Distributional Analysis

Distributional analysis is a term I coined for a very simple yet powerful way of analyzing datasets. It means that you think of the dataset as a distribution within a large multidimensional space, which you then can examine through its marginal statistics in any two-dimensional subspace.

The best way to understand this is through examples. So let's turn our attention to exploring a dataset of freely-drifting subsurface oceanographic floats. These instruments record latitude, longitude, and temperature as they drifter around with the currents at more-or-less fixed pressure levels. You'll need to download "floats.nc" from my web site. (This is included in the full distribution of the course materials.)

This notebook requires my jlab toolbox to be installed. You will also need to have set up Jupyter Lab to work with Matlab, following these instructions.

In [1]:
```
set(groot,'defaultfigurepaperposition',[0 0 10 5]) %set default figure size

datadir = '/Users/lilly/Desktop/Dropbox/Matlab/jdata/floats/';  %path to the directory
ncload([datadir 'floats.nc'])
floats
```

```
floats =
  struct with fields:

     expid: [3091x1 double]
    typeid: [3091x1 double]
        id: [3091x1 double]
       num: {3091x1 cell}
       lat: {3091x1 cell}
       lon: {3091x1 cell}
         u: {3091x1 cell}
         v: {3091x1 cell}
         p: {3091x1 cell}
         t: {3091x1 cell}
    filled: {3091x1 cell}
```

The first thing you notice is that most of the variables are things called "cells". A cell array is a very useful type of data structure in Matlab, so it's worthwhile to take a few minutes to introduce it to you.

Introduction to Cell Arrays

The floats.mat dataset consists of primarily of the positions of floats as they follow the currents, giving us time series of latitude and longitude. However, all the floats trajectories are of different lengths.

Let's say that we were to try to store this data as a matrix. We could put time in rows, with each column corresponding to a different float. By the way, this is a good time to mention that in my convention, time is always oriented to increase in the row direction. In other words, if I work with a single time series, I always use a column vector, and not a row vector. In Matlab x=[1:10]' is a column vector while x=[1:10] is a row vector.

If we were to store the float data as a matrix, then the number of rows would have to be the same as the number of data points of the longest trajectory. This would be very inefficient because we would then have a lot of empty space in the matrix. It works ok for small datasets, but for large data sets you can quickly run into memory trouble. So it doesn't really make sense to store this data as a matrix.

A cell array offers us an easy solution. A cell array is like a regular array, except the entries can be anything. Here is an example:

In [2]:
```
a{1}='apple';
a{2}='banana';
```

```
a{3}='pear';
a
```

```
a =
  1x3 cell array
    {'apple'}    {'banana'}    {'pear'}
```

Individual entries of cell arrays are accessed through curly braces ...

In [3]:
```
a{2}
```

```
ans =
    'banana'
```

...whereas parantheses let us access a subset of the cell array itself:

In [4]:
```
a(1:2)
```

```
ans =
  1x2 cell array
    {'apple'}    {'banana'}
```

Note that a(1:2) is another cell array, where as a{1} is a string in this case.

Cell arrays can also be of different sizes, and can even combine objects of different types:

In [5]:
```
b{1,1}='age';
b{1,2}=46;
b{2,1}='height';
b{2,2}=179;
b
```

```
b =
  2x2 cell array
    {'age'   }    {[ 46]}
    {'height'}    {[179]}
```

Now b{1,1} is a string while b{1,2} is an integer. As you can see cell arrays are pretty flexible. However, in what follows we are going to limit ourselves to working with one particular format, cell arrays of numerical arrays.

To keep track of arrays of different lengths, we can store them as a cell array.

In [6]:
```
x=[1:10]'; %the prime in Matlab is the transpose
y=[1:20]'; %(or conjugate transpose for complex valued-arrays)
z=[1:4]';  %it converts a row vector into a column
c{1,1}=x;
c{2,1}=y;
c{3,1}=z;
c
```

```
c =
  3x1 cell array
    {10x1 double}
    {20x1 double}
    { 4x1 double}
```

Now c is a cell array in the shape of a column vector, and each of its elements is itself a column vector. We can think of this as being like stacking x, y, and z on top of each other. (Note that if

we had just written c{1}=x rather than c{1,1}=x and so forth, c would have become a row vector than a column vector.)

