Variance Ellipses

In this lab, you'll be working with current meter data, that is, observations of the horizontal ocean currents made from a fixed point. The main analysis tool we will learn are variance ellipses, a fundamental tool for analyzing velocity datasets or any kind of bivariate (or complex-valued) data. You'll see their relationship to the two-dimensional histogram, and will learn some associated analysis and visualization ideas, in particular the value of finding a suitable coordinate rotaiton.

For this assignment, we are going to use a mooring from the Labrador Current on the west side of the Labrador Sea known as the 'm1244' mooring. Please download my version of it here. (This is included in the full distribution of the course materials.)

Many thanks to Jan-Adrian Kallmyr for helping with translating the Matlab tutorial into Python.

In [1]:

```
import warnings
warnings.filterwarnings('ignore')  #suppress some warnings about future code changes

import netCDF4 as nc
import xarray as xr
import numpy as np
import datetime          #https://docs.python.org/3/library/datetime.html
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.signal as sg  #Package for signal analysis
import scipy.ndimage as si #Another package for signal analysis
from scipy.interpolate import interp1d  #for converting cell to grid-centered coordi
from scipy import stats    #Used for 2D binned statistics
from mpl_toolkits.axes_grid1 import make_axes_locatable #For plotting interior colob
import cartopy.crs as ccrs

#function for coverting Matlab's datenum into Python's datetime
#from https://gist.github.com/victorkristof/b9d794fe1ed12e708b9d
#with modifications to support array input and output
def datenum_to_datetime(datenum):
    """
    Convert Matlab datenum into Python datetime.

    Args:
        datenum: Date in datenum format

    Returns:
        Datetime object corresponding to datenum.
    """
    date=[]
    for day in datenum:
        date.append(datetime.datetime.fromordinal(int(day)) \
           + datetime.timedelta(days=day%1) \
           - datetime.timedelta(days=366))
    date=np.array(date)
    return date

plt.rcParams["figure.figsize"] = (10,8) #set default figure size
```

A Quick Look at the Data

Now let's load and examine the data.

In [2]:
```
datadir = "../data/"  #choose this appropriate for your system
filename = "ml244.nc"
ds = xr.open_dataset(datadir+filename) #Load the dataset
ds = xr.Dataset.transpose(ds)  #Enforce the convection that time is in rows
print(ds)
```

```
<xarray.Dataset>
Dimensions:  (time: 7371, depth: 4)
Dimensions without coordinates: time, depth
Data variables:
    lat      float64 ...
    lon      float64 ...
    num      (time) float64 ...
    depths   (depth) float64 ...
    u        (time, depth) float64 ...
    v        (time, depth) float64 ...
    t        (time, depth) float64 ...
    p        (time, depth) float64 ...
```

The variables are as follows:

depths --- Depths of 4 different currents meters, in meters
num -- date in Matlab's "datenum" format
p --- Pressure in decibar
t --- Temperature in Centigrade
cv --- Complex velocity u+iv in cm/s, u = eastward, v = northward

Before we proceed let's find out the sampling interval and duration of the time series.

In [3]:
```
num = ds["num"] #time in days starting from 0000-00-00
delta = (ds["num"][1] - ds["num"][0])*24 #sampling interval in converted to hours
T = (ds["num"][-1] - ds["num"][0])/365 #duration in days converted to years
print(delta.data)
print(T.data)
```

```
2.0000000009313226
1.6826484018263776
```

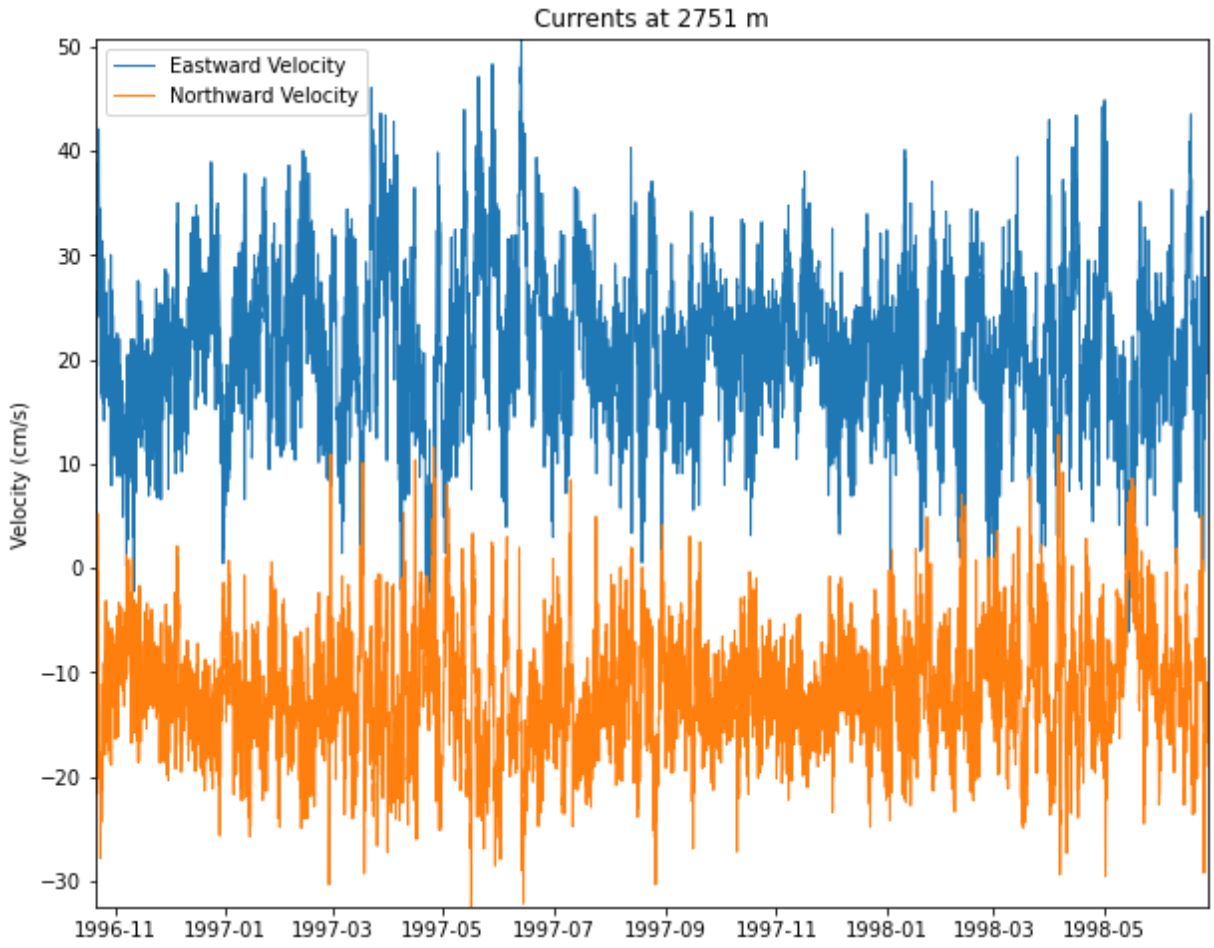
So we have a roughly 1.7 year record of the currents sampled every two hours.

Let's take a look at the currents the deepest depth.

In what follows, we're going to take advantage of Python's very useful ability to define functions on the fly. This will let us make efficiently re-use code in later plots.

In [4]:
```
def plot_complex_velocity(date,cv):
    """
    Line plot of the real and imaginary parts of a complex velocity.

    Args:
        date: Date as a datetime object
        cv: Complex velocity u+iv in cm/s, with time in *rows*

    Returns:
        Nothing
    """
    ax=plt.gca()
```

```
    ax.plot(date,cv.real, linewidth=1)
    ax.plot(date,cv.imag, linewidth=1)
    ax.autoscale(enable=True, tight=True)
    plt.ylabel('Velocity (cm/s)')

date = datenum_to_datetime(ds["num"].data)       #date as a datetime object
cv   = ds['u']+1j*ds['v']                        #complex-valued velocity

fig, ax = plt.subplots(1, 1)
plot_complex_velocity(date,cv.data[:,3])
ax.legend(['Eastward Velocity', 'Northward Velocity'])
plt.title('Currents at ' + str(int(ds["depths"].data[3])) + ' m');
```



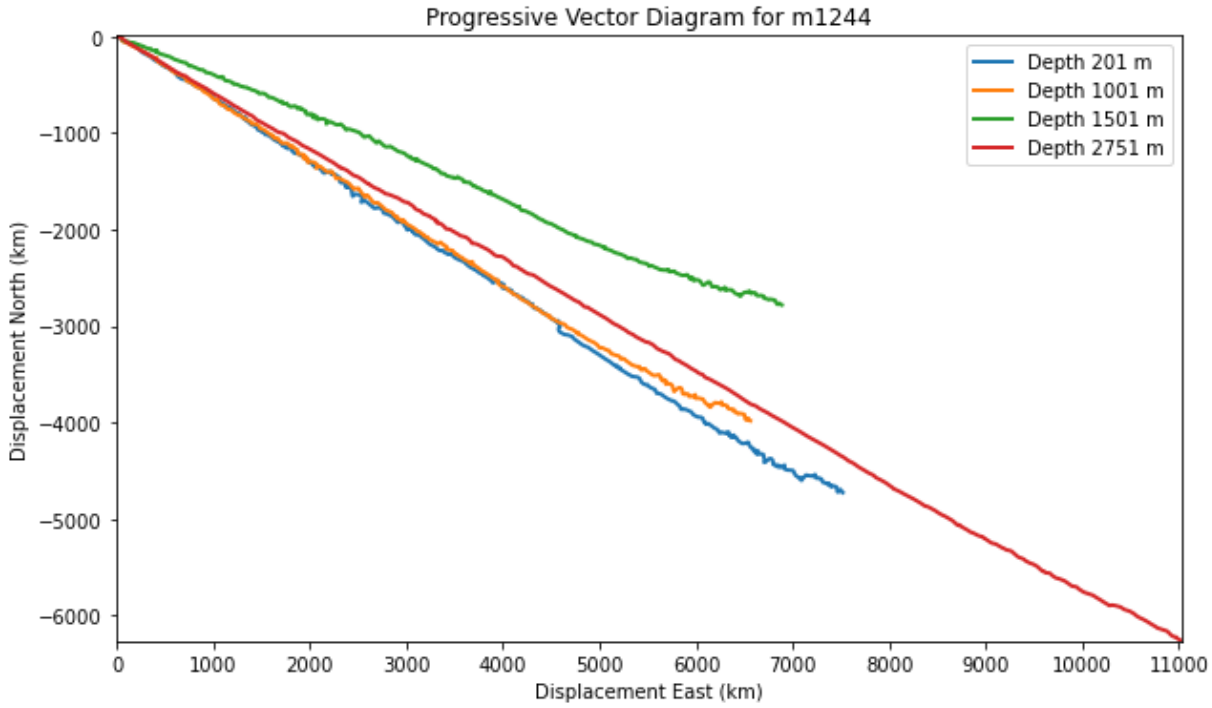
Here a mean value is readily apparent in both the eastward and northward components. In addition, we see a lot of small-scale variability, possibly with a greater amplitude in the eastward component. Hints of multiple timescales of variability are present, with a fine noise-like variability superposed on somewhat longer timescales.

