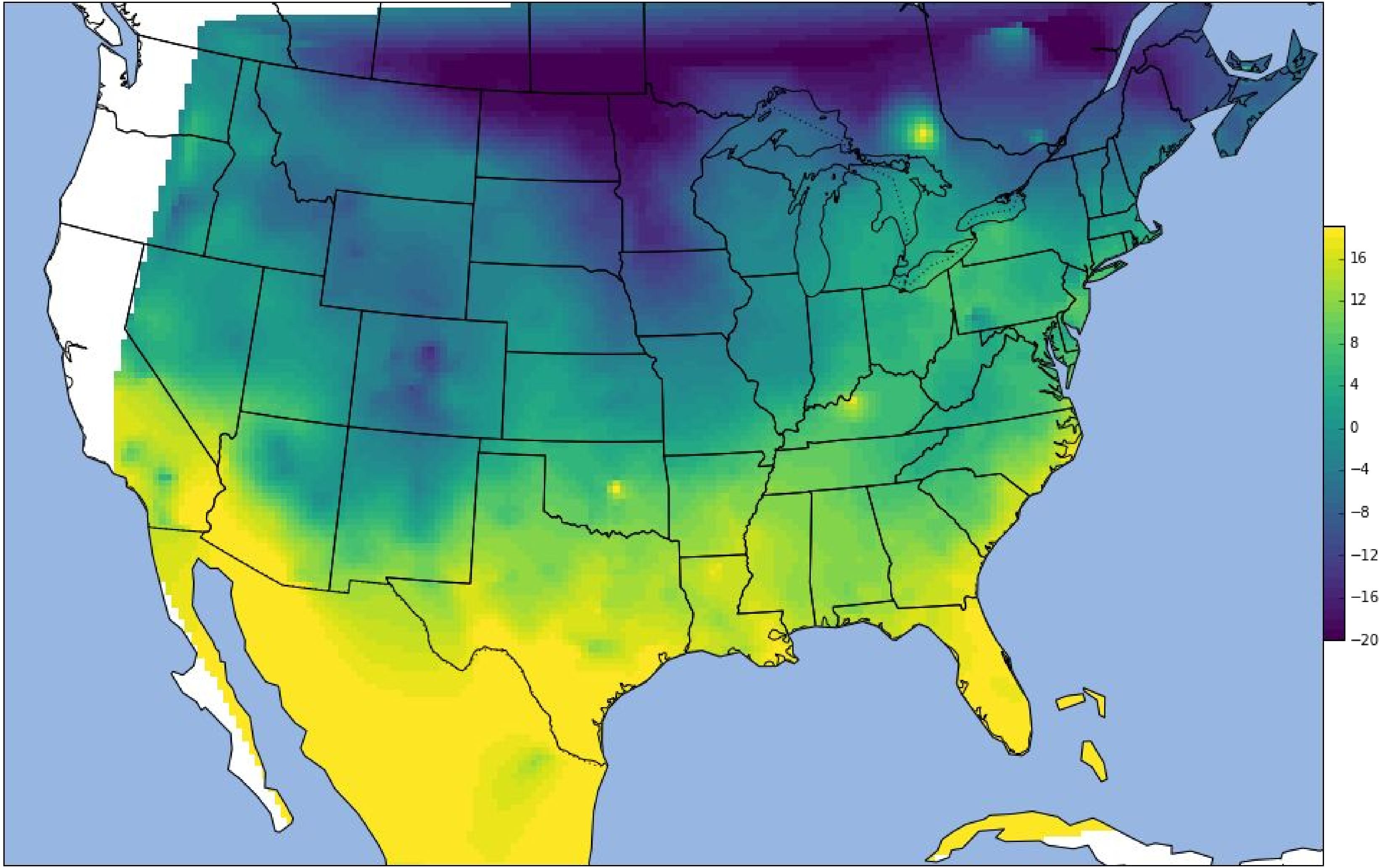
cubic

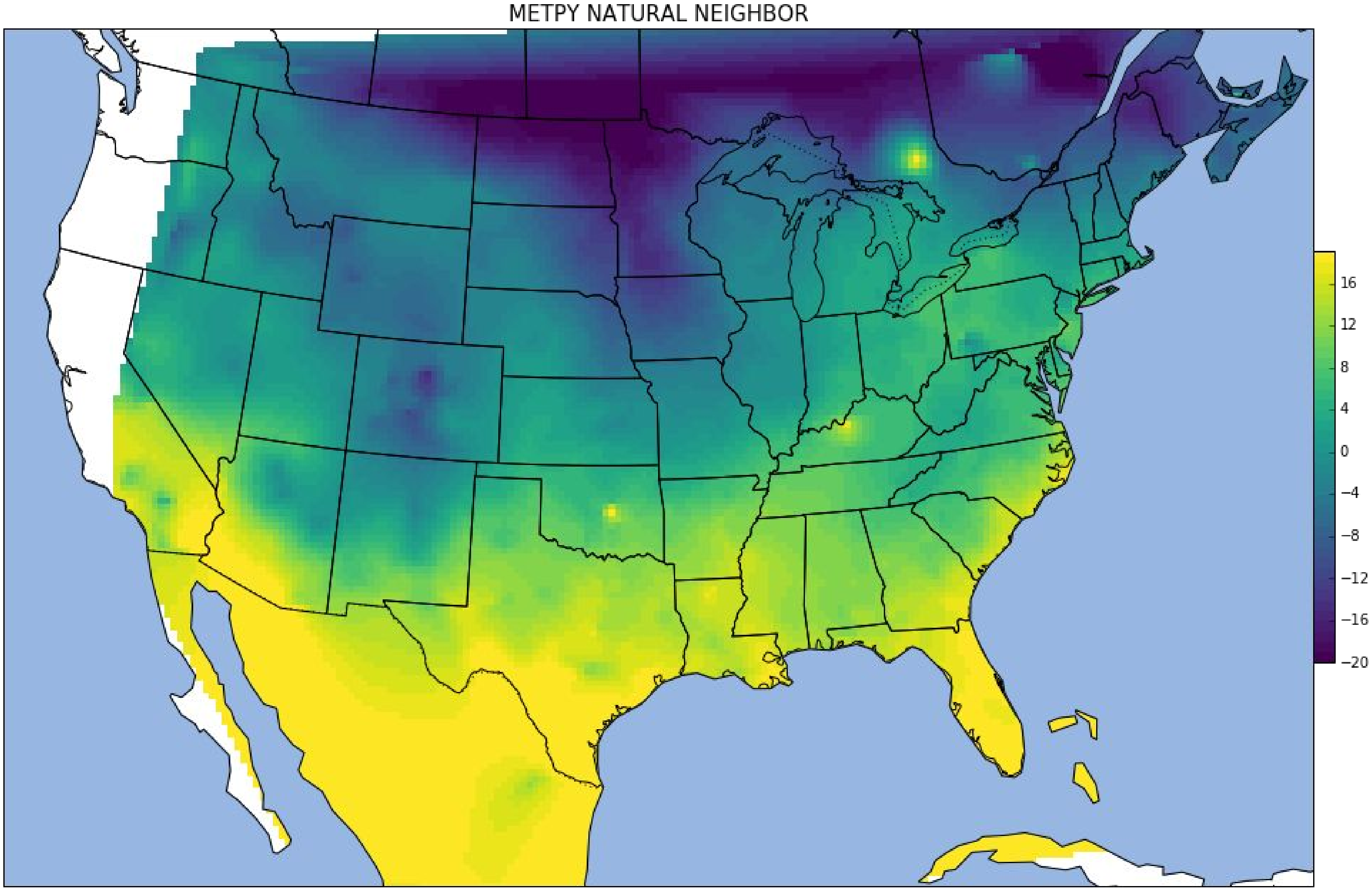


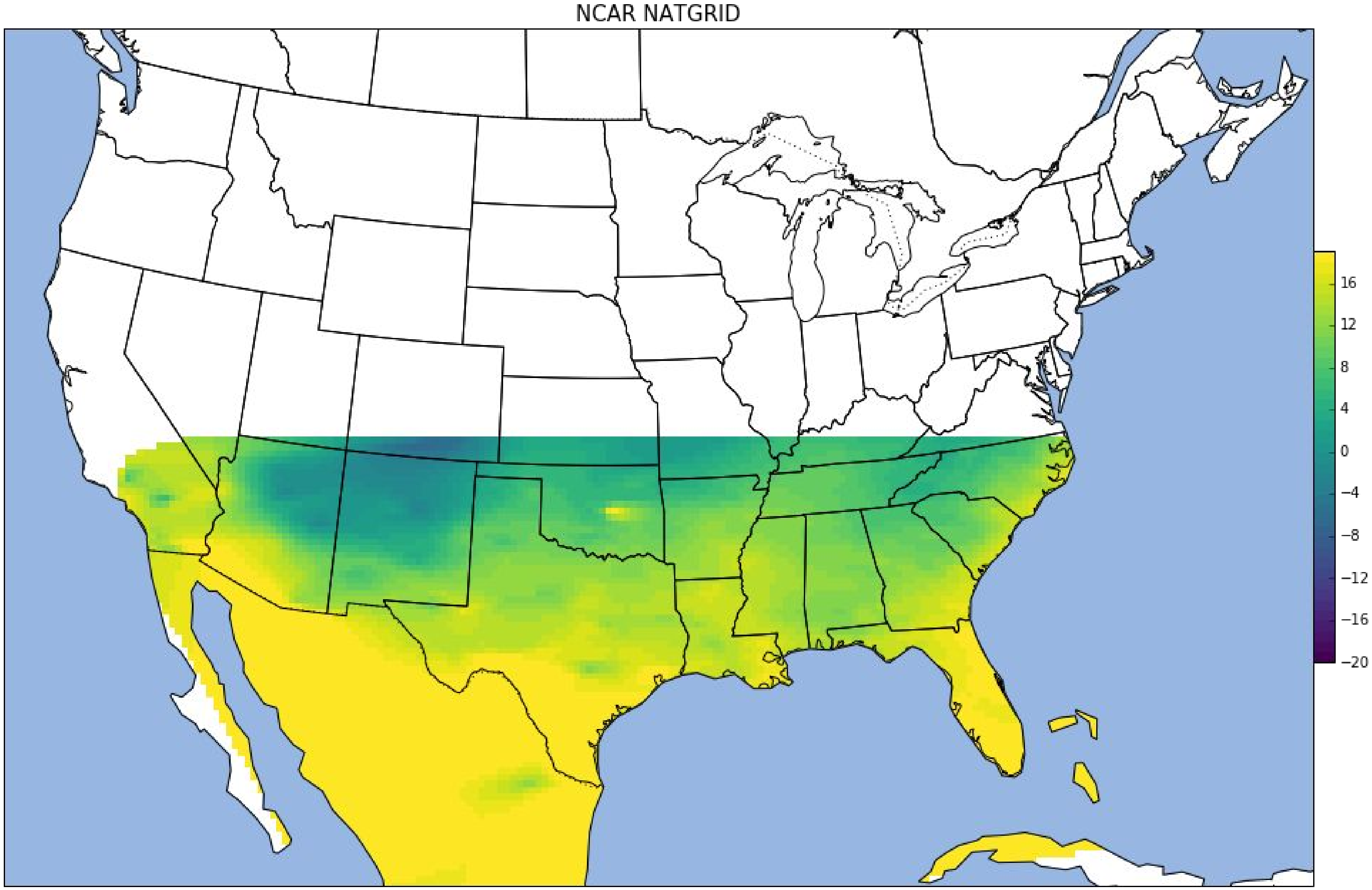
linear

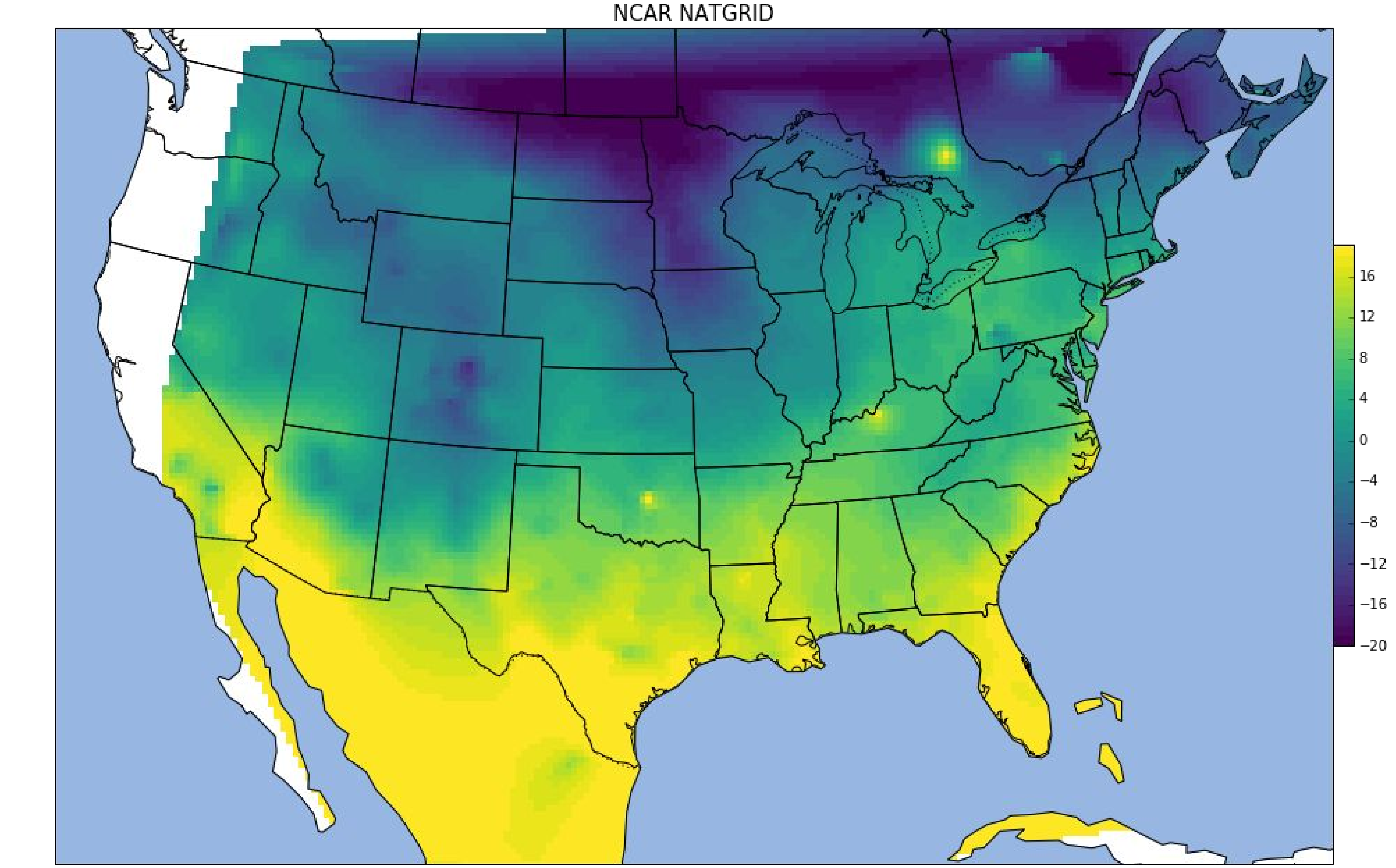


NCAR NATGRID