For simplicity we will refer to this format as "cell array format." This is the format used by floats.mat. This turns out to be hugely useful in many applications. I use it so often I wrote a whole module for working with data in this format called jCell, which you can read about by typing "help jcell".

For now though, I just wanted to explain cell arrays and the basic structure of the floats.mat variables.

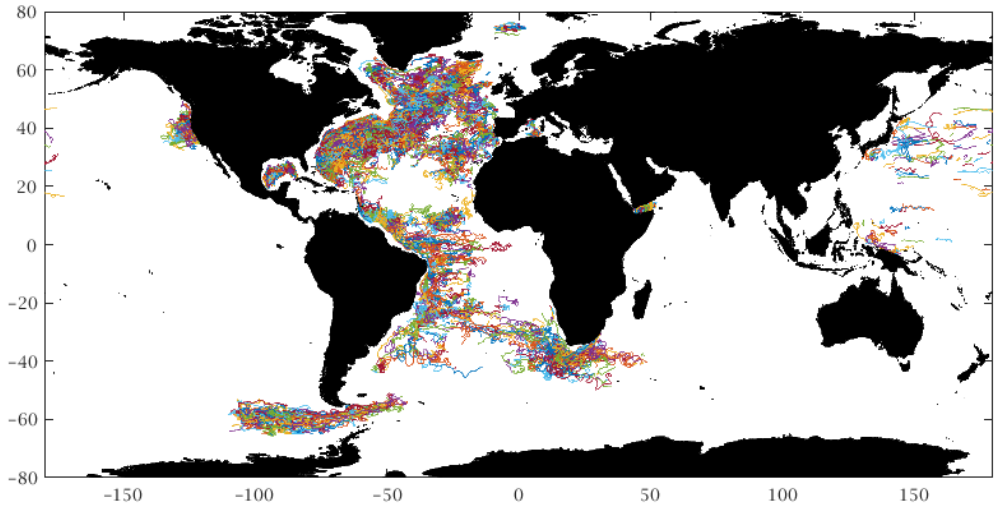
We see that we have a data set with many different variables and two dimensions: column and segment. The column dimension has all data points from all float trajectories concatenated together, with nans marking the tails, while the segment dimension has one element per float trajectory (e.g. float id number).

Exploring Floats.nc

Let's make a basic plot of floats.nc. We will plot each trajectory with a different color.

After that detour into cell arrays, we're now ready to plot the data. A handy jCell function called cellplot lets us plot data in cell array format without needing to loop. Go ahead and evaluate the following, which might take a while because the data is pretty large:

In [7]:
```
use floats
cellplot(180,lon,lat),axis tight
axis([-180 180 -80 80])
topoplot continents,latratio(25)
```



Note that double-clicking on this plot will make it larger, and will also show it with better resolution.

The first argument to cellplot, 180, cuts the longitude as it crosses from +180 back to -180 or vice-versa, preventing horizontal lines that would otherwise appera in this plot.

Here topoplot is a convenient function for plotting bathymetry and topography. We will look at this more later so you don't need to worry about it too much for now.