It is a little difficult to make sense of this plot, however, because it is not rotated into the most useful coordinate system. Let's figure out a sensible rotation angle.

Choosing a Coordinate Rotation

To do this, we will look at the progressive vector diagram. For a current measurement at a point, the progressive vector diagram is defined as a plot of its time integral. In other words, the progressive vector diagram shows the displacement that would occur if a particle were advected with the given currents.

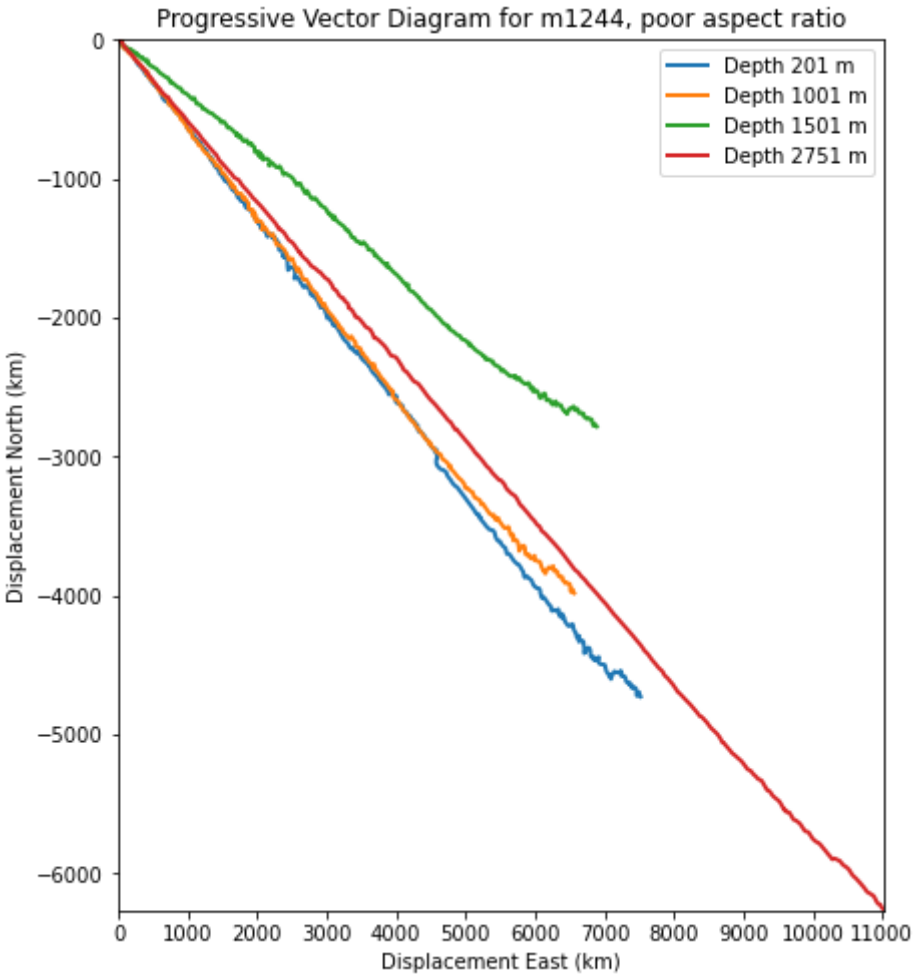
In [5]:
```
def provec(dt,cv,ticks = np.arange(-50, 50, step=1)*1000):
    """
    Compute and plot the progressive vector diagram.

    Args:
        dt: Sampling interval in seconds
        cv: Complex velocity u+iv in cm/s, with time in *rows*
        ticks: Tickmark locations in km (optional); defaults to every 1000 km

    Returns:
        Nothing
    """
    cx=np.cumsum(cv, axis=0)*dt/100/1000  #complex displacement x+iy in kilometers

    ax=plt.gca()
    ax.plot(cx.real,cx.imag,linewidth=2)
    ax.set_aspect('equal')  #this sets the data aspect ratio to 1:1   <--------------

    plt.xlabel('Displacement East (km)')
    plt.ylabel('Displacement North (km)')
    plt.xticks(ticks), plt.yticks(ticks)    #set x-tick and y-tick locations
    ax.autoscale(enable=True, tight=True)   # tighten axes limits around the data

dt=(date[1]-date[0]).seconds;     #sampling interval in seconds
fig, ax = plt.subplots(1, 1)
provec(dt,cv.data)
strs=[]
for depth in ds["depths"].data:
    strs.append('Depth ' + str(int(depth)) + ' m')
ax.legend(strs)
plt.title('Progressive Vector Diagram for m1244');
```



We see a strong mean flow at all depths directed to the east-southeast. This matches what we saw above with the line plot, where the positive mean of the eastward currents is roughly twice as large as the negative mean of the southward currents.

Note there is an important line in the "provec" function that sets the aspect ratio. If we had omitted it,we could have gotten something tlat looks like this:

In [6]:
```
#demonstration of what could happen if you fail to set the correct aspect ratio
fig, ax = plt.subplots(1, 1)
provec(dt,cv.data)
ax.set_aspect(2) #sets the height to width ratio to 2 (incorrectly)
ax.legend(strs)
plt.title('Progressive Vector Diagram for m1244, poor aspect ratio');
```



Here, the currents appear to be directed due southeast. However, there is a problem with this plot: the x and y axes, which are the same physical quantity of displacement, correspond to different physical lengths! If we re-scaled the figure window, the direction of the mean flow would appear to change. Thus our perspective on the comparison between the eastward and northward currents is distorted.

This problem is fixed by setting the data aspect ratio. Not setting the data aspect ratio correctly is actually a common problem, even, regrettably, in many published papers.

Rotating by the Mean Flow Direction

It's natural to rotate our current meter data so that the east-southeastward direction of the mean flow corresponds to the first velocity component, and the direction normal to that to the second velocity component. In other words, we will rotate our (u,v) data to become $)\tilde u,\tilde v)$ with $\tilde u$ corresponding to the 'downstream' direction and $\tilde v$ corresponds to the 'cross-stream' direction. We will take a little time to do this so we understand how rotations work.

Firstly we find the direction of the mean flow at the deepest current meter, where the flow is strongest.

In [7]:
```
np.angle(np.mean(cv[:,3]),deg=True) #angle in degrees
```

Out[7]:
```
-29.602806827016337
```

So the angle is about 30 degrees clockwise from due east. That looks about right!

Rotation of a complex-valued number $z=u+iv$ are straightforward. To rotate $z=|z|e^{i\varphi}$ through some angle ϕ, we simply multiply by $e^{i\phi}$. This leads to a new version of z , denoted $\tilde z =\tilde u+ i\tilde v=|z|e^{i(\varphi+\phi)}$.

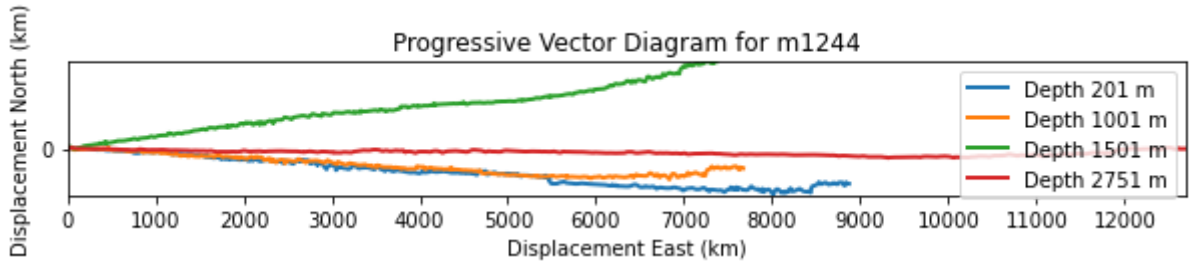
In this case, we want to choose the rotation angle ϕ as the *negative* of the angle of the mean flow. Let's look at the mean real and imaginary components of the velocity before and after this rotation.

In [8]:
```
print(np.mean(cv[:,3]).data)
phi=-np.angle(np.mean(cv[:,3]));
print(np.mean(cv[:,3]).data*np.exp(1j*phi))
```

```
(20.785806533575666-11.809328448895153j)
(23.90627515245989+0j)
```

This shows that after the rotation, the mean of the cross-stream velocity (the imaginary part) is zero, as expected. Now let's re-plot the progressive vector diagram.

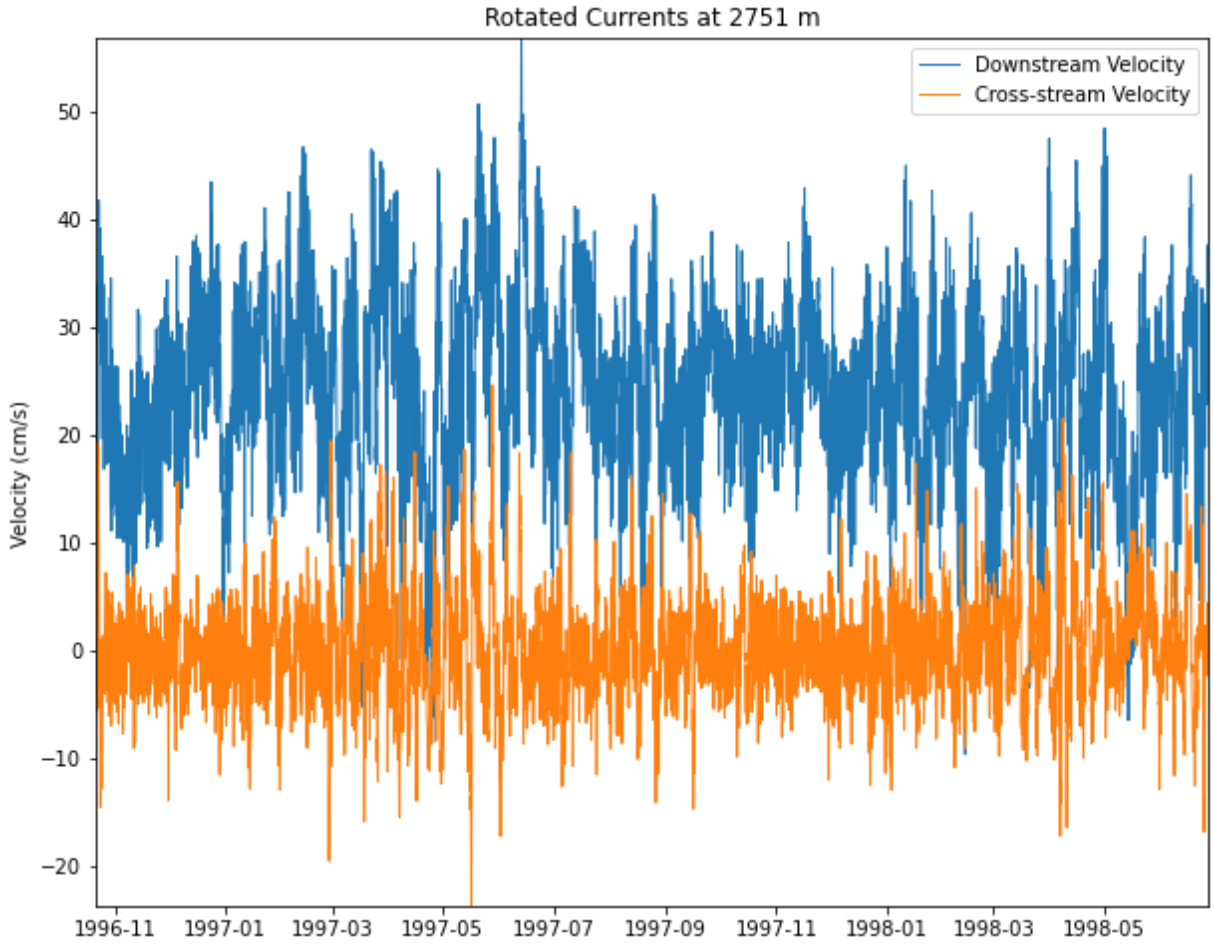
In [9]:
```
cvr=cv.data*np.exp(1j*phi)  #rotated version of the velocity
fig, ax = plt.subplots(1, 1)
provec(dt,cvr)
ax.legend(strs)
plt.title('Progressive Vector Diagram for m1244');
```



At first glance, this plot appears rather boring. But actually, boring can be a good sign because it means we've found a way to look at our dataset in such a way that it simplifies!