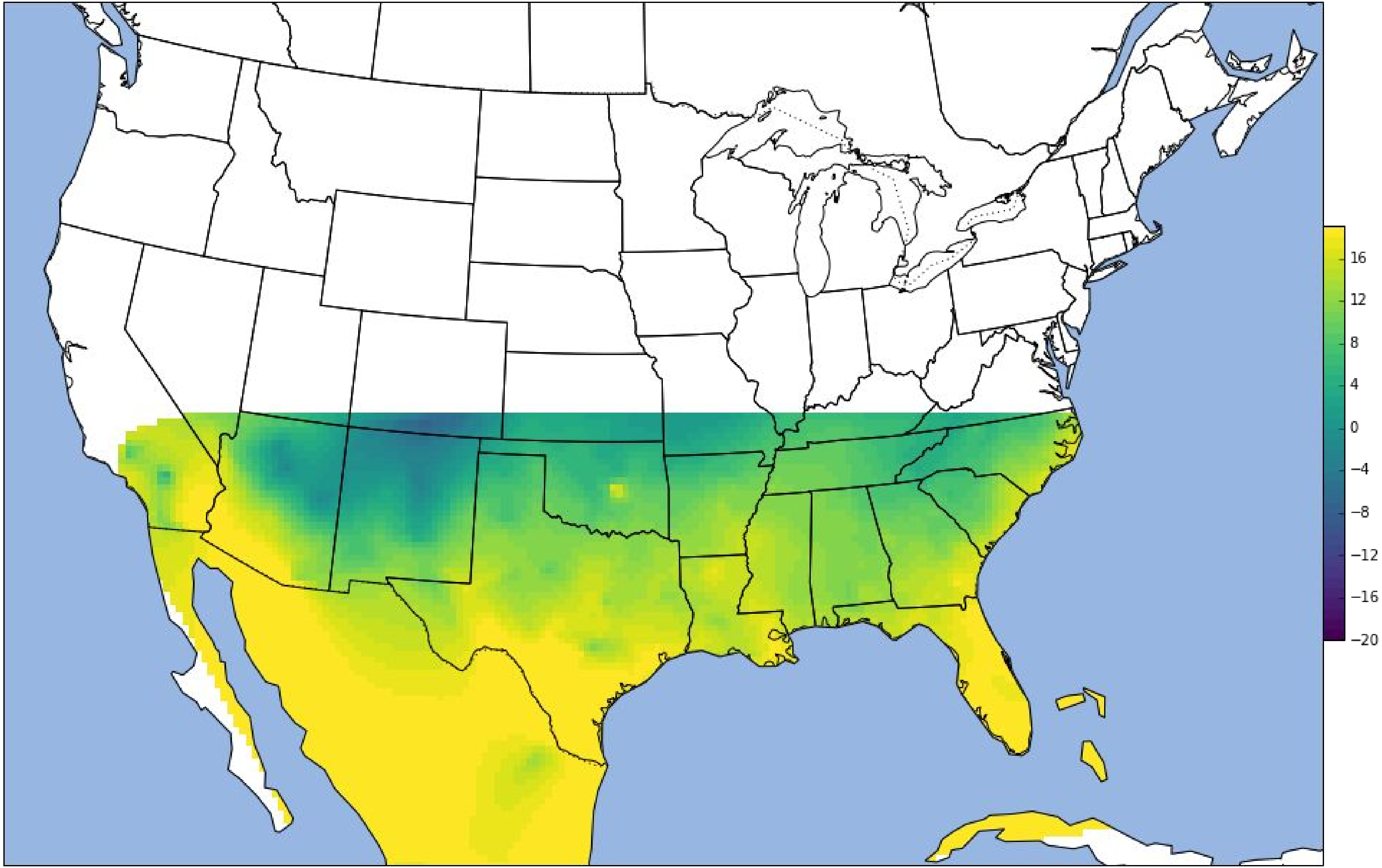


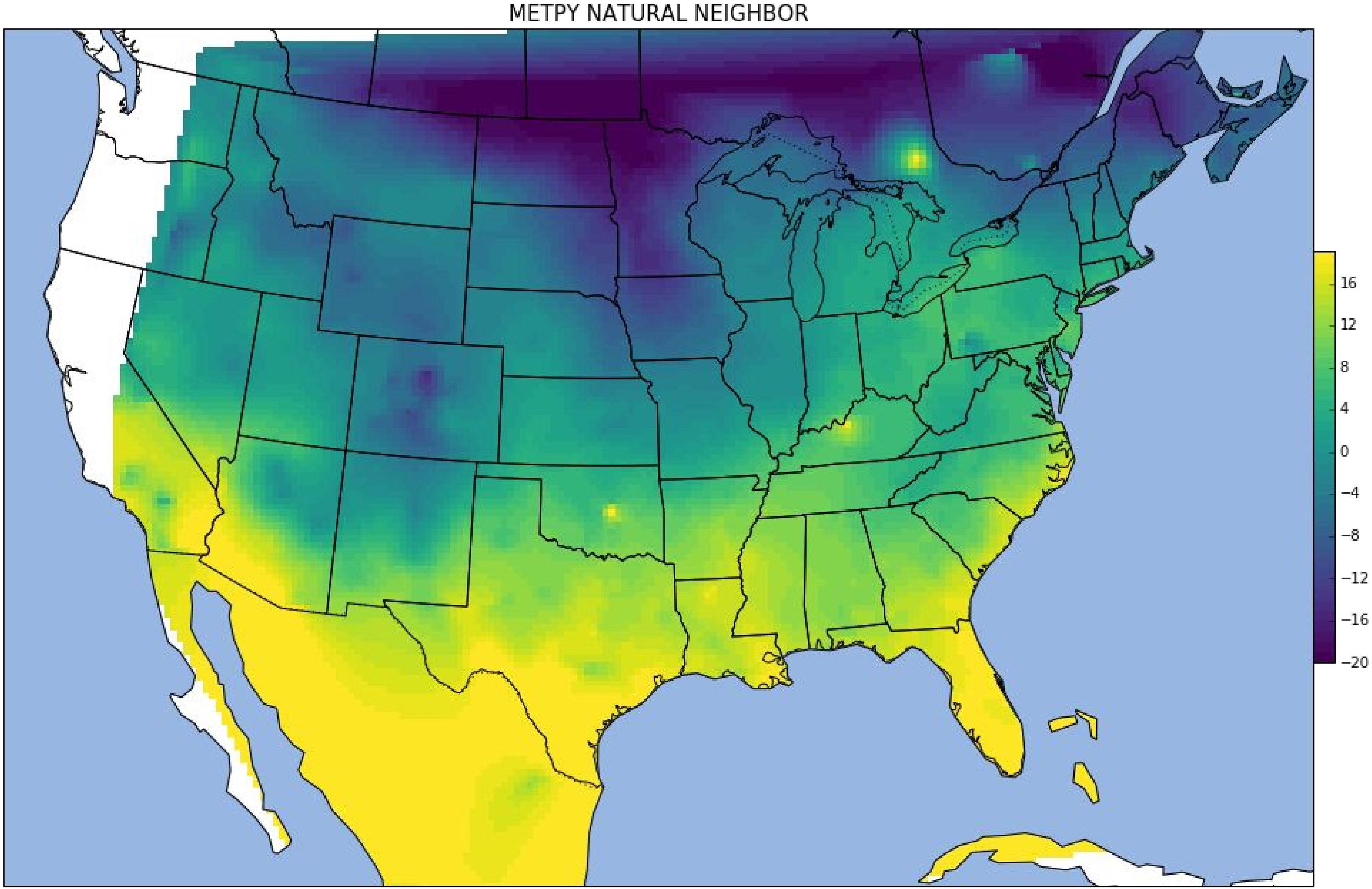

METPY NATURAL NEIGHBOR
16
12
8
4
0
−4
−8
−12
−16
−20


NCAR NATGRID
16
12
8
4
0
−4
−8
−12
−16
−20


NCAR NATGRID
16
12
8
4
0
−4
−8
−12
−16