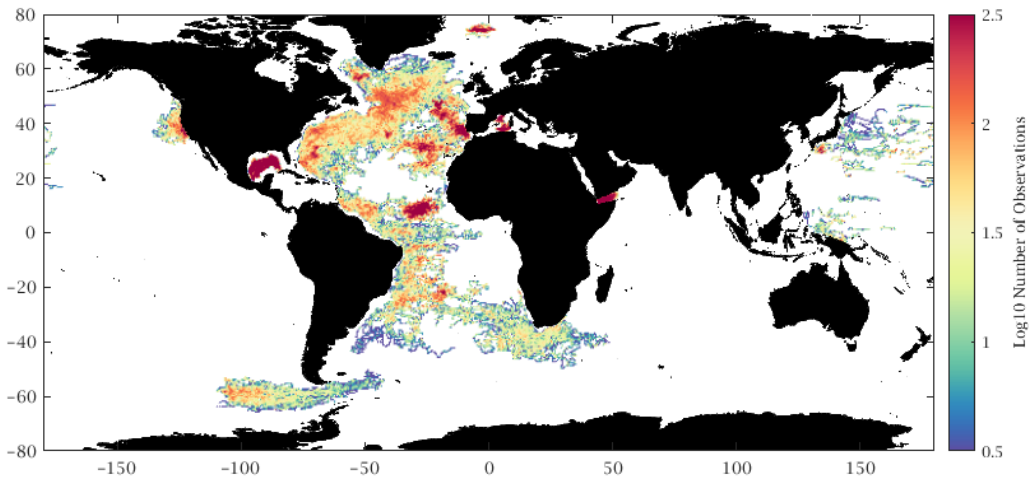
The other command, latratio, sets the aspect ratio of the plot correct for the input latitude. In other words, latratio(30) sets the aspect ratio to be 1 to 1 at 30 degrees latitude. This enables us to decide what aspect ratio we want a latitude/longitude plot to be. Otherwise, stretching a figure window in Matlab will stretch latitude and longitude accordingly, and we can often end up with some really distorted looking maps.

Two-Dimensional Histograms

After all of that we are finally ready to begin working with the distributional data analysis. Firstly we want to examine the data density, that is, the number of observations per grid point. To do this we will make a two-dimensional histogram in latitude--longitude space.

Two-dimensional histogram are made in jLab by the routine twodhist. This is essentially the same functionality as Matlab's histcounts2, but as twodhist predates histcounts2, I continue to use the former function.

In [8]:
```
[mat,xmid,ymid]=twodhist(lon,lat,[-180:1/2:180],[-80:1/2:80]);
jpcolor(xmid,ymid,log10(mat))
caxis([0.5 2.5]),hc=colorbar;
axis([-180 180 -80 80])
topoplot continents,latratio(25)
hc.Label.String='Log10 Number of Observations';
```



We see that the observation density is high throughout the Atlantic, and is particularly high in the Gulf of Mexico, the Gulf of Aden adjacent to the Red Sea, and several other hot spots.

The plotting command jpcolor is a version of Matlab's pcolor that does a better job of plotting the values in our matrix at their correct locations. Unlike pcolor, jpcolor does not cut off one row and one column of the plotted matrix. See the jpcolor help for details.

The call to twodhist requests a two-dimensional histogram of longitude and latitude locations. The last two arguments of twodhist specify the bin edges. In the output of twodhist, xmid and

ymid are the bin centers corresponding to the input edge arrays, while mat is the number of data points per bin.

Here we have plotted the logarithm of the data density in order to better visualise the vast geographic differences in this quantity.

Twodhist works when the input arrays are in cell array format, as they are here, or regular numerical arrays. Because we're just counting the number of data points in bins, the format of the input variables doesn't matter. They can be matrices, arrays, or arrays in cell array format, as long as they're all the same size as each other.