Let's return to the line plot we made earlier, but now make it for the rotated velocity data.

In [10]:
```
fig, ax = plt.subplots(1, 1)
plot_complex_velocity(date,cvr[:,3])  #line plot with rotated version of the velocity
ax.legend(['Downstream Velocity','Cross-stream Velocity'])
plt.title('Rotated Currents at ' + str(int(ds["depths"].data[3])) + ' m');
```

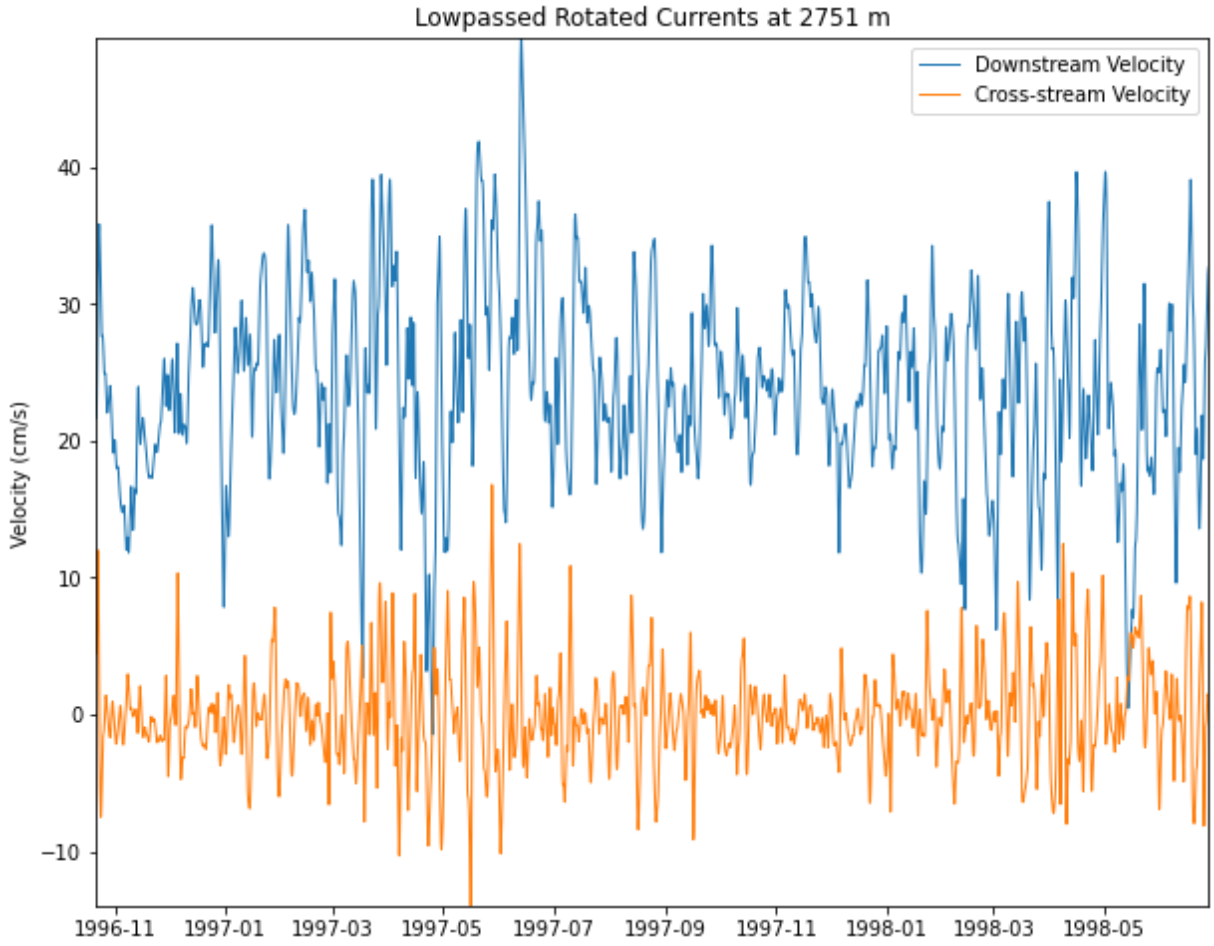


The downstream and cross-stream components are seen to be distinctly different. The cross-stream flow oscillates about a mean of zero, which is by construction, and also presents higher-frequency variability than does the downstream flow. Or, perhaps more accuately, it appears that the cross-stream flow is *lacking* an intermediate-timescale component that is present in the downstream flow.

Let's take a closer look at the timescales present in this dataset with a simple lowpass filter. We'll use a 24-point filter, which corresponds to a 2-day running mean since the sampling interval is 2 hours.

In [11]:

```
window = sg.windows.hann(24)       #24-point hanning window
window = window / np.sum(window)   #normalized to sum to unity
fcvr =  si.convolve(cvr[:,3], window, mode="mirror")  #filtered version of velocity a

#note I'm using convolution from the image rather than the signal package because the
#allows nicer handling of the boundary condition.  When working with a 2D array of tim
#you'll want to use the signal version

fig, ax = plt.subplots(1, 1)
plot_complex_velocity(date,fcvr)
ax.legend(['Downstream Velocity','Cross-stream Velocity'])
plt.title('Lowpassed Rotated Currents at ' + str(int(ds["depths"].data[3])) + ' m');
```



Here we can see clearly that the downstream flow presents an intermediate-timescale variability that is not present in the cross-stream flow.

Our rotation, designed just based on the *mean*, has thus also revealed distinctions in *variability*. The variability of the flow is now seen as being *anisotropic*, that is, lacking the property of being the same in all directions. This was not visible before the rotation because the downsteam and cross-stream components were mixed.

In general, finding ways to "rotate" a dataset, perhaps in an abstract way, in such a way that variability presents anisotropic structure is a quite simple yet powerful approach we can use to unlock its information content.

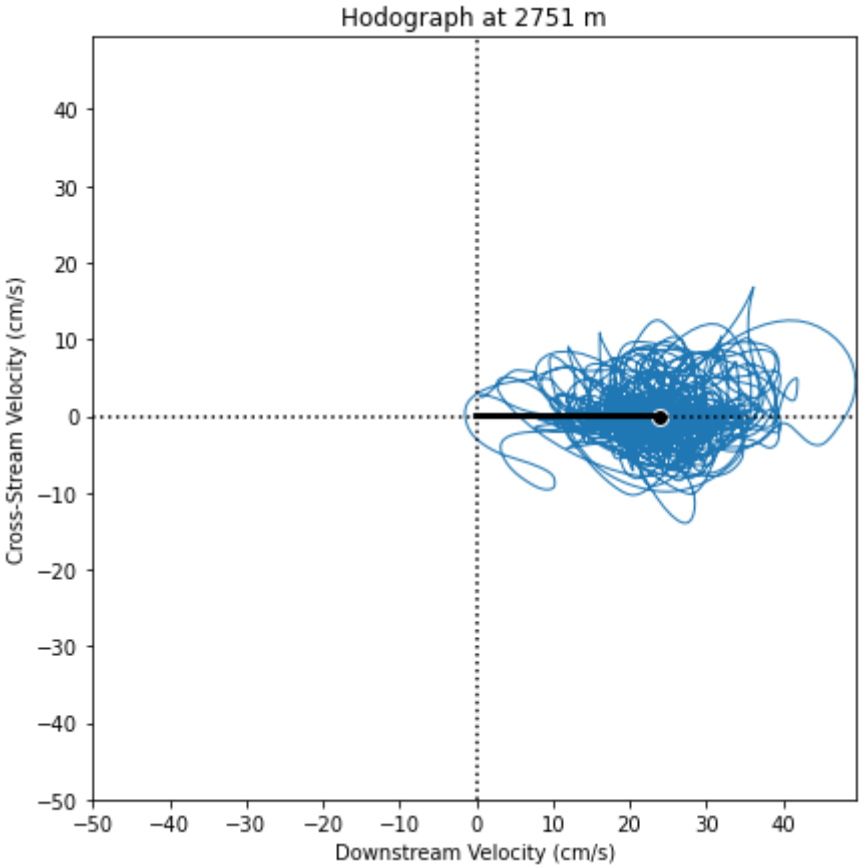
Here, the different timescales between the two components is not what one would expect if the variability were entirely due to the advection of eddies past the mooring; the timescales in that case would be the same. So this plot informs the physical hypotheses we would frame regarding the nature of the variability.

Two-Dimensional Histograms

Next we will summarize the statistics of the velocity through looking at its properties on the u,v plane. First, we'll make a simple line plot of $\tilde u$ versus $\tilde v$---a type of plot known as a *hodograph*---for the rotated versions of the currents we created earlier. We'll work with the lowpassed version of the time series for the moment.

In [12]:
```
def hodograph(cv,ticks = []):
```

```
    """
    Plot the velocity hodograph.

    Args:
        cv: Complex velocity u+iv, with time in *rows*
        ticks: [optional] Tickmark locations

    Returns:
        Nothing
    """


    ax=plt.gca()
    ax.plot(cv.real, cv.imag, linewidth=1) #plot u vs v
    ax.set_aspect('equal')  #this sets the data aspect ratio to 1:1
    ax.autoscale(enable=True, tight=True)

    #Next, plot the mean as a heavy line ending in a circle
    ax.plot(np.mean(cv.data).real, np.mean(cv.data).imag, "wo", markerfacecolor="k",
    ax.plot([0,np.mean(cv.data).real],[0,np.mean(cv.data).imag],color="k",linewidth=3

    ax.axhline(0, linestyle=":", color="black")
    ax.axvline(0, linestyle=":", color="black")
    plt.xlabel('Eastward Velocity')
    plt.ylabel('Northward Velocity ')
    plt.title('Velocity Hodograph')

    #set x and y to have the same axes
    plt.xlim([min(ax.get_xlim()[0],ax.get_ylim()[0]), max(ax.get_xlim()[1],ax.get_yli
    plt.ylim([min(ax.get_xlim()[0],ax.get_ylim()[0]), max(ax.get_xlim()[1],ax.get_yli

    if np.size(ticks)>0:
        plt.xticks(ticks),plt.yticks(ticks)

    return

fig, ax = plt.subplots(1, 1, figsize=(7, 7))
hodograph(fcvr,ticks=np.arange(-50, 50, step=10))
plt.xlabel('Downstream Velocity (cm/s)')
plt.ylabel('Cross-Stream Velocity (cm/s)')
plt.title('Hodograph at ' + str(int(ds["depths"].data[3])) + ' m');
```



The mean flow is plotted for reference. Note that the time integral of this plot is the progressive vector diagram.