−20

METPY NATURAL NEIGHBOR

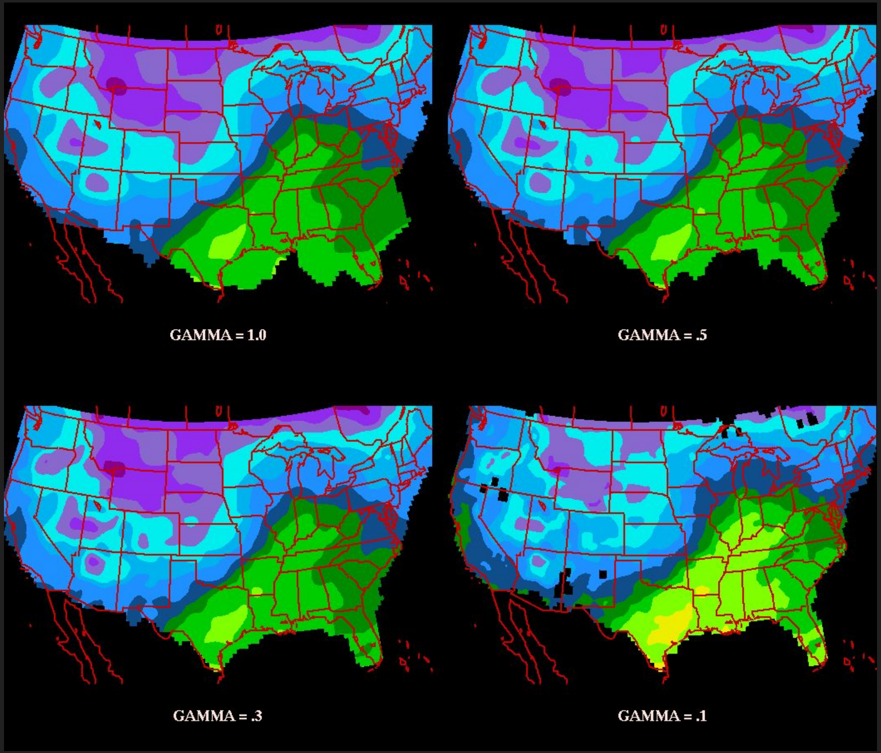




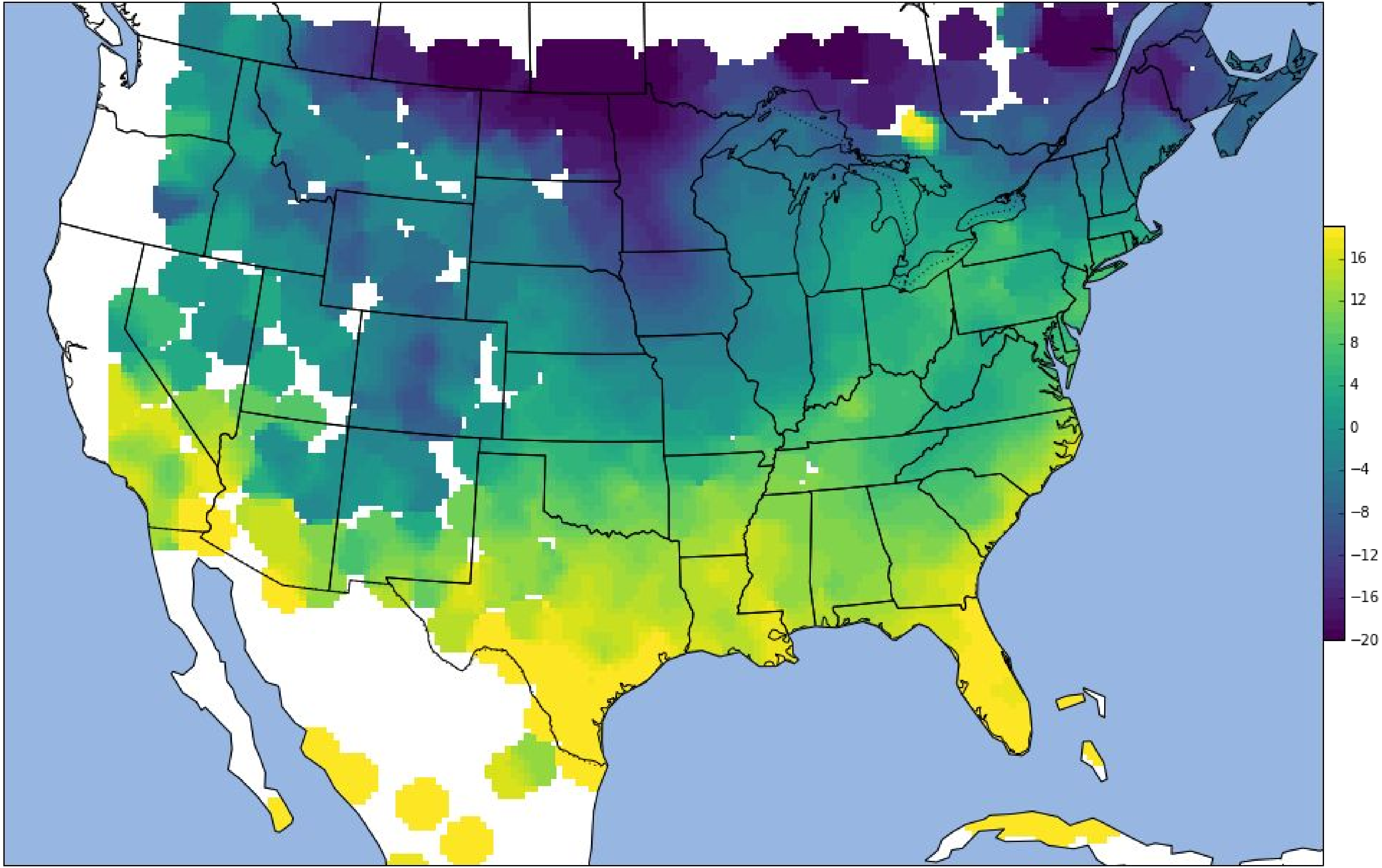
METPY NATURAL NEIGHBOR
16
12
8
4
0
−4
−8
−12
−16
−20

Step 2 of 4: Implement Barnes / Cressman

