

# Using R to View OPeNDAP Files in the NCAR/EOL Field Data Archive

Updated: 11 April 2022

**Authorship:** NCAR Earth Observing Laboratory Data Management & Services Group

**Contact:** [eol-archive@ucar.edu](mailto:eol-archive@ucar.edu)

**Licensing:** The code associated with this document is provided freely and openly. Users are hereby granted a license to access and use this code, *unless otherwise stated, subject to the terms and conditions of the GNU Affero General Public License 3.0 (AGPL-3.0;* <https://www.gnu.org/licenses/agpl-3.0.en.html>). This documentation and associated code are provided “as is” and are not supported. By using or downloading this code, the user agrees to the terms and conditions set forth in this document.

**Acknowledgment:** This work was sponsored by the National Science Foundation. This material is based upon work supported by the National Center for Atmospheric Research, a major facility sponsored by the National Science Foundation and managed by the University Corporation for Atmospheric Research. Any opinions, findings and conclusions or recommendations expressed in this material do not necessarily reflect the views of the National Science Foundation.

## General Description

This document serves as a guide on how R can be used with OPeNDAP within the EOL Field Data Archive to plot variables found in files with OPeNDAP access without needing to download the files. Two examples of R plotting scripts can be found in the files “**R\_xy\_plot.R**” and “**R\_xyz\_plot.R**”. The scripts were created by NCAR EOL and show how 2D and 3D plots can be created in R, respectively. Note that these scripts are provided “as is” and are not supported.

## OPeNDAP within the EOL Field Data Archive

The EOL Field Data Archive (FDA) contains datasets that have OPeNDAP compatibility. An example dataset in the FDA would be GCIP/ESOP-95 Surface: Hourly Surface Meteorological Composite. This dataset was used to generate the plots in the sections below.

For this dataset, 2D plots were created for a site (Ashton) for Temperature and Dew Point using script **R\_xy\_plot.R**. A 3D plot was also created using **R\_xyz\_plot.R** that shows Temperature at a specific time over the entire area of interest in the file.

## **How to use the R Plot tools with the OPeNDAP Capabilities in the EOL Field Data Archive**

### **Choose a dataset in the EOL FDA that has OPeNDAP Capabilities**




Figure 1 shows the top portion of the EOL FDA dataset description page for the GCIP/ESOP-95 Surface: Hourly Surface Meteorological Composite dataset. The OPeNDAP feature is located in the “Data access” section of the dataset page. Clicking on the “OPeNDAP access” link will show the OPeNDAP page for the dataset. This page will show a list of all the files in the dataset and their corresponding OPeNDAP links. If a dataset in the EOL FDA does not have an OPeNDAP link in the “Data access” section, then the dataset does not have OPeNDAP capabilities. The user must then download the data (via the Order link) and apply the provided R scripts to plot the data.

### **GCIP/ESOP-95 Surface: Hourly Surface Meteorological Composite**

**Summary**

The GCIP/ESOP-95 Hourly Surface Composite contains data from several networks (i.e., Artais Automated Weather Observation System, Handar AWOS, and Qualimetrics AWOS, Oklahoma Mesonet, Department Of Energy Atmospheric Radiation Measurement Surface, High Plains Climate Network, Automated Surface Observing System, Wind Profiler Network, national Climatic Data Center Surface Airways Observations, and Colorado Agricultural Meteorological data) for the ESOP 95 domain. Data from these sources were merged and quality controlled to form this Surface Composite.

**Data access**

-  [ORDER](#) data for delivery by FTP
-  [Preview](#) dataset (plots/images)
-  [OPeNDAP access](#)

**Figure 1:** Top portion of the dataset description page for the EOL GCIP/ESOP-95 Surface: Hourly Surface Meteorological Composite dataset. The OPeNDAP feature for datasets (if applicable) is located in the “Data access” section of the dataset description page.