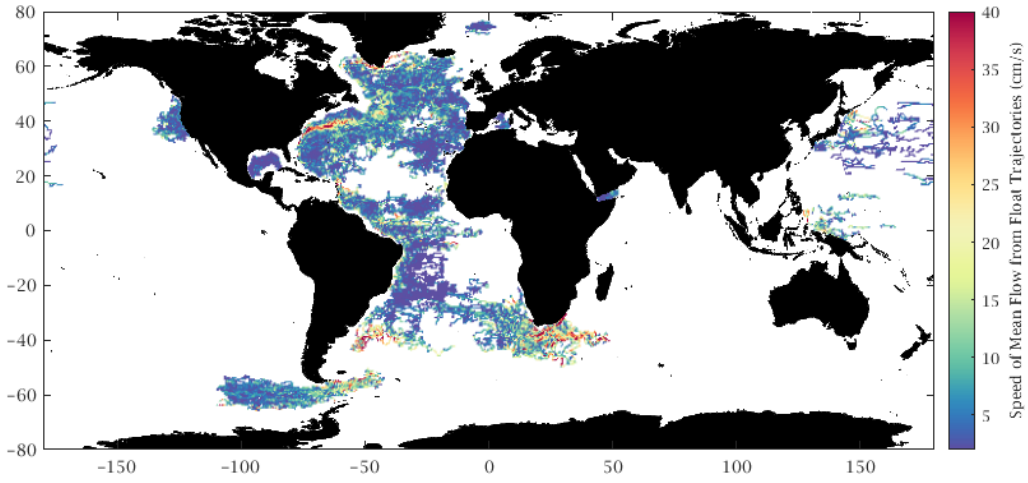
A final point is that twodhist works through a clever algorithm that uses no explicit loops, by directly looking up the index into the bin count matrix mat. This enables it to be fast (by Matlab standards) even for very large datasets.

Two-Dimensional Means and Standard Deviations

Another important statistic we can plot is the speed of the mean flow. The two-dimensional mean is computed in jLab with the function twodstats.

In [9]:

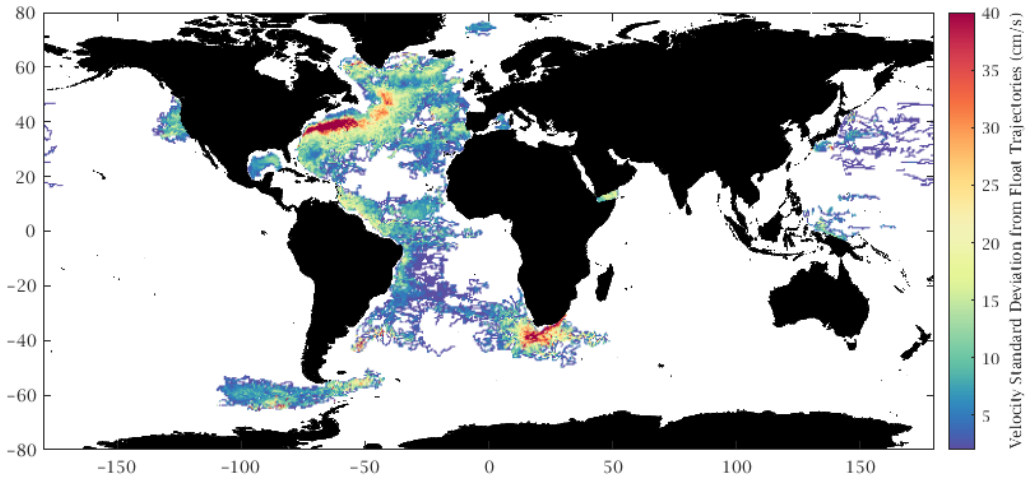
```
[meanu,xmid,ymid]=twodstats(lon,lat,u,[-180:1/2:180],[-80:1/2:80]);
[meanv,xmid,ymid]=twodstats(lon,lat,v,[-180:1/2:180],[-80:1/2:80]);

jpcolor(xmid,ymid,sqrt(meanu.^2+meanv.^2))
caxis([2 40]),hc=colorbar;
axis([-180 180 -80 80])
topoplot continents,latratio(25)
hc.Label.String='Speed of Mean Flow from Float Trajectories (cm/s)';
```



In the North Atlantic, we see that the mean flow is large over the Gulf Stream, as expected, as well as around the periphery of Greenland where there is a known boundary current. The high velocities in the South Atlantic and North Pacific correspond to areas where the preceding map showed a low sample density, and therefore, the mean flows in these areas are probably not well resolved.

Another interesting statistics is the velocity standard deviation. This is also implemented by twodstats, where it is output as the fifth output argument.