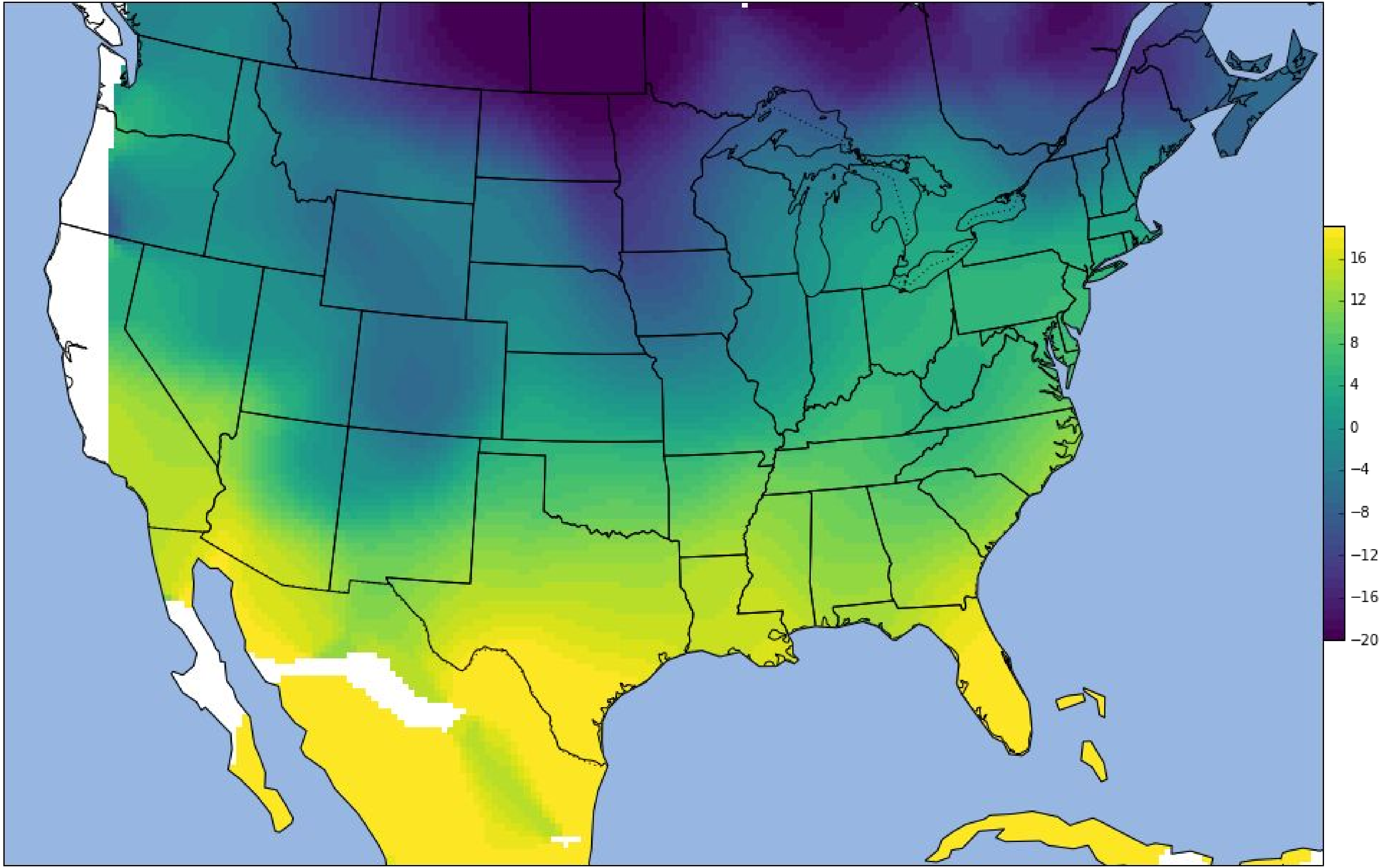
- Much easier to accomplish
  - Inverse distance weighting
- Barnes weight
  - $w = e^{(-\text{distances} / (\text{kappa} * \text{gamma}))}$
  - Distances between grid point and observations within given radius
  - Kappa is based on average distance between observations
  - Gamma is a smoothing parameter
- Cressman weight
  - w = (radius - distances) / (radius + distances)