### **Get the Link to the File for Plotting**

Figure 2 shows the “OPeNDAP access” page for the EOL GCIP/ESOP-95 Surface: Hourly Surface Meteorological Composite dataset. This page lists all files in the dataset. From here the OPeNDAP link files can be found. Clicking the “scissors” symbol will show the OPeNDAP link for a specific file. This link can then be fed into the R scripts for plotting. For the plots shown in this document, file “ES95HRLY\_950715.qcf” was used and the OPeNDAP link for that file is:

[https://data.eol.ucar.edu/opendap/data/esop\\_95/hrly\\_sfc/ES95HRLY\\_950715.qcf](https://data.eol.ucar.edu/opendap/data/esop_95/hrly_sfc/ES95HRLY_950715.qcf)

## GCIP/ESOP-95 Surface: Hourly Surface Meteorological Composite

Back to [dataset homepage](#)

[OPeNDAP](#) access links are available below. The "Webform" links provide an HTML web interface to the OPeNDAP data protocol and is useful for limited textual review of the data file. The "DAP" links provide access for OPeNDAP-enabled software applications. You cannot access the DAP link via a web browser or standard HTTP. Right-click the DAP link or use the scissors icon to copy the link for pasting into your application.

See the [EOL data archive OPeNDAP help](#) page for more info.

183 files

Max results: 100

File info	Download	OPeNDAP
<a href="#">ES95HRLY_950401.qcf</a>	1911 KiB	<a href="#">Webform</a> <a href="#">DAP</a> ✂ <div> <a href="https://data.eol.ucar.edu/opendap/data/esop_95/h">https://data.eol.ucar.edu/opendap/data/esop_95/h</a></div>
<a href="#">ES95HRLY_950402.qcf</a>	2 MiB	<a href="#">Webform</a> <a href="#">DAP</a> ✂
<a href="#">ES95HRLY_950403.qcf</a>	2 MiB	<a href="#">Webform</a> <a href="#">DAP</a> ✂

**Figure 2:** The OPeNDAP page for the EOL GCIP/ESOP-95 Surface: Hourly Surface Meteorological Composite dataset. Clicking on the “scissors” symbol for a file will show the OPeNDAP link for that specific file.

### Store the Data File Link in the R script

In the R script, this link to the data file should be stored in a variable as shown in Figure 3 below. Figure 3 shows how the file link is stored in the provided sample scripts. (Note: All of the R figures come from the sample script “R\_xy\_plot.R”.)

```
39 # Store the URL for the files in a variable.
40
41 url <- "https://data.eol.ucar.edu/opendap/data/esop_95/hrly_sfc/ES95HRLY_950715.qcf"
42
43 # Open the file.
44
45 nc <- nc_open(url)
46
47 # Store the info in the file into a text file for the viewer to read.
48
49 {
50     sink('metadata.txt')
51     print(nc)
52     sink()
53 }
```

**Figure 3:** Line 41 shows the URL for the file being stored into the variable “url”. Line 45 opens the file. Lines 49-53 saves the file locally on the machine. The user can then use this text file to view the information on the file and can see the variables in the file.

```

55 # Save the date and time into variables.
56
57 date <- ncvar_get(nc, "QCF.date_nominal")
58 time <- ncvar_get(nc, "QCF.time_nominal")
59
60 # Take the date and time and create the datetime.
61
62 datetime <- paste(date, time, sep=" ")
63
64 # Only keep the unique datetimes.
65
66 unique_datetime <- unique(datetime)
67
68 # Load other variables to plot.
69
70 dew_point <- ncvar_get(nc, "QCF.dew_point_temperature")
71 air_temp <- ncvar_get(nc, "QCF.air_temperature")
72
73 network <- ncvar_get(nc, "QCF.network_name")
74 station <- ncvar_get(nc, "QCF.platform_name")
75
76 # Combine the network and station and create a location.
77
78 location <- paste(network, station)
79
80 unique_location <- unique(location)
81 unique_location <- sort(unique_location)
82
83 # Save the locations into a text file for the viewer to see.
84
85 {
86     sink('location_list.txt')
87     print(unique_location)
88     sink()
89 }

```

**Figure 4:** This figure shows how the variables in the file are used in the R script in lines 55-74. Line 62 takes the date and time and combines them to make datetimes and takes the unique datetime in line 66. Lines 76-81 take the network and station and combine them to make one location. Lines 83-89 shows the script saving the list of locations into a text file for the user to review.