At first glance, this plot is not very informative. It basically tells us that there is varability about the mean, with greater variability along the downstream axis then the cross-stream axis, which we already knew. But we can use statsitics to nicely summarize what we are seeing.

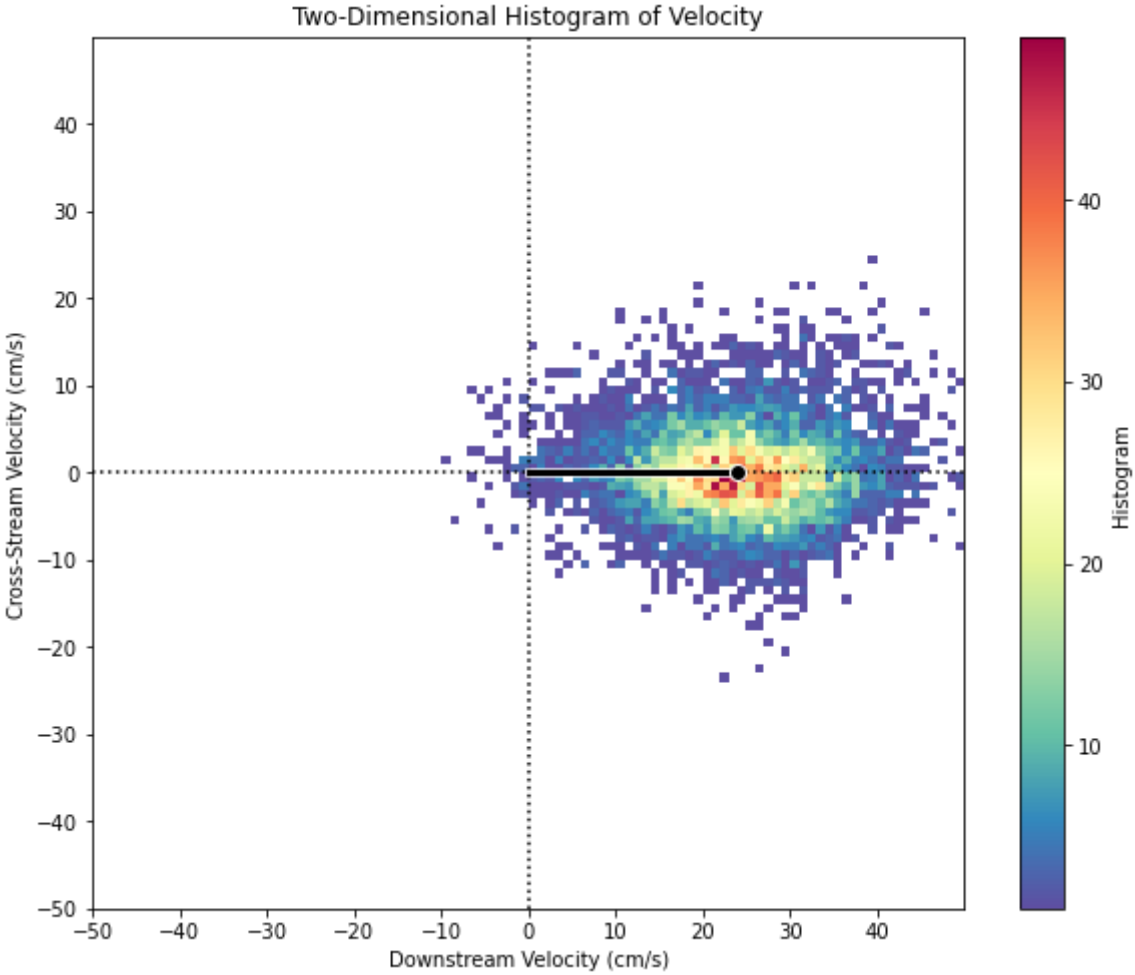
To do this we will look at two-dimensional distributions of the velocity. Firstly, we plot the two-dimensional histogram at the deepest depth.

In [13]:

```
def plot_twodstat(xbins,ybins,x,y,z=False,statistic="count",tickstep=False,axlines=
    """
    Compute and plot two a dimensional statistic.

    Args:
        xbins: Array of bin edges for x-bins
        ybins: Array of bin edges for y-bins
        x: Array of x-values to be binned
        y: Array of y-values to be binned; same size as x

    Optional Args:
        z: Array of z-values for which statistic is be formed; same size as x
        statistic: "count", "log10count", "mean", "median", or "std";
            defaults to "count", in which case the z argument is not needed
        tickstep: X- and y-axis tick step, a length 2 tuple; defaults to auto
        axlines: Axis origin locations for horizontal and vertical lines,
            a length 2 tuple, defaults to (0,0), lines omitted if False
        cmap: Colormap, defaults to Spectral_r
        colorbar: Plots a colorbar, defaults to True
        meandot: Plots a dot at the mean value, defaults to true
        meanline: Plots a line from the origin to the mean value, defaults to false
        axisequal: Sets plot aspect ratio to equal, defaults to false
```

```
    Returns:
        im: Image handle

    The computation of the statistic is handled by stats.binned_statistic_2d.

    Note for the computation of the standard deviation, we are using the form
    <(z - <z>)^2> = <z^2> - <z>^2, which is much faster than the algorithm used
    by stats.binned_statistic_2.

    Note also that z may be complex valued, in which case we define the standard
    deviation the square root of <(z - <z>)(z - <z>)^*> = <|z|^2> - |<z>|^2,
    which will be real-valued and non-negative.
    """


    #plot just one twodhist
    if statistic=="count":
        q = stats.binned_statistic_2d(x, y, None, bins=[xbins, ybins], statistic="cou
        q[q==0]=np.nan  #swap zero values for NaNs, so they don't appear with a color
        clabel='Histogram'
    elif statistic=="log10count":
        q = stats.binned_statistic_2d(x, y, None, bins=[xbins, ybins], statistic="cou
        q[q==0]=np.nan  #swap zero values for NaNs, so they don't appear with a color
        q=np.log10(q)
        clabel='Log10 Histogram'
    elif statistic=="mean":
        q = stats.binned_statistic_2d(x, y, z, bins=[xbins, ybins], statistic="mean")
        clabel='Mean'
    elif statistic=="median":
        q = stats.binned_statistic_2d(x, y, z, bins=[xbins, ybins], statistic="median
        clabel='Median'
    elif statistic=="std":
        #we are doing this ourselves because the algorithm used by binned_statistic_2d
        #is remarkably slow
        if np.all(np.isreal(z)): #real-valued case
            q2    = stats.binned_statistic_2d(x, y, z**2, bins=[xbins, ybins], statis
            qbar  = stats.binned_statistic_2d(x, y, z, bins=[xbins, ybins], statistic
            q = np.sqrt(q2 - qbar**2)
        else:  #complex-valued case
            q2    = stats.binned_statistic_2d(x, y, np.abs(z)**2, bins=[xbins, ybins]
            qbarr = stats.binned_statistic_2d(x, y, z.real, bins=[xbins, ybins], stat
            qbari = stats.binned_statistic_2d(x, y, z.imag, bins=[xbins, ybins], stat
            qbar = qbarr + 1j* qbari
            q = np.sqrt((q2 - np.abs(qbar)**2).real)

        clabel='Standard Deviation'

    ax=plt.gca()

    im=ax.pcolormesh(xbins, ybins, np.transpose(q), cmap=cmap, shading="flat")
    if colorbar:
        cb=fig.colorbar(im, ax=ax)
        cb.set_label(clabel)

    if axisequal:
        ax.set_aspect("equal")

    if not(not(axlines)):
        ax.axhline(axlines[0], linestyle=":", color="k")
        ax.axvline(axlines[1], linestyle=":", color="k")

    if meanline:
        #plt.arrow(0,0,np.mean(x),np.mean(y),width=0.8,length_includes_head=False,face
        ax.plot([0,np.mean(x)],[0,np.mean(y)],color="w",linewidth=4.5)
        ax.plot([0,np.mean(x)],[0,np.mean(y)],color="k",linewidth=3)
```

```
    if meandot:
        ax.plot(np.mean(x),np.mean(y), "wo", markerfacecolor="k", markersize=8)

    plt.xlim([min(xbins), max(xbins)]),plt.ylim([min(ybins), max(ybins)])

    if not(tickstep==False):
        plt.xticks(np.arange(min(xbins), max(xbins), tickstep[0]))  # set x-label loca
        plt.yticks(np.arange(min(ybins), max(ybins), tickstep[1]))  # set x-label loca

    return im

fig,ax = plt.subplots(1,1)
bins = np.arange(-50, 50.01, 1)
plot_twodstat(bins,bins,cvr.real[:,3],cvr.imag[:,3],tickstep=(10,10),meanline=True,ax
plt.xlabel('Downstream Velocity (cm/s)'),plt.ylabel('Cross-Stream Velocity (cm/s)')
plt.title('Two-Dimensional Histogram of Velocity');
```



This plots shows the number of observations within each bin, which we have specified to be 1×1 cm/s bins ranging from -50 to 50 cm/s in both axes. The white bins are never observed.

The histogram of the velocity has a roughly elliptical shape, elongated along the x-axis. We now have a picture of a "cloud" of possibilities, with, generally speaking, bins closer to the location of the mean flow occurring more frequently.