In [10]:
```
[meanu,xmid,ymid,~,stdu]=twodstats(lon,lat,u,[-180:1/2:180],[-80:1/2:80]);
[meanv,xmid,ymid,~,stdv]=twodstats(lon,lat,v,[-180:1/2:180],[-80:1/2:80]);

jpcolor(xmid,ymid,sqrt(stdu.^2+stdv.^2))
caxis([2 40]),hc=colorbar;
axis([-180 180 -80 80])
topoplot continents,latratio(25)
hc.Label.String='Velocity Standard Deviation from Float Trajectories (cm/s)';
```



Here, we're using the same color axis as in the previous plot to make these two quantities directly comparable. Note that the area of high velocity variance in the vicinity of the Gulf Stream appears broader than the area where the mean flow is large in magnitude. This is an intuitive result because the Gulf Stream is characterized by meanders about its mean position. Also in the plot, a high variance region south of Africa, associated with the Agulhas current, appears more clearly than was seen in the mean flow map.

Two-Dimensional Means of Further Quantities

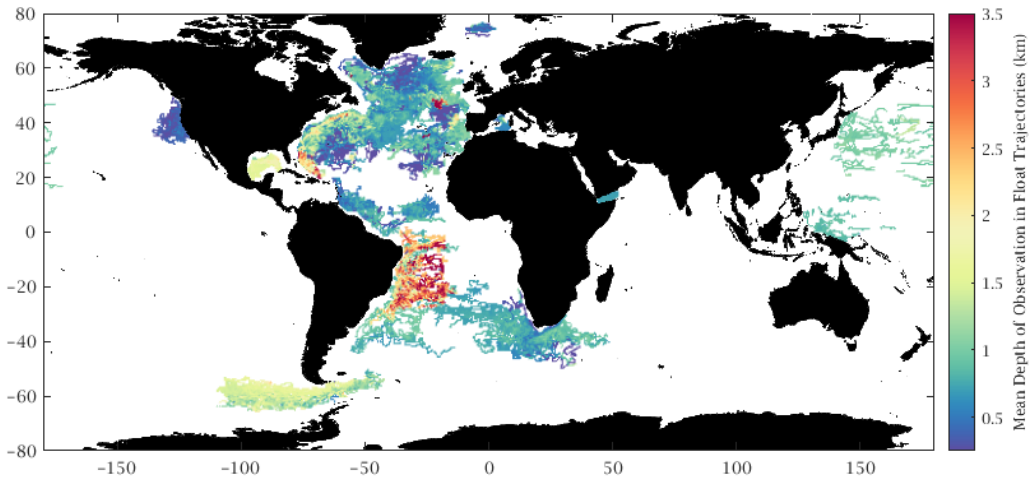
There are less obvious aspects of the data that we can also examine through the use of plots of its statistics in two dimensions. The floats are not all at the same vertical level. Rather, they are set to observe a target pressure level, which can be approximeaned as a depth level; this pressure level tends to wander a bit with time as temperature and salinity properties of the water column change.

Thus the above maps of velocity have mixed up measurements from many different vertical levels. To get an idea of the spatial distribution of the float sampling with respect to pressure, we can look at the mean *pressure* rather than the mean *velocity*.

Note that in the following we make use of the approximation that 1 decibar of pressure is about 1 m of depth in the ocean.