Figure 4 shows how to save the data from the file. The command “ncvar\_get” is used to save the data from the .qcf file. The line below shows the format of this command.

```
ncvar_get(nc, Variable Tag)
```

The “Variable Tag” in the line above can be determined from the metadata saved as shown in Figure 3.

```
48 variables (excluding dimension variables):  
  char QCF.date_nominal[maxStrlen64,QCF]  
    Description: UTC Nominal Date of Observation (YYYY/MM/DD)  
  char QCF.time_nominal[maxStrlen64,QCF]  
    Description: UTC Nominal Time of Observation (HH:mm:ss)
```

**Figure 5:** This figure shows how variables are formatted in the metadata file created by the script as shown in Figure 3. The two variables shown in figure 5 are the nominal date and time. The description gives more information about the variables. The variable tag used in the “ncvar\_get” command is next to the variable type. In figure 5, the tags are “QCF.date\_nominal” and “QCF.time\_nominal”. The tags come after the variable type “char”.

```

91 # For this example, the first location is used for plotting.
92
93 example_location <- unique_location[1]
94
95 # Take the number of times and create an array of the same size to create an
96 # array to store the indices of repeating locations.
97
98 time_size <- nrow(time)
99 inc_array <- rep(0, nrow(time))
100
101 # Find the indices where the location repeats.
102
103 inc <- 1
104
105 for (i in 1:time_size){
106     current_station <- location[i]
107     if (current_station == example_location){
108         inc_array[i] <- i
109         inc <- inc + 1
110     }
111 }
112
113 }
114
115 }
116 }

```

**Figure 6:** Line 93 shows the script using the first location in the list of locations. The user can specify any desired location and replace the location used above. Line 98 saves the length of the nominal time of the file or the number of nominal times shown in the file. This amount is equal to the number of lines in the file. Line 99 is the variable used to store the indices of where the desired location is found. Lines 103-116 show how the indices are found and saved.

Figure 6 shows the process of finding where the indices of a given location are. An index corresponds to the line in the file where the desired location is found. The “for” loop searches the entire file and finds where the desired location is and stores the index for later use.

```
118 # Take the array of indices and remove the 0's.  
119  
120 final_array <- inc_array[inc_array != 0]  
121  
122 # Use the line commented below to see the indices of the location.  
123  
124 # print(final_array)  
125  
126 # Create arrays to store the temp of the given location.  
127  
128 final_dew_temp <- rep(0, length(final_array))  
129 final_temp <- rep(0, length(final_array))  
130
```

**Figure 7:** Line 120 takes the array that stored the indices and removes the 0's. The array was originally populated with 0's so these values must be removed. The commented line 124 can be used to print the indices for the user to review. Lines 128 and 129 create arrays to save the temperatures for the location later on.

```

131 # For this file, the data contains -999 values. Here these
132 # values are set to NaN so they will not be plotted.
133
134 for (k in 1:length(final_array)){
135
136     if (air_temp[final_array[k]] <= -100){
137
138         final_temp[k] <- NaN
139
140     }
141
142     if (air_temp[final_array[k]] > -100){
143
144         final_temp[k] <- air_temp[final_array[k]]
145
146     }
147
148     if (dew_point[final_array[k]] <= -100){
149
150         final_dew_temp[k] <- NaN
151
152     }
153
154     if (dew_point[final_array[k]] > -100){
155
156         final_dew_temp[k] <- dew_point[final_array[k]]
157
158     }
159
160 }

