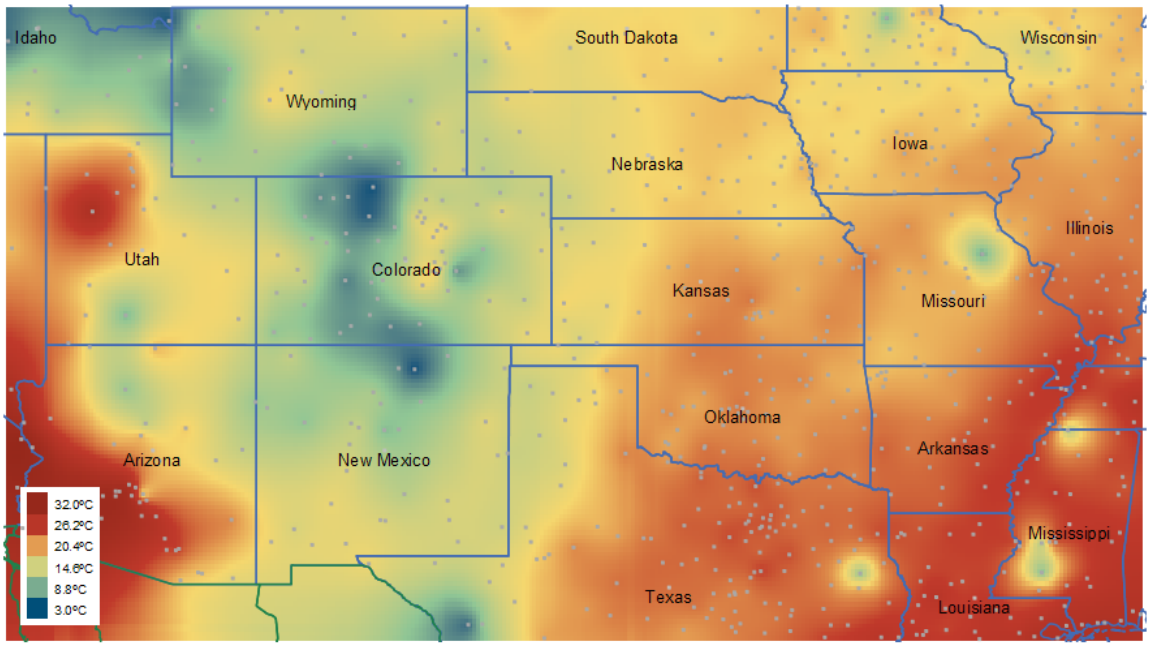
Smoothly Shaded Weather Map

Introduction

This application generates a smoothly shaded temperature map using data from weather stations scattered across the US.



The data (located in an Excel spreadsheet attached to this workbook) contains the longitude and latitude of each weather station, and the recorded temperature. This application uses Maple 2018's new Interpolation package to generate a regular grid of locations and temperatures from the irregular weather station data, emloying kriging interpolation.

The parameters Nx and Ny govern the resolution of the shading; higher values will produce smoother shading, but will take more processing time.

```
> restart:
  with(Interpolation):
  with(plots):
  with(plottools):
  with(DataSets):
  with(ColorTools):
  with(LinearAlgebra):
```

```
with(DocumentTools):
with(FileTools):
```

Parameters

Consider data only within these minimum and maximum values of longitude and latitude

```
> minX := -115:
  maxX := -88:
  minY := 30:
  maxY := 45:
```

Number of regularly spaced grid points interpolated from unstructured data (determines smoothness of shading)

```
> Nx := 100:
  Ny := 50:
```

Position of legend and number of bars

```
> legPosX := -114.5:
  legPosY := 30.5:
  nBars   := 6:
```

Import data

```
> data:=ImportMatrix("this:///temperatures.xlsx")
```

$$data := \begin{bmatrix} 2238 \text{ x } 5 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix} \quad (3.1)$$

Only keep weather stations between the minimum and maximum longitudes and latitudes

```
> k := 1:
  filteredStations := Matrix(2000,5):

  for i from 2 to RowDimension(data) do
     if data[i, 3] > minY and data[i, 3] < maxY and data[i ,4] >
  minX and data[i, 4] < maxX then
        filteredStations[k, ..] := data[i, ..]:
        k := k + 1:
     end if:
  end do:

  filteredStations := filteredStations[1 .. k, ..]:
```

Interpolate the irregular data onto a grid

Interpolating function

```
> f := Interpolate(<<filteredStations[.., 4] | filteredStations[.
  ., 3]>>, filteredStations[.., 5], method = kriging):
```

Generate regularly spaced longitude/latitude points and interpolate temperature onto the grid.