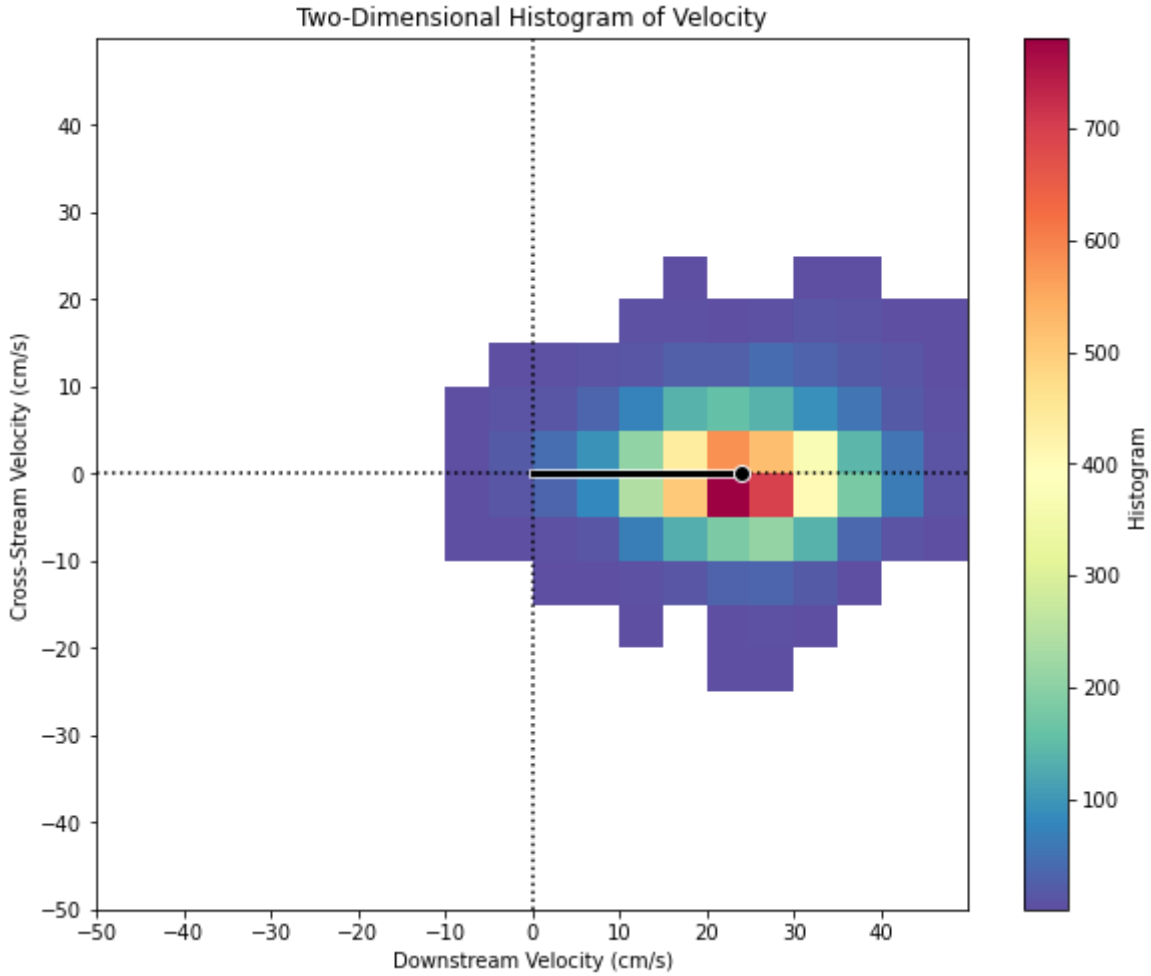
Note this plot contains more information than was visible in the line plot, because while the line plot also showed us the *shape* of the distribution, it did not provide us with information as to the relative frequency of occurrence of velocity values.

Adjusting 2D Histogram Plots

Two-dimensional statistics are an exceedingly powerly analysis tool. Before proceeding, let's take a look at how we can choose our settings to get the most out of them.

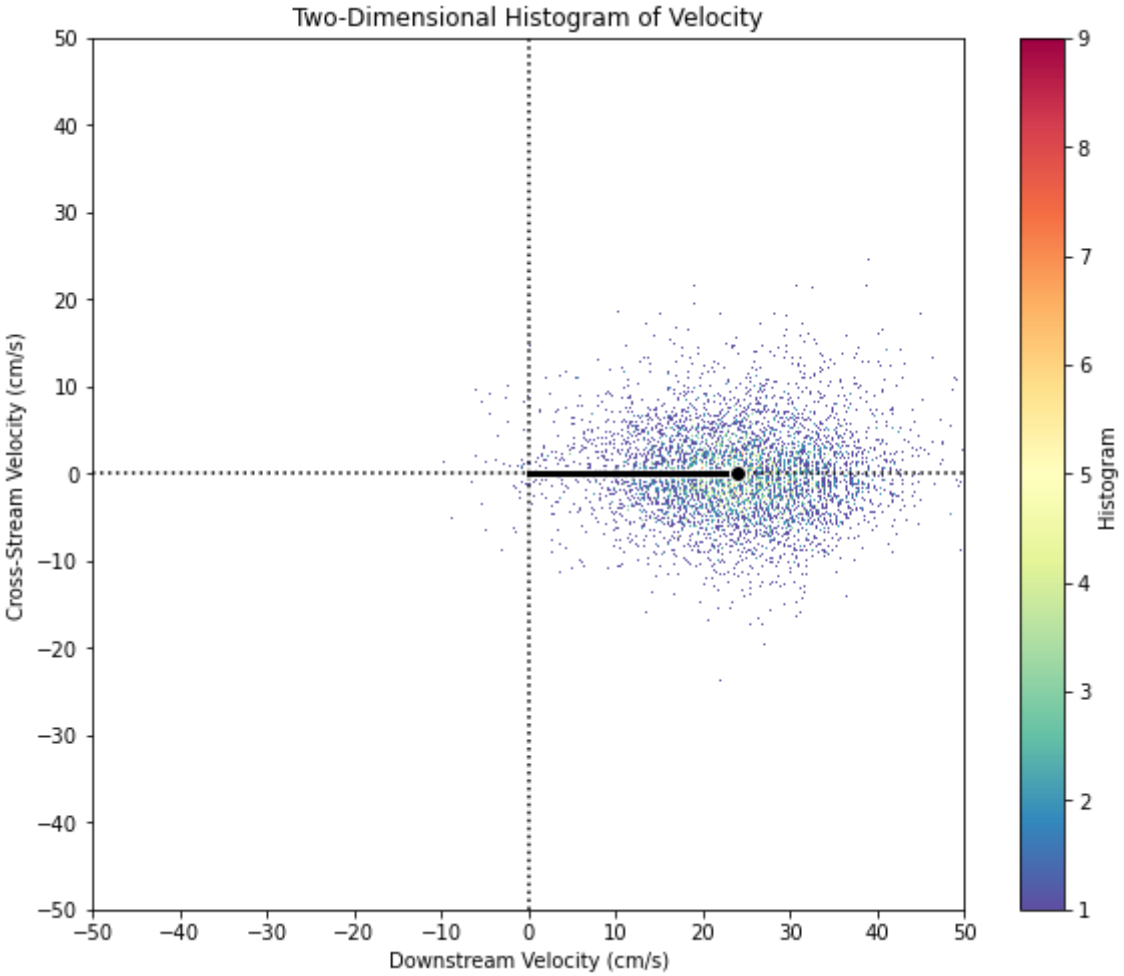
First, you'll want to pay attention to your choice of bin size.

In [14]:
```
#same plot as above but with x- and y-bins of [-50:5:50]
fig,ax = plt.subplots(1,1)
bins = np.arange(-50, 50.01, 5)
plot_twodstat(bins,bins,cvr.real[:,3],cvr.imag[:,3],tickstep=(10,10),meanline=True,ax
plt.xlabel('Downstream Velocity (cm/s)'),plt.ylabel('Cross-Stream Velocity (cm/s)')
plt.title('Two-Dimensional Histogram of Velocity');
```



Here, the bins have been chosen to be too coarse, so we can't see much structure, and the plot has a blocky look to it.

In [15]:
```
#same plot as above but with x- and y-bins of [-50:.2:50]
fig,ax = plt.subplots(1,1)
bins = np.arange(-50, 50.01, .2)
plot_twodstat(bins,bins,cvr.real[:,3],cvr.imag[:,3],tickstep=(10,10),meanline=True,ax
plt.xlabel('Downstream Velocity (cm/s)'),plt.ylabel('Cross-Stream Velocity (cm/s)')
plt.title('Two-Dimensional Histogram of Velocity');
```

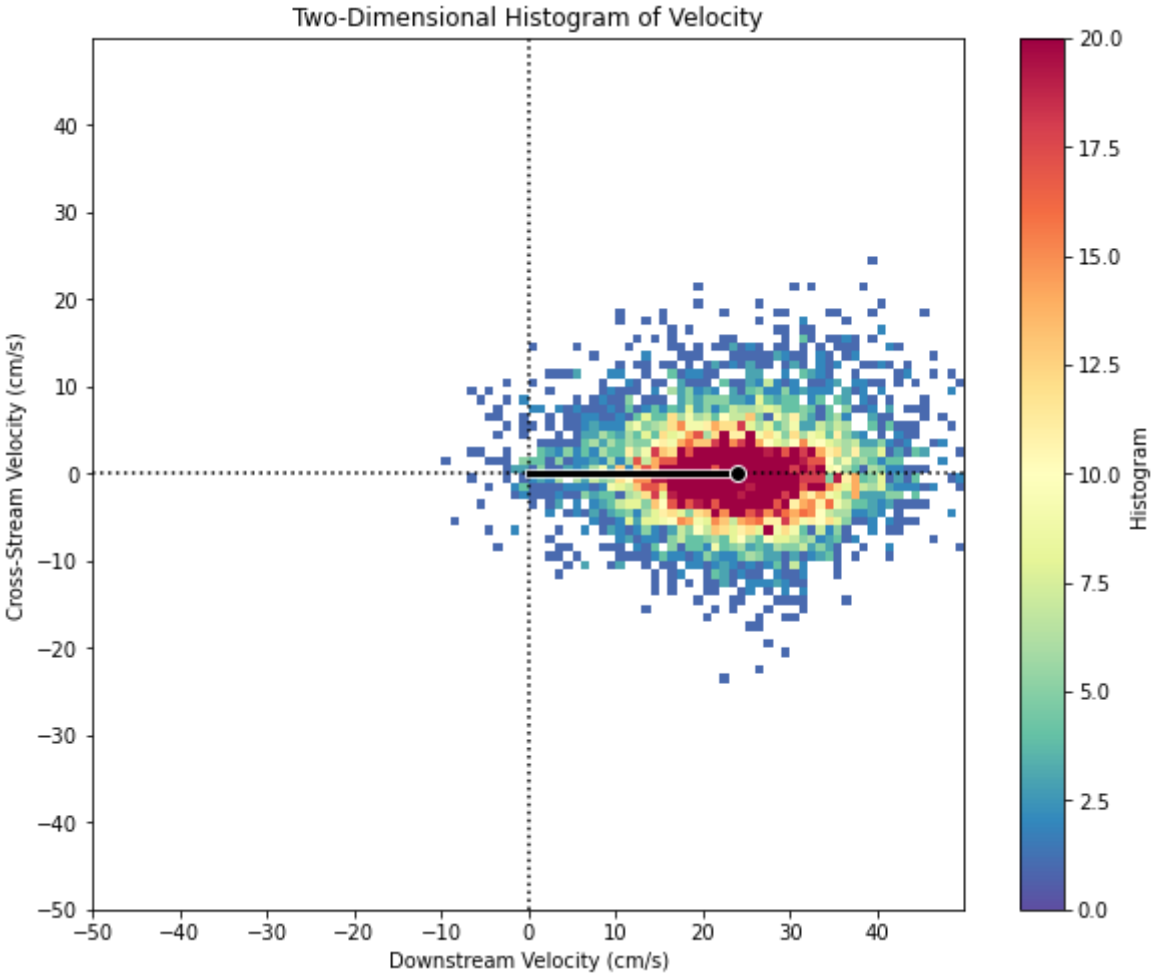


Here, the bins have been chosen to be too fine, so the "cloud" of points has been reduced to a mist, and again, we can't see much. Thus, you'll want to play around with your bin sizes to choose one that seems to reveal the most structure. The 1 cm/s bins we used earlier seem ideal for this dataset.

You'll also want to pay attention to your choice of color scale.