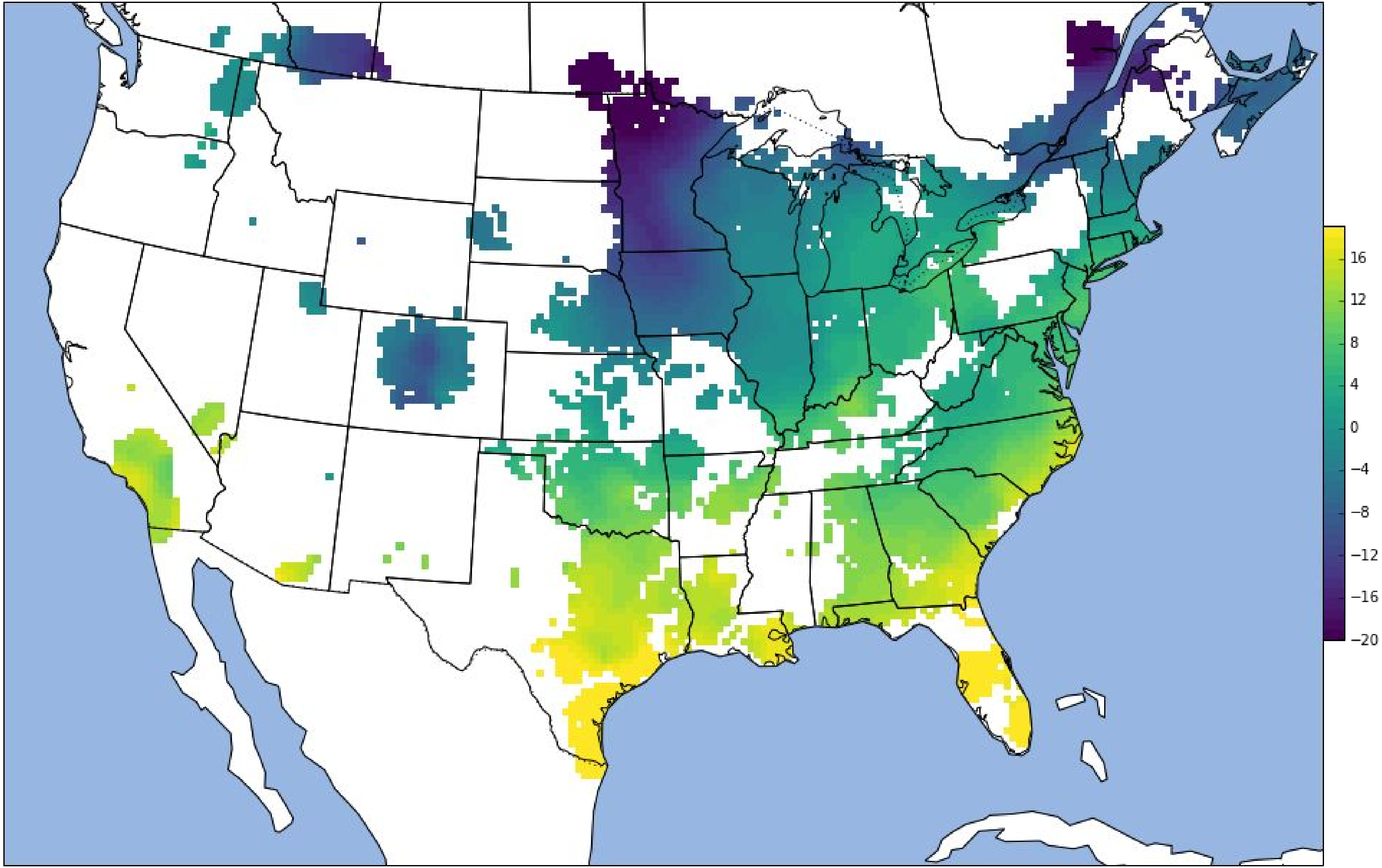
Cressman: search radius = 100km, Min Neighbors = 1, Cell size = 25 km



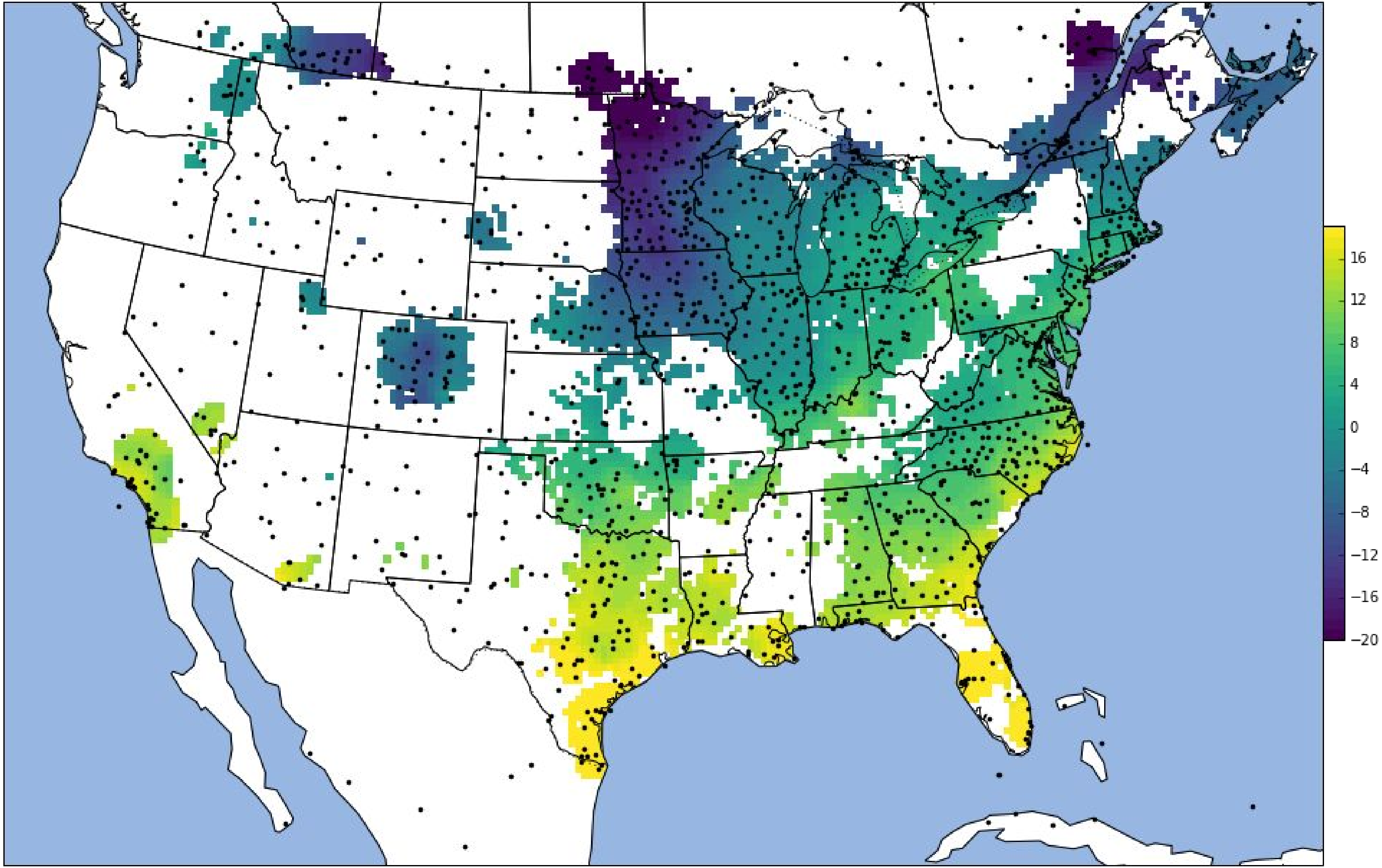
Cressman: search radius = 300km, Min Neighbors = 1, Cell size = 25 km