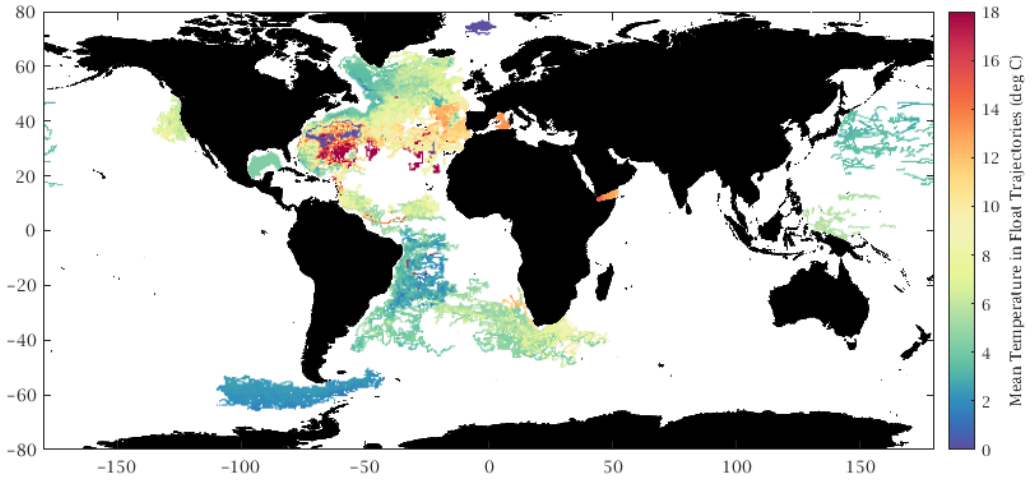
In [11]:
```
[meanp,xmid,ymid]=twodstats(lon,lat,celldiv(1000,p),[-180:1/2:180],[-80:1/2:80]);
```

```
%note that celldiv is used to divide a cell array, like p, by a scalar, converting dba

jpcolor(xmid,ymid,meanp)
caxis([.25 3.5]),hc=colorbar;
axis([-180 180 -80 80])
topoplot continents,latratio(25)
hc.Label.String='Mean Depth of Observation in Float Trajectories (km)';
```



Similarly, we can look at the mean temperature observed by the floats.

In [12]:
```
[meant,xmid,ymid]=twodstats(lon,lat,t,[-180:1/2:180],[-80:1/2:80]);

jpcolor(xmid,ymid,meant)
caxis([0 18]),hc=colorbar;
axis([-180 180 -80 80])
topoplot continents,latratio(25)
hc.Label.String='Mean Temperature in Float Trajectories (deg C)';
```

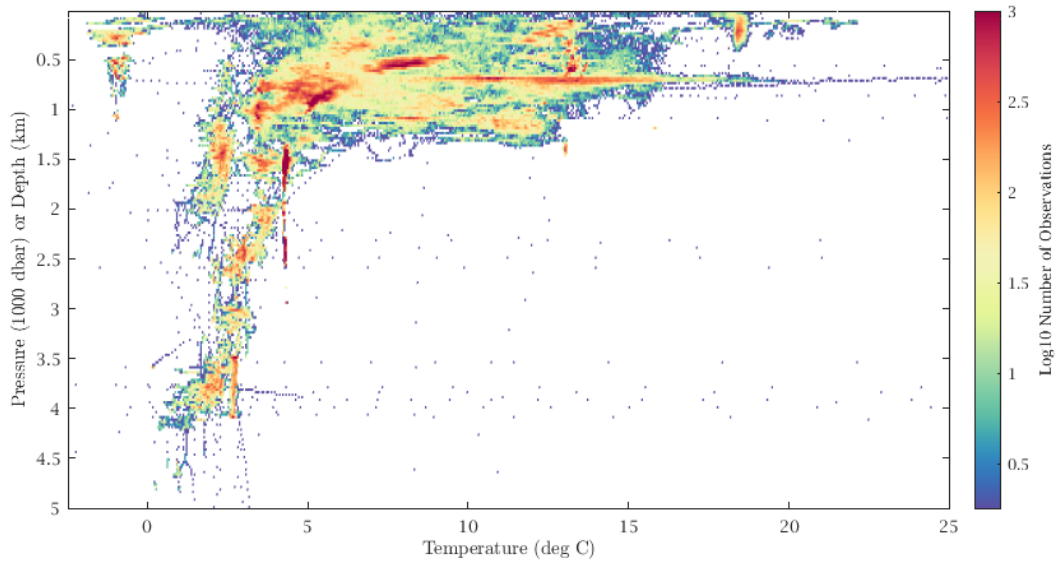


Rotating the Distribution

How is the observed temperature distribution related to the observational depths (or pressures) seen in the previous plot? We can explore this relationship, still using two-dimensional statistics,

by changing our perspective and examining the distribution of the data in *pressure and temperature space*.