In [16]:

```
#same plot as above but with a different color axis
fig,ax = plt.subplots(1,1)
bins = np.arange(-50, 50.01, 1)
im=plot_twodstat(bins,bins,cvr.real[:,3],cvr.imag[:,3],tickstep=(10,10),meanline=True
plt.xlabel('Downstream Velocity (cm/s)'),plt.ylabel('Cross-Stream Velocity (cm/s)')
plt.title('Two-Dimensional Histogram of Velocity');
im.set_clim(vmin=0, vmax=20) #set colorbar limits to an inopportune value
```

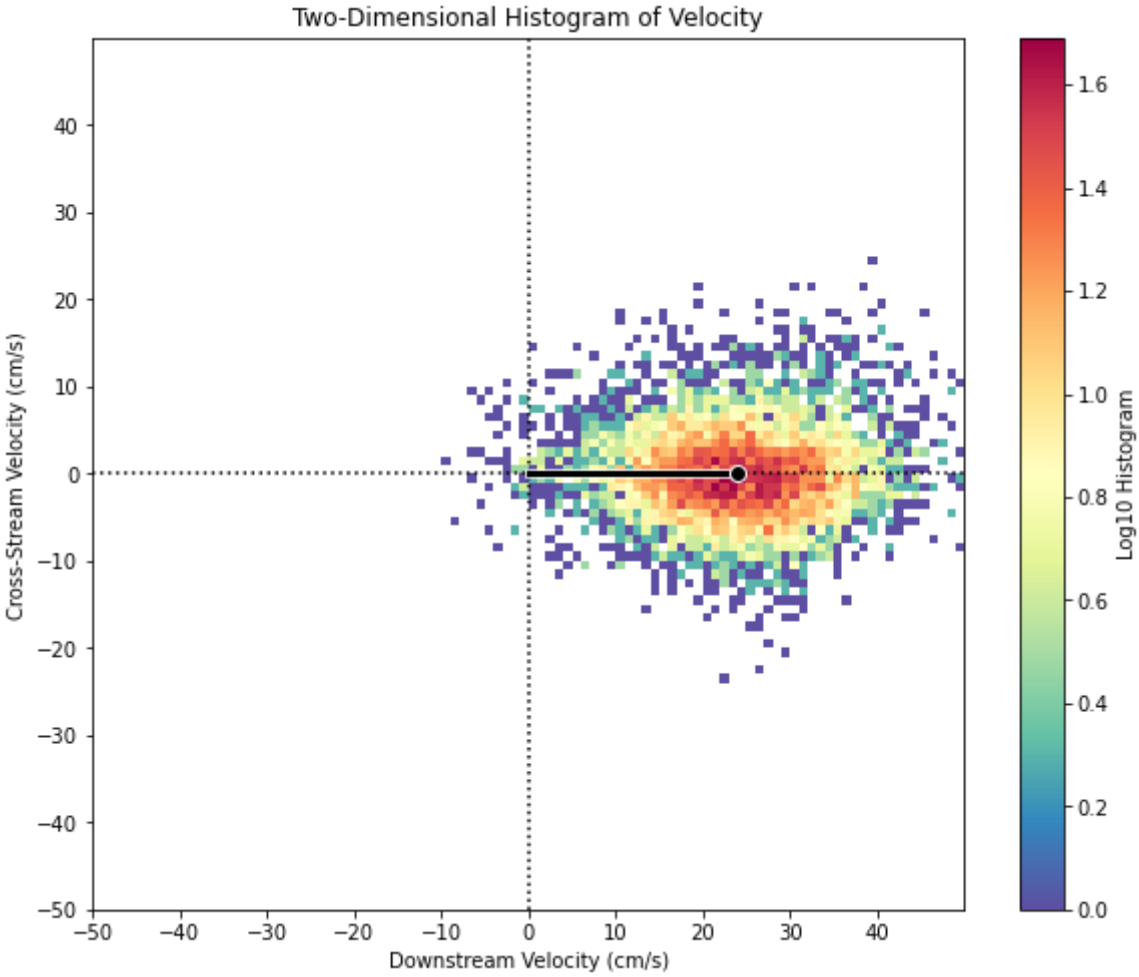


If you see a splotch of all one color in your dataset, one likely explanation is that the color axes limits have been set inappropriately, wiping out some of the information. Thus you'll want to check the color axes and see if an improvement could be made.

Finally, one useful trick is to plot not the number of observations per bin, but rather the *logarithm* of the number of observations, like so:

In [17]:

```
#same plot as earlier but with a logarithmic color axis
fig,ax = plt.subplots(1,1)
bins = np.arange(-50, 50.01, 1)
im=plot_twodstat(bins,bins,cvr.real[:,3],cvr.imag[:,3],tickstep=(10,10),statistic="lo
plt.xlabel('Downstream Velocity (cm/s)'),plt.ylabel('Cross-Stream Velocity (cm/s)')
plt.title('Two-Dimensional Histogram of Velocity');
```



This has the effect of allowing regions with radically different densities to be visualized together. The logarithm "flattens" the plot, since for exmaple, log10(100)=2, log10(10)=1, and log10(1)=0.

In this plot, the structure on the flanks of the distribution, where very few data points occur, now appears more visible. In general, you'll want to play around to see if linear or logarithmic histograms are more informative.

The Variance Ellipse

Finally we are ready to look at a quantity called the variance ellipse. We will study this more in the lectures; here we will just get a feeling for how it works.

In the code below, first we form the terms in the covariance matrix. Then the expressions for a, b, and theta amount to diagonalizing the covariance matrix with an eigenvalue decomposition.

In [18]:

```
def variance_ellipse(u,v):
    """
    Compute parameters of the variance ellipse.

    Args:
        u: 1-D array of eastward velocities (or real part of complex variable)
        v: 1-D array of northward velocities (or imaginary part of complex variable)

    Returns:
        a: semi-major axis
        b: semi-minor axis
        theta: orientation angle counterclockwise from x axis, in radians
```

```
    """

    #compute terms in the covariance matrix
    cuu=np.mean(np.multiply(u-np.mean(u),u-np.mean(u)))
    cvv=np.mean(np.multiply(v-np.mean(v),v-np.mean(v)))
    cuv=np.mean(np.multiply(u-np.mean(u),v-np.mean(v)))

    detc=np.real(cuu*cvv-cuv**2) #determinant of covariance matrix
    trc=cuu+cvv #trace of covariance matrix

    a=np.sqrt(trc/2+np.sqrt(trc**2-4*detc)/2)#semi-major axis
    b=np.sqrt(trc/2-np.sqrt(trc**2-4*detc)/2)#semi-minor axis
    theta=np.arctan2(2*cuv,cuu-cvv)/2#orientation angle

    return a,b,theta
```

Because the velocity consists of a *pair* of variables u and v, its second-order statistics are not just the variances of u and v separately. One must also take into account their covariance.

The diagonalization of the covariance matrix expresses this same information in a different way. It will be shown in class that a and b are the semi-major and semi-minor axes of an ellipse, respectively, while theta is the orientation of the major axis of the ellipse with respect to the x-axis.

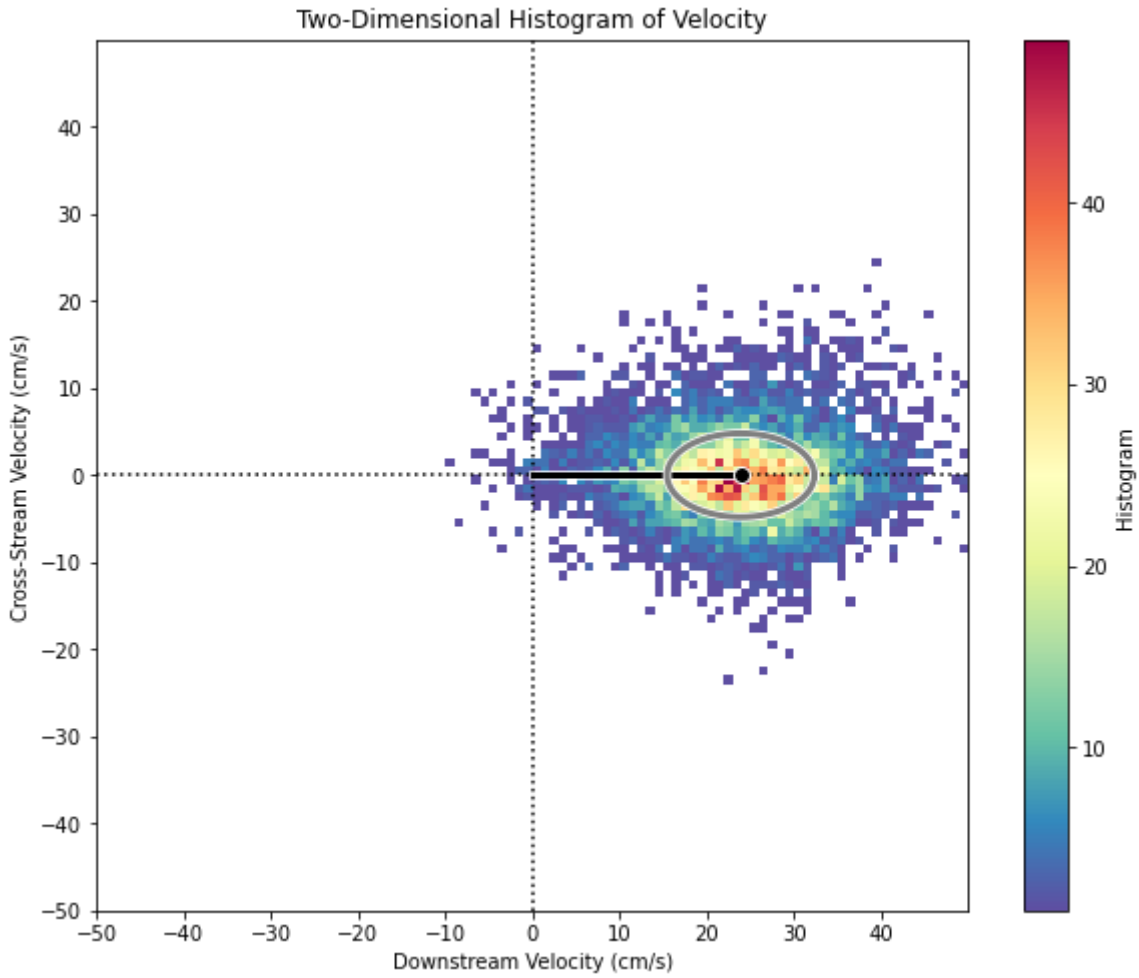
This quantity is called the variance ellipse. Just as the variance is the fundamental second-order statistical quantity for a single or univariate time series, such as u, the variance ellipse is the fundamental second-order statistical quantity for a pair of time series (u,v).

Now we return to the 2D histogram presented earlier, adding a plot of the variance ellipse.