```

**Figure 8:** This shows a for loop that goes through all the temperature arrays. The file contains some -999 values or missing values. These values are changed to NaN values so they will not be plotted. The "for" loops form pairs of indices and temperature values and save those pairs to specific air\_temp and dew\_point arrays.



```

168 # Create the data frame for plotting.
169
170 df <- data.frame(
171   df_datetime = unique_datetime,
172   df_air_temp = final_temp,
173   df_dew_temp = final_dew_temp
174 )
175
176 # Plot the data.
177
178 ggplot(data = df, aes(x = df_datetime, group = 1)) +
179   geom_line(aes(y = df_air_temp, color = "darkred")) +
180   geom_line(aes(y = df_dew_temp, color = "steelblue")) +
181   geom_point(aes(y = df_air_temp, color = "darkred")) +
182   geom_point(aes(y = df_dew_temp, color = "steelblue")) +
183
184 # Adjust labels, title, legend, etc.
185
186 xlab(" ") +
187 ylab("Temperature (Celsius)") +
188 labs(title=paste("GCIP/ESOP-95 Hourly Surface: ", unique_location, sep=""), color = "Legend") +
189 scale_color_hue(labels = c("Air Temp", "Dew Point Temp")) +
190 theme(plot.title = element_text(hjust = 0.5)) +
191 theme(axis.text.x = element_text(angle = 90))
192
193 # Save the file as a .png.
194
195 ggsave("test.png")

```

**Figure 9:** The data frame is created in lines 168-176 for plotting. A data frame is a table and is used to format the data for plotting. Lines 180-184 create the lines and points shown on the plot. Lines 188-193 create the x and y labels, the title, the legend, and other features of the plot. See Figure 10 below.