In [13]:
```
[mat,xmid,ymid]=twodhist(t,celldiv(1000,p),[-2.5:.05:25],[0:.025:5]);

jpcolor(xmid,ymid,log10(mat))
caxis([0.25 3]),hc=colorbar;
xlabel('Temperature (deg C)')
ylabel('Pressure (1000 dbar) or Depth (km)')
hc.Label.String='Log10 Number of Observations';
flipy
```



This very beautiful plot shows how the float measurements are sampling the basic temperature structure of the ocean. A number of interesting features are apparent.

What is important to note is that it is essentially just the first two-dimensional statistic that we plotted earlier---the number of observations in bins. However, rather than plotting the number of samples in lat--lon space, here we are using temperature--pressure space.

When you have an object in three dimensions, its shadow changes as you rotate it. The shadow is the projection of the higher-dimensional object into two-dimensional space.

Our dataset is a distribution of values in multiple dimensions---latitude, longitude, pressure, temperature, u velocity, v velocity, speed, data, time of year, etc.---and so can be thought of as a multidimensional object. When we plot its distribution in different two-dimension spaces, it is like rotating this multidimensional object to look at its shadows.

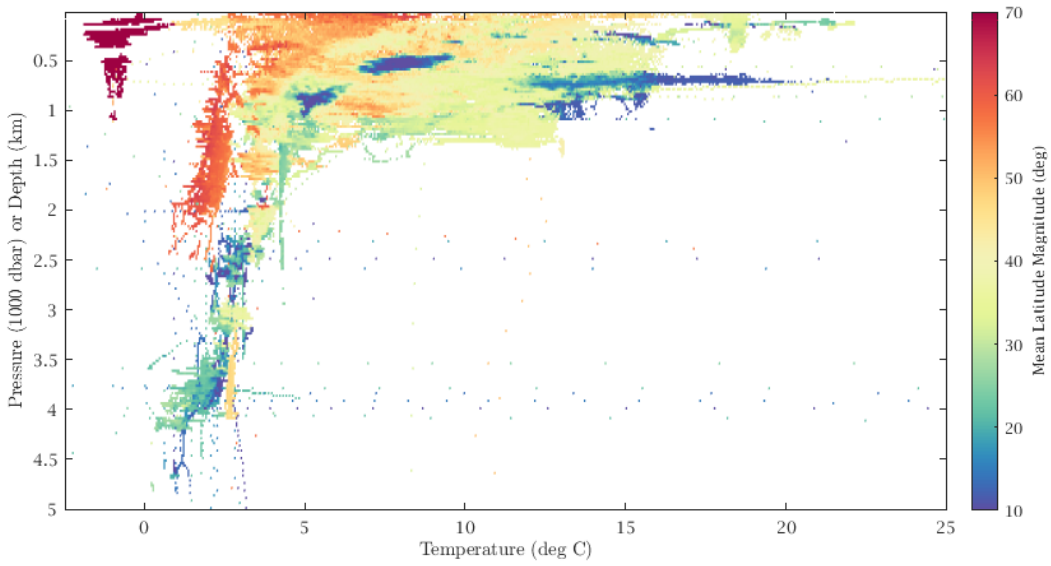
The data itself is a distribution in a high dimensional space that we can't visualize directly, although we can picture it abstractly. In statistical terms, collapsed statistics---such as the latitude and longitude distribution for all values of the other parameters---are known as *marginal statistics*.

This very simple idea is what is meant by distributional analysis. We are rotating the data and looking at its statistics in different two-dimensional subspaces.

We'll show a few more final examples. The temperature--pressure distribution certainly varies with latitude. Let's look at this in two different ways, firstly through examining the mean latitude

magnitude, abs(lat), as a function of the temperature--pressure plane.