In [19]:

```
#now we add a plot of the variance ellipse
def plot_ellipse(a,b,theta,center=(0,0),color=(0.5,0.5,0.5),linewidth=3,outline="w",a
    """
    Plot an ellipse.

    Args:
        a: Ellipse semi-major axis
        b: Ellipse semi-minor axis
        theta: Ellipse orientation angle in radians

    Optional Args:
        center: The ellipse center, a complex number or 2-tuple; defaults to (0,0)
        color: Ellipse line color, a 3-tuple; defaults to (0.5,0.5,0.5)
        linewidth: Ellipse line width; defaults to 3
        outline: Color for an outline around the ellipse; defaults to "w"
        aspect: Aspect ratio by which to adjust the ellipse; defaults to one
        transform: Coordinate transform to apply for use with Cartopy; defaults to nor

    Returns:
        h: Handle to ellipse object
        z: Array of complex numbers describing the ellipse periphery
    """
    phi=np.arange(0,2*np.pi+.1,.1)

    if isinstance(center,complex):
        x=center.real
        y=center.imag
    else:
```

```
        x=center[0];
        y=center[1];

    z=np.exp(1j*theta)*(a*np.cos(phi)-1j*b*np.sin(phi))
    ax = plt.gca()

    if not(transform):
        ax.plot(np.real(z)+x, (1/aspect)*np.imag(z)+y,linewidth=linewidth*1.5,color=o
        h=ax.plot(np.real(z)+x, (1/aspect)*np.imag(z)+y,linewidth=linewidth,color=col
    else:
        ax.plot(np.real(z)+x, (1/aspect)*np.imag(z)+y,linewidth=linewidth*1.5,color=o
        h=ax.plot(np.real(z)+x, (1/aspect)*np.imag(z)+y,linewidth=linewidth,color=col

    return h,z

#same plot as above using 1x1 cm/s bins
fig,ax = plt.subplots(1,1)
bins = np.arange(-50, 50.01, 1)
plot_twodstat(bins,bins,cvr.real[:,3],cvr.imag[:,3],tickstep=(10,10),meanline=True,ax
plt.xlabel('Downstream Velocity (cm/s)'),plt.ylabel('Cross-Stream Velocity (cm/s)')
plt.title('Two-Dimensional Histogram of Velocity');
#form ellipse parameters for rotated velocity at deepest depth
a,b,theta = variance_ellipse(cvr.real[:,3],cvr.imag[:,3])
plot_ellipse(a,b,theta,center=np.mean(cvr[:,3]));
```

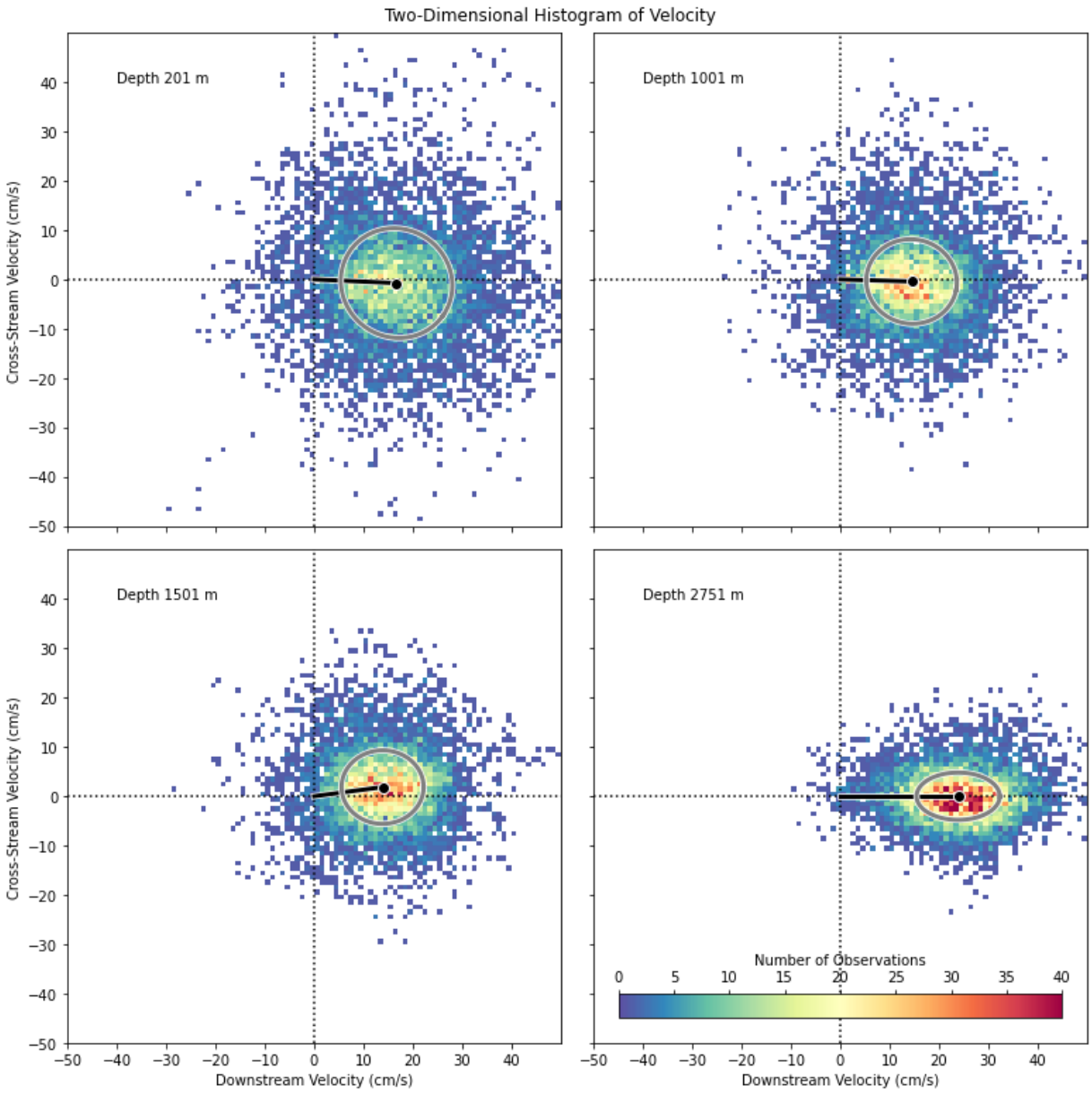


You can see that the basic shape of the velocity distribution is indeed capture by the variance ellipse. Variance ellipses are a powerful tool for describing the basic features of velocity fluctuations, as we shall see.

Next we will make the nicer version of the 2D histogram for all four depths, by just looping over the code blocks we have already used.

In [20]:

```
fig,ax = plt.subplots(2,2,figsize=(11,11),sharex=True,sharey=True)
ax=ax.flatten() # easier to iterate over

bins = np.arange(-50, 50.01, 1)

im=[]
for n in range(len(ax)):
    plt.sca(ax[n])
    im.append(plot_twodstat(bins,bins,cvr.real[:,n],cvr.imag[:,n],tickstep=(10,10),me
    plt.text(-40,40,'Depth ' + str(int(ds["depths"].data[n])) + ' m')

    a,b,theta = variance_ellipse(cvr.real[:,n],cvr.imag[:,n])
    plot_ellipse(a,b,theta,center=np.mean(cvr[:,n]));

    #label only the leftmost and lowermost axes
    if n==2 or n==3:
        plt.xlabel('Downstream Velocity (cm/s)')
    if n==0 or n==2:
        plt.ylabel('Cross-Stream Velocity (cm/s)')

    im[n].set_clim(vmin=0, vmax=40) #set colorbar limits identically on all plots

plt.suptitle('Two-Dimensional Histogram of Velocity')
plt.tight_layout()

#make an interior colorbar in the last subplot
divider = make_axes_locatable(ax[n])
cb = fig.colorbar(im[n], cax=ax[n].inset_axes((0.05, 0.05, 0.9, 0.05)),orientation='h
cb.ax.xaxis.set_ticks_position("top")
cb.ax.xaxis.set_label_position("top")
cb.set_label('Number of Observations')
```



This shows that as we proceed down in depth, the current variability becomes increasingly anisotropic.

Near the surface, the variabilty has no preferred orientation, and the variance ellipse is nearly ciruclar. Moving downwards, the distribution becomes increasingly "squished" in the cross-stream direction, leading to variance ellipses elongated along the x-axis. Another way to say this is that the velocity fluctuations become increasingly *polarized* with depth.

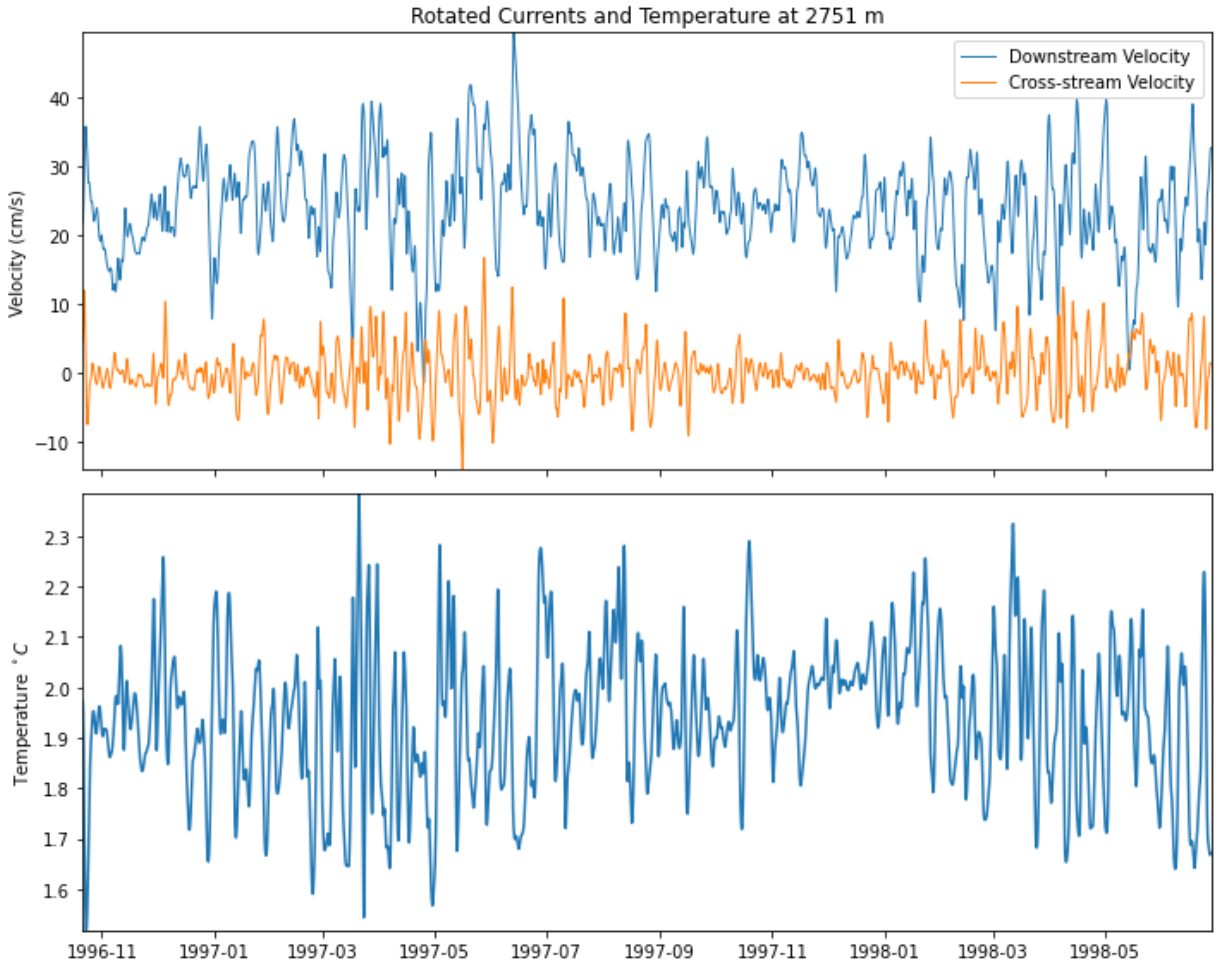
We can also see slight changes in the orientation of the mean flow relative to the mean flow at depth.