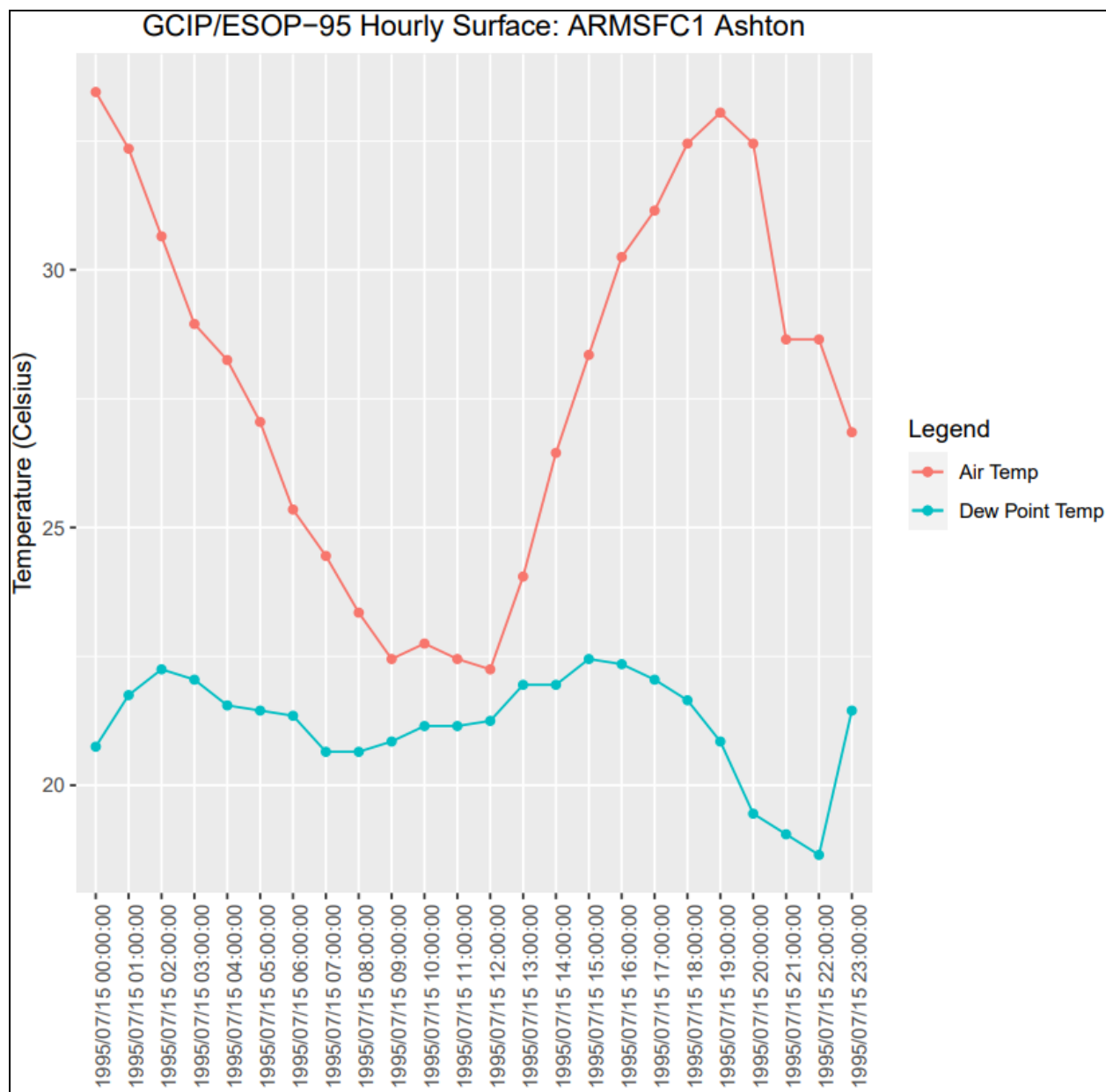


Figure 10: The final 2D plot.

### 3D Plotting using the “R\_xyz\_plot.R” Script

```
39 # Create a variable and store the URL of the file.
40
41 url <- "https://data.eol.ucar.edu/pendap/data/esop_95/hrly_sfc/ES95HRLY_950715.qcf"
42
43 # Open the file.
44
45 nc <- nc_open(url)
46
47 # Store the contents of the files into a text file.
48
49 {
50     sink('metadata.txt')
51     print(nc)
52     sink()
53 }
54
55 # Store the location, temp, and other desired info into variables.
56
57 lat <- ncvar_get(nc, "QCF.latitude")
58 long <- ncvar_get(nc, "QCF.longitude")
59
60 dew_point <- ncvar_get(nc, "QCF.dew_point_temperature")
61 air_temp <- ncvar_get(nc, "QCF.air_temperature")
62
63 date <- ncvar_get(nc, "QCF.date_nominal")
64 time <- ncvar_get(nc, "QCF.time_nominal")
65
66 # Take the date and time and create the datetime.
67
68 datetime <- paste(date, time, sep=" ")
69
70 # Find the unique time and datetime.
71
72 unique_datetime <- unique(datetime)
73 unique_time <- unique(time)
```

**Figure 11:** The 3D script starts in a way similar to the 2D script. The 3D script stores the URL of the file with the data to be plotted, saves a metadata file, stores the desired variables and creates the datetimes information.

```

75 # This example takes the first time in the file.
76
77 test_time <- 1
78
79 # Save the desired time into variable.
80
81 t <- unique_time[test_time]
82
83 # Create an array to store the line numbers where the desired time is located.
84
85 array_time <- rep(0, length(date))
86
87 # Find the indices of the desired time and store it in the array.
88
89 for (i in 1:length(date)){
90     if (t == time[i]){
91         array_time[i] <- t
92     }
93 }
94
95 }
96
97 }