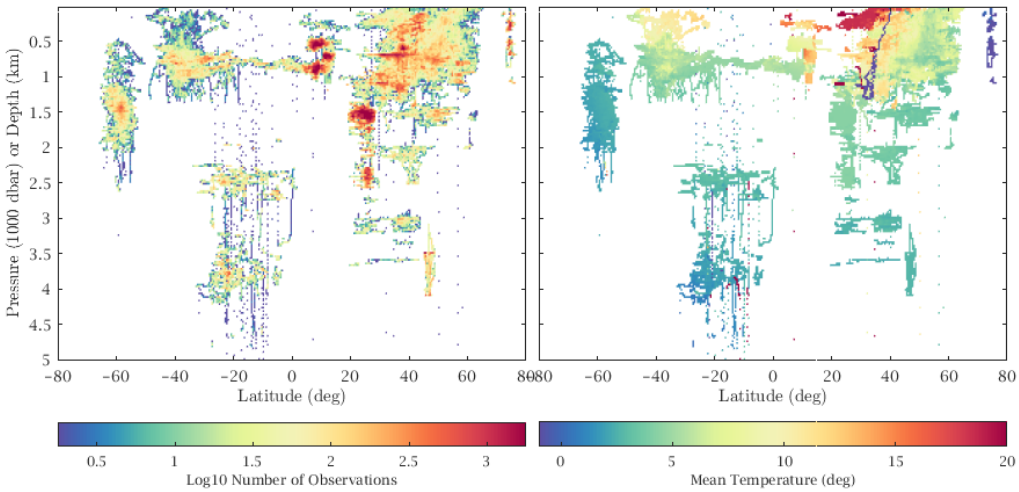
In [14]:
```
[meanlat,xmid,ymid]=twodstats(t,celldiv(1000,p),cellabs(lat),[-2.5:.05:25],[0:.025:5])
%cellabs applies the abs function to a cell array

jpcolor(xmid,ymid,meanlat)
caxis([10 70]),hc=colorbar;
xlabel('Temperature (deg C)')
ylabel('Pressure (1000 dbar) or Depth (km)')
hc.Label.String='Mean Latitude Magnitude (deg)';
flipy
```



Here we see, unsurprisingly, that temperatures below zero tend to occur at high latitudes, while warmer temperatures tend to occur at lower latitudes. A number of low-latitude blobs stand out that correspond to regions of high sampling density from the previous plot.

There are three dimensions being visualized in this plot---temperature, pressure, and latitude. Let's rotate that three dimensional distribution, and look at it from another angle.

In [18]:
```
[meant,xmid,ymid,mat]=twodstats(lat,celldiv(1000,p),t,[-80:1/2:80],[0:.025:5]);
%the fourth argument output from twodstats is the observation histogram

subplot(1,2,1)
jpcolor(xmid,ymid,log10(mat))
caxis([.25 3.25]),hc=colorbar('SouthOutside');
set(gcf,'paperposition',[0 0 11 5 ])
xlabel('Latitude (deg)')
ylabel('Pressure (1000 dbar) or Depth (km)')
hc.Label.String='Log10 Number of Observations';
xlim([-80 80]),flipy

subplot(1,2,2)
jpcolor(xmid,ymid,meant)
caxis([-1 20]),hc=colorbar('SouthOutside');
xlabel('Latitude (deg)')
ylabel('Pressure (1000 dbar) or Depth (km)')
hc.Label.String='Mean Temperature (deg)';
xlim([-80 80]),flipy

packfig(1,2)
```



The left-hand plots shows the distribution of observations on the latitude--pressure plane, while the right-hand plot shows the average temperature on this plane. If you compare with the previous plots on the temperature--pressure plane, you can match up a number of clouds of observations. For example, look between 0 and 20 degrees latitude in the above plot and see what this corresponds to on the temperature--pressure plane.

Using distributional analysis, we can rapidly explore datasets now matter how they are organized. With a little imagination, patterns hidden in the data can begin to emerge, helping us determine what our next steps should be. I find this approach to be remarkably powerful, and it is normally my first step in diving into a new dataset.

A few ideas that we didn't explore here, but that are often useful, are (i) using date, time of year, or time of day, as one of our dimensions, (ii) using other statistics, such as the median or high-order statistics like skewness and kurtosis, and (iii) using frequency, arising from the spectral analysis methods we'll explore later, as one of the dimensions.

The End