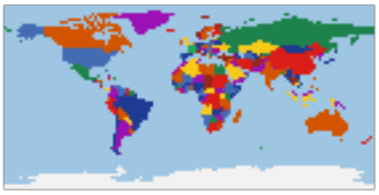
```
> deltaX := (maxX - minX) / Nx:
  deltaY := (maxY - minY) / Ny:

> M := Matrix(Nx * Ny, 3, datatype = float[8]):

> for i from 1 to Nx do
     for j from 1 to Ny do
        M[Ny * (i - 1) + j, 1] := minX + deltaX * (i - 1) + 0.5 *
  deltaX:
        M[Ny * (i - 1) + j, 2] := minY + deltaY * (j - 1) + 0.5 *
  deltaY:
        M[Ny * (i - 1) + j, 3] := f(minX + deltaX * (i - 1) + 0.5
  * deltaX, minY + deltaY * (j - 1) + 0.5 * deltaY):
        end do:
  end do:
```

Build the map

```
> m := Builtin:-WorldMap():
  SetProjection(m, Geographic)
```



A map of the world

projection: Geographic

```
> SetView(m, minX, minY, maxX, maxY):
```

Add state names

```
> ref    := Reference("Builtin", "Geonames"):
  places := ref[[Country = "United States", Type = "first-order
  administrative division"]]:
  AddPoints(m, places, symbol = solidcircle, symbolsize = 1, color
  = grey):
> USMap  := Display(m, mapdata = fine, thickness = 1, style =
  line):
```

Colors

```
> colorHighest  := Color("RGB", [150/255, 40/255,  27/255]):
```

```
colorVeryHigh := Color("RGB", [192/255, 57/255,  43/255]):
colorHigh     := Color("RGB", [245/255, 215/255, 110/255]):
colorMid      := Color("RGB", [144/255, 198/255, 149/255]):
colorLow      := Color("RGB", [0/255,   79/255,  121/255]):
```

Generate colors for shading

```
> Nt := 100:
  G  := [ Gradient(colorLow      .. colorMid,      number = Nt / 4
  - 2, best)[]
        ,Gradient(colorMid      .. colorHigh,     number = Nt / 4
  - 2, best)[]
        ,Gradient(colorHigh     .. colorVeryHigh, number = Nt / 4
  - 2, best)[]
        ,Gradient(colorVeryHigh .. colorHighest,  number = Nt / 4
  - 2, best)[]
       ]:
```

Generate a matrix of value-color pairs.
1st column is Nt equaly spaced values between the minimum and maximum interpolated values
2nd column is Nt colors

```
> valueColor := << Vector([seq((ceil(max(M[.., 3])) - floor(min(M
  [.., 3]))) / (Nt - 1) * (i - 1) + floor(min(M[.., 3])), i = 1 ..
  Nt)]) | Vector(G) >>:
```

Procedure to match a temperature to the closest color

```
> closestColorMatch := proc(foo)
     local match, i:
     match := 1:
     for i from 1 to Nt do
        if abs(foo - valueColor(i, 1)) <  abs(foo - valueColor
  (match, 1)) then
          match := i:
        end if:
     end do:
     return valueColor[match, 2]:
  end proc:
```

Generate the smooth temperature shading and plot the weather stations

Boxes whose (i) center-point is on the grid and (ii) a color that represents the temperature at that point

```
> boxes := seq( rectangle( [M[i, 1] - deltaX / 2, M[i, 2] - deltaY
  / 2]
                          ,[M[i, 1] + deltaX / 2, M[i, 2] + deltaY
  / 2]
                          ,style = surface, color =
  closestColorMatch(M[i,3])
                         )
              ,i = 1 .. Nx * Ny):
```

```
> stations := pointplot(<<filteredStations[.., 4] |
  filteredStations[.., 3]>>, annotation = filteredStations[.., 5],
  symbol = solidcircle, symbolsize = 1, color = "DarkGrey"):
```

Legend

```
> legColors := [valueColor[1,2], seq(valueColor[ceil(Nt / (nBars -
  1) * (i - 1)), 2], i = 2 .. nBars)]:
  legText   := [valueColor[1,1], seq(valueColor[ceil(Nt / (nBars -
  1) * (i - 1)), 1], i = 2 .. nBars)]:

> tempText := seq(textplot( [ legPosX + 0.55
                             ,(i - 1) * 0.5 + legPosY + 0.2
                             ,sprintf("%0.1f°C", legText[i])
                            ]
                            ,font = [Arial, 8], align = {right}
                          )
                  ,i = 1 .. nBars):
> colorBar := seq(rectangle( [legPosX, (i - 1) * 0.5 + legPosY]
                            ,[legPosX + 0.5, (i - 1) * 0.5 + 0.5 +
  legPosY]
                            ,color = legColors[i], style = surface

                           )
                  ,i = 1 .. nBars):
> legBack := rectangle([legPosX - 0.15, legPosY - 0.15], [legPosX
  + 0.5 + 1.25, legPosY + 0.15 + nBars * 0.5], color = white,
  style = surface):
```

Display the map

```
> weatherMap := JoinPath([TemporaryDirectory(), "weathermap.png"]
  ):
```

Write the plot out to a temp file, and wait 1 second

```
> plotsetup(png, plotoutput = weatherMap)
> display(colorBar, legBack, tempText, USMap, boxes, stations,
  view = [minX .. maxX, minY .. maxY], size = [1000, 500], axes =
  none, font = [Arial, 10])
> endtime := time() + 1:
  while time() < endtime do end do;
```

Display the weather map on a label component

```
> SetProperty(weatherMap_lb, image, weatherMap):
>
```

Weather Map