```

**Figure 12:** For this 3D example, the first saved time from the data is selected (line 77) and stored (line 81). Line 85 creates an array to store the indices where data (from multiple stations) exists at this time throughout the data file. The “for” loop in lines 89-97 finds the indices where this specific time is found in the data.

```

99 # Create a new array that does not have the 0s in 'array_time'.
100
101 final_array_time <- array_time[array_time != "0"]
102
103 # Create variables that go with the indeces found eariler to store the info.
104
105 final_dew_temp <- rep(0, length(final_array_time))
106 final_temp <- rep(0, length(final_array_time))
107 final_latitude <- rep(0, length(final_array_time))
108 final_longitude <- rep(0, length(final_array_time))
109
110 # Find the info of desired variables for the given location at the given
111 # time and store the info into the variables.
112
113 ind <- 1
114
115 for (i in 1:length(final_array_time)){
116     if (t == time[i]){
117
118         final_latitude[i] <- lat[i]
119         final_longitude[i] <- long[i]
120         final_dew_temp[i] <- dew_point[i]
121         final_temp[i] <- air_temp[i]
122         ind <- ind + 1
123
124     }
125 }
126
127 }

```

**Figure 13:** Line 101 removes the zero values from the array as the array was initialized with zeros. Lines 105-108 create arrays to store the location and temperature values later on. Lines 113-127 show the “for” loop used to find the location and temperature values corresponding to the time indices located before.

```
129 # The file contains -999 values. Setting these values to NaN allows
130 # the script to ignore plotting these values.
131
132 for (i in 1:length(final_dew_temp)){
133     if(final_dew_temp[i] < -100){
134         final_dew_temp[i] <- NaN
135     }
136 }
137
138 for (i in 1:length(final_temp)){
139     if(final_temp[i] < -100){
140         final_temp[i] <- NaN
141     }
142 }
143
144 }
```