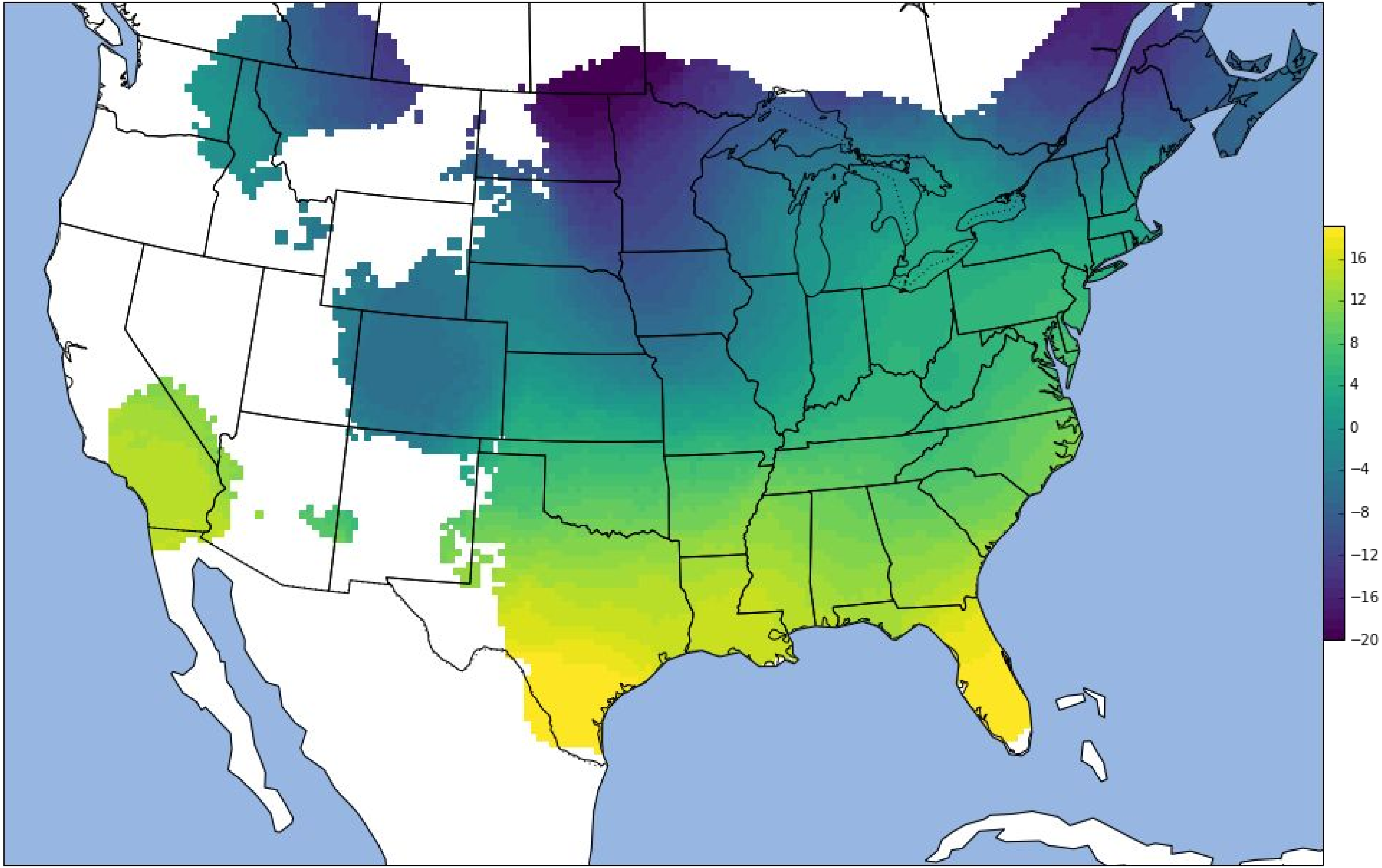
Cressman: search radius = 100km, Min Neighbors = 5, Cell size = 25 km