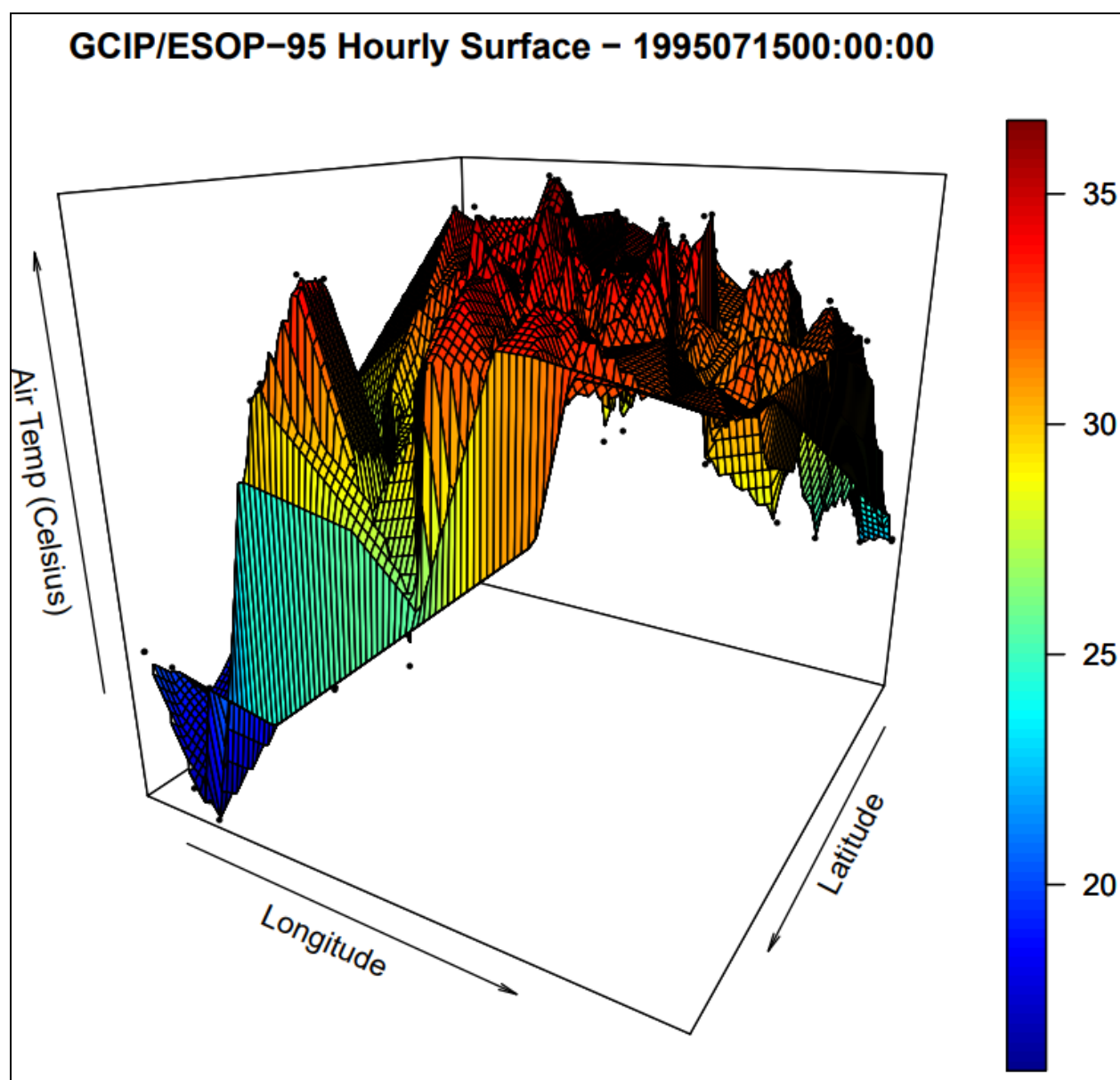
**Figure 14:** The input file contains -999 values for missing temperatures. These values are set to NaN so these values will be ignored during plotting.

```

152 # Create a data frame to plot.
153
154 data_grid <- data.frame(
155
156     data_col = c(final_temp),
157     axis1 = c(final_latitude),
158     axis2 = c(final_longitude)
159 )
160
161 # Remove any lines that contain missing values in the data frame.
162
163 data_grid_omit <- na.omit(data_grid)
164
165 # Create a matrix from the data frame to plot.
166
167 mat = matrix(final_temp, nrow=length(final_latitude),
168     ncol=length(final_longitude))
169
170 # Use the interp function to interpolate the grid.
171
172 grid <- interp(data_grid_omit$axis1, data_grid_omit$axis2,
173     data_grid_omit$data_col, duplicate="strip", nx = 100, ny = 100)
174
175 # Use the mesh to create the grid.
176
177 M <- mesh(grid$x, grid$y)
178
179 surf3D(M$x, M$y, grid$z, xlab="Latitude", ylab="Longitude",
180     zlab="Air Temp (Celsius)",
181     main=paste("GCIP/ESOP-95 Hourly Surface - 19950715", t, sep=""),
182     colvar = grid$z, colkey = TRUE,
183     box = TRUE, bty = "b", phi = 20, theta = 120)
184 #, border = "black")
185
186 points3D(data_grid_omit$axis1, data_grid_omit$axis2, data_grid_omit$data_col,
187     data_grid_omit$data_col, add=TRUE, pch=20, cex=0.5, col = "black", size=4)

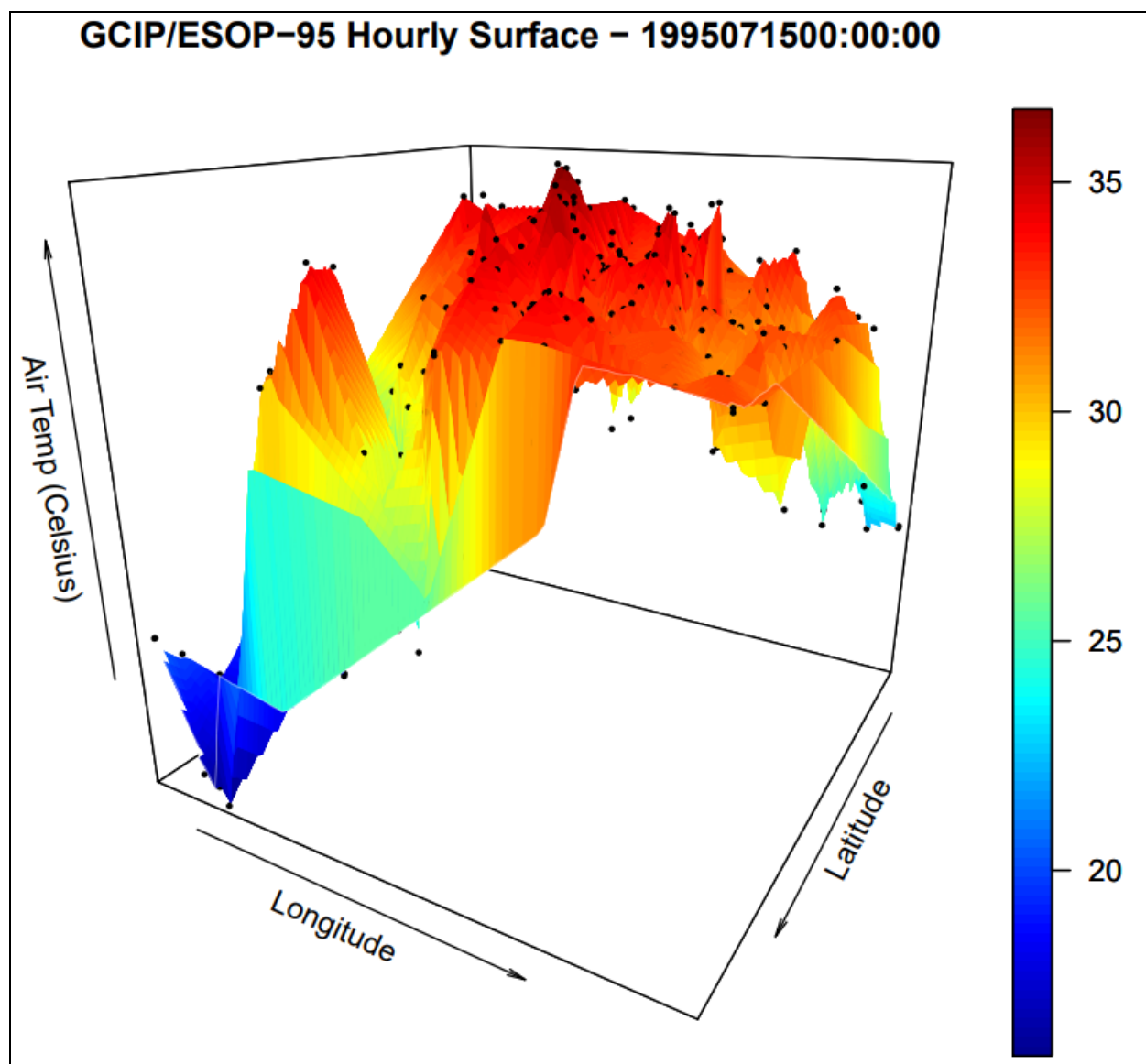
```

**Figure 15:** Lines 154-159 create the data frame for plotting. Line 163 removes any data lines that contain missing values like the NaN values set before. Lines 167-168 creates a matrix from the variables for plotting. Lines 172-173 interpolate a grid for plotting. Line 177 creates a mesh grid for the plot (*Note: removing the “border” removes the grid in the plot. The values for “phi” and “theta” change the viewing angle*). Lines 179-184 create the surface plot. Lines 186-187 creates the points shown on the 3D plot.



**Figure 16:** The final 3D plot of Air Temperature with the grid.





**Figure 17:** The 3D plot of Air Temperature without the grid.