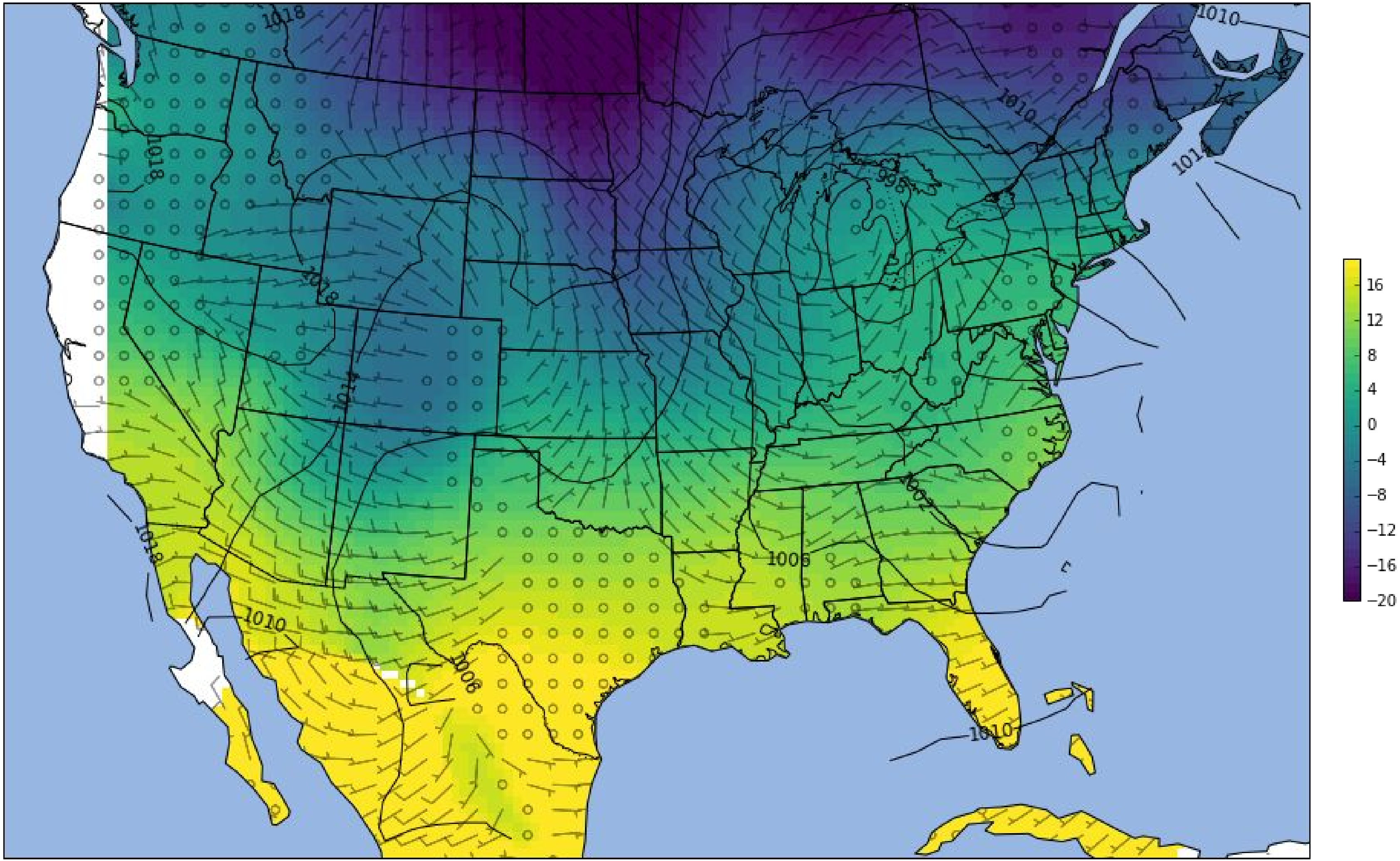
Cressman: search radius = 100km, Min Neighbors = 5, Cell size = 25 km



Barnes: search radius = 300km, Min Neighbors = 20, Cell size = 25 km



Surface Temperature (shaded), SLP, and Wind.



Step 3 of 4: Create user interface

- Build grids and interpolate points based on input data
  - Min / Max lat & lon values determine range
  - User defined or default grid cell sizes
  - More options for inverse distance schemes
    - Smoothing, Search Radius, etc.
- Return interpolated surface in correct shape, along with meshgrid/fishnet values that can quickly be thrown at matplotlib / cartopy / other visualization packages
- Make sure everything is tested
  - Somewhat difficult for 2D data!
  - As of this presentation, 98% test coverage
- https://github.com/ahaberlie/MetPy/tree/master/metpy/gridding
- https://github.com/ahaberlie/MetPy/blob/master/examples/notebooks/Point_Interpolation.ipynb
- https://github.com/ahaberlie/MetPy/blob/master/examples/notebooks/Wind_SLP_Interpolation.ipynb

Step 4 of 4: Create a mapping class*

- There is still a lot of code required to create maps
- Can we implement some GEMPAK-like functionality in Python?
  - Especially those affiliated with configuring map options
- [Traitlets config file]
- [Traitlets mapping class]

```
MAP      Map color/dash/width/filter flag
MSCALE   fgc;bgc;mask/units/lat;hide/values/anch/x;y/ln;wd/freq|text_info|t
GAREA    Graphics area
PROJ     Map projection/angles/margins|drop flag
SATFIL   Satellite image filename(s)
RADFIL   Radar image filename(s)
IMCBAR   Color/ornt/anch/x;y/ln;wd/freq
LATLON   Line color/dash/width/freq/inc/label/format
PANEL    Panel loc/color/dash/width/regn
TITLE    Title color/line/title
TEXT     Size/fnt/wdth/brdr/N-rot/just/hw flg
CLEAR    Clear screen flag
DEVICE   Device|name|x size;y size|color type
LUTFIL   Enhancement lookup table filename
STNPLT   Txtc/txt attr|marker attr|stnfil#col
VGFILE   Vgfile | scale file | attribute file | filter
AFOSFL   AFOS Graphics File
AWPSFL   AWIPS Graphics File
LINE     Color/type/width/label/smth/fltr/scflg
WATCH    End time|Wtch clrs|Wtch Tm;Status Ln Tm|Watch Num;
WARN     End time|TS;TN;FF clrs|Tm|Lb|Outline|Poly
HRCN     End time|colors|syms|Tm|Lb|Mt|Qw|F12|F24|F36|F48|F72|F96|F120|Name
ISIG     End time|colors|Sym|Tm|Id|Mv|F1
```

*Work in progress!

Step 5 of 4: Addressing performance issues

- Employ Cython to compile python-like syntax to C
- Was able to reduce the runtime by a factor of ~10 for some basic calculations
- Circumcircle radius
    - Before: 4.91 microseconds
    - After: 412 nanoseconds
- Circumcenter
    - Before: 2.3 microseconds
    - After: 271 nanoseconds
- Find Natural Neighbors:
    - Calls circumcenter and circumcircle radius
    - Reduced time by .3 milliseconds per iteration
- More work to be done!
    - Have fun Ryan
- https://github.com/ahaberlie/MetPy/blob/cythonize/examples/notebooks/Cython_demos.ipynb

Summary

- Created wrappers for scipy interpolation functions
  - Housekeeping is as invisible to the user as they want
- Implemented interpolation schemes in Python
  - Natural neighbor
  - Barnes
  - Cressman
- Developed a user interface for interpolating 2D data
  - High test coverage
  - 3D eventually?
- Began work on a GEMPAK-like mapping class
- Investigated performance enhancing cython

Thanks to everyone for the great summer!