Velocity vs. Temperature Fluctuations

Let's move on now to take a look at the co-variability of temperature and velocity. First we will make a simple time series plot. This is just the same lowpass-filtered rotated velocity signal we've been looking at, with filtered temperature at the same depth plotted underneath it.

In [21]:
```
fig, ax = plt.subplots(2, 1, sharex=True)
```

```
window = sg.windows.hann(24)       #24-point hanning window
window = window / np.sum(window)  #normalized to sum to unity
fcvr =  si.convolve(cvr[:,3], window, mode="mirror")  #filtered version of velocity a
ft =  si.convolve(ds["t"][:,3], window, mode="mirror") #filtered version of temperatu

plt.sca(ax[0])
plot_complex_velocity(date,fcvr)
ax[0].legend(['Downstream Velocity','Cross-stream Velocity'])
ax[1].plot(date,ft)
ax[1].set_ylabel('Temperature $^\circ C$')
ax[1].autoscale(enable=True, tight=True)   # tighten axes limits around the data
plt.title('Rotated Currents and Temperature at ' + str(int(ds["depths"].data[3])) + '
fig.tight_layout()
```



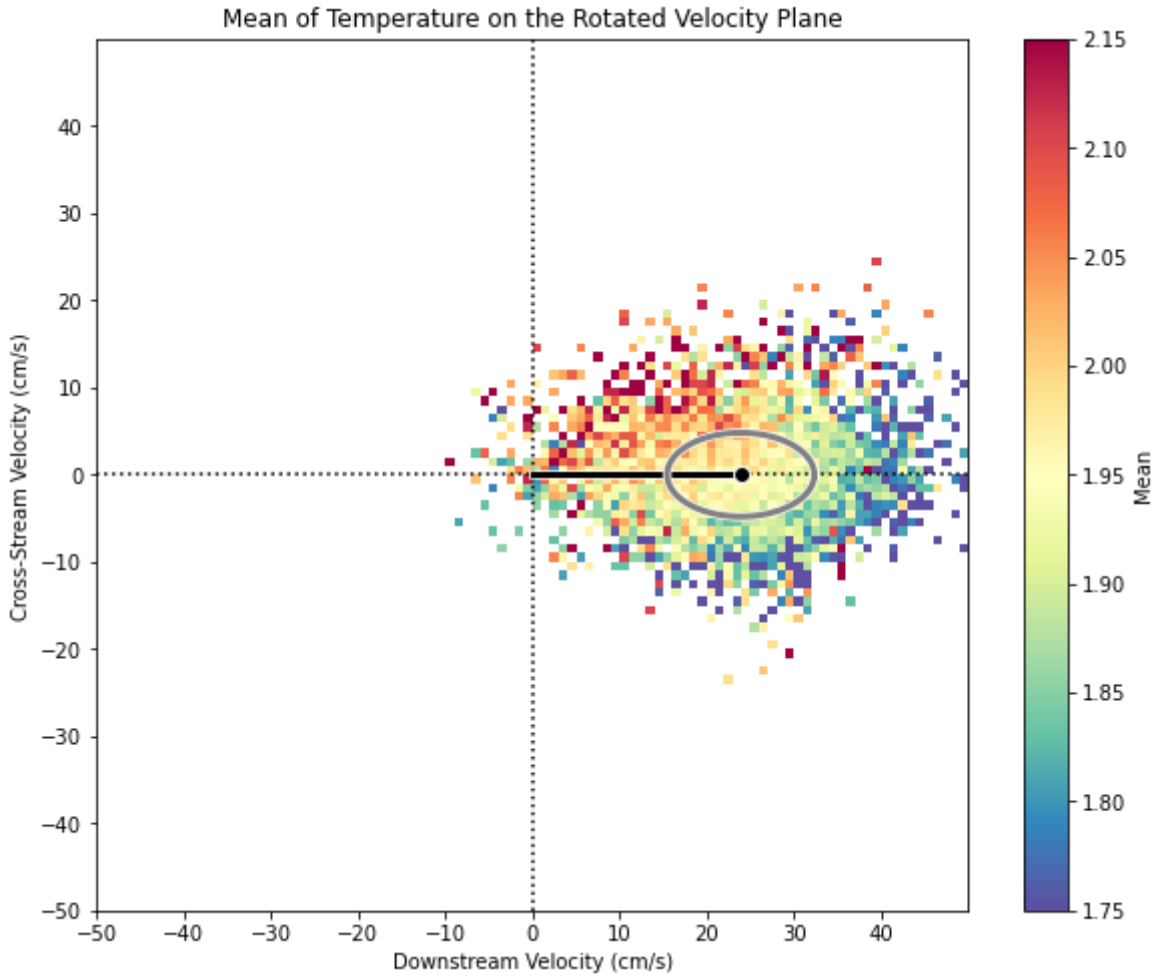
Temperature and velocity indeed present variability on similar timescales, but it's difficult to ascertain what is going on from looking at this plot. In particular, it's not clear whether or not there is meaningful co-variability.

To examine this we are going to look at the two-dimensional mean of temperature t as a function of the $(\tilde u, \tilde v)$ plane. This is also known as the *conditional mean*, because the mean of t is taken relative to a particular range of values of $\tilde u$ and $\tilde v$.

In [22]:

```
#same plot as above using 1x1 cm/s bins
fig,ax = plt.subplots(1,1)
bins = np.arange(-50, 50.01, 1)
im=plot_twodstat(bins,bins,cvr.real[:,3],cvr.imag[:,3],ds["t"][:,3],statistic="mean",
plt.xlabel('Downstream Velocity (cm/s)'),plt.ylabel('Cross-Stream Velocity (cm/s)')
plt.title('Mean of Temperature on the Rotated Velocity Plane');
a,b,theta = variance_ellipse(cvr.real[:,3],cvr.imag[:,3])
```

```
plot_ellipse(a,b,theta,center=np.mean(cvr[:,3]));
im.set_clim(vmin=1.75, vmax=2.15)
```



This shows that there is a definite pattern to temperature fluctuations with respect to velocity fluctuations. Warm temperatures tend to occur when the downstream flow is weak and the cross-stream flow is in the positive direction. Cold temperatures tend to occur when the downstream flow is strong and the cross-stream flow is in the negative direction.

The meaning of this pattern is not entirely clear, but it definitely suggests a particular relationship between the currents and temperature anomalies, perhaps associated with coherent eddies.

Moreover, we see that there is a tendency for velocity anomalies to systematically transport heat toward the positive cross-stream direction, which as will be seen later is offshore in this case.

Note that although a relationship is clear in the 2D histogram, it does not correspond to a simple correlation with downstream or cross-stream velocity. If we simply found correlation coefficients, we would miss the details of the pattern contained in this plot.

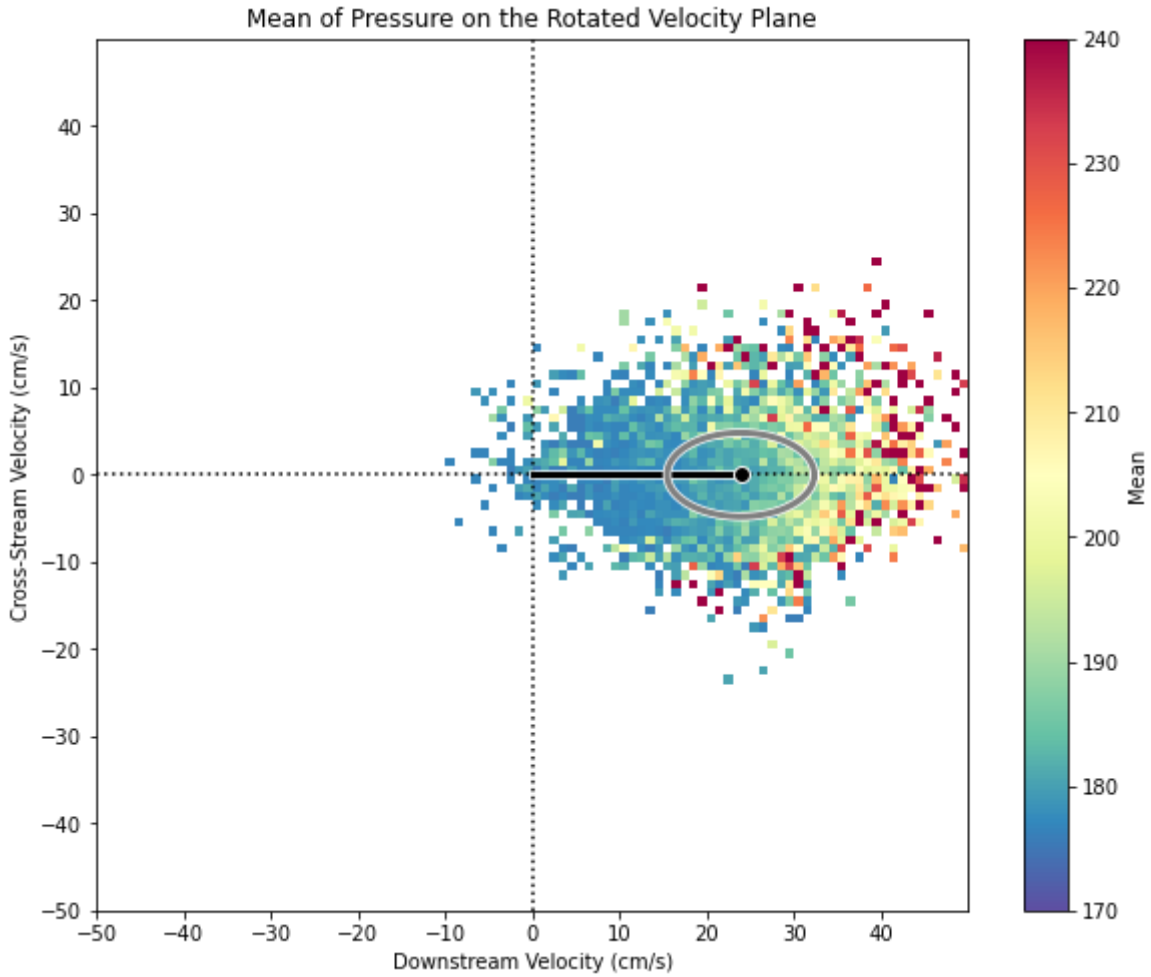
Let's examine the hypothesis that this pattern of temperature and velocity variability is an artifact of mooring towdown. That is, we conjecture that the mooring is going up and down due in the water column due to changes in drag, and this could interact with a pre-existing temperature gradient to generate the pattern we saw above as an artifact.

To examine this will make a plot as above, but this time with pressure as the variable we're taking the mean of. Because we don't have pressure at the deepest depth, we will substitute

pressure at where we do have it, at the shallowest depth.

Note, in this next figure we are plotting the mean value of the pressure observed *at the shallowest depth* as a function of the currents observed *at the deepest depth*.

In [23]:
```
#same plot as above using 1x1 cm/s bins
fig,ax = plt.subplots(1,1)
bins = np.arange(-50, 50.01, 1)
im=plot_twodstat(bins,bins,cvr.real[:,3],cvr.imag[:,3],ds["p"][:,0],statistic="mean",
plt.xlabel('Downstream Velocity (cm/s)'),plt.ylabel('Cross-Stream Velocity (cm/s)')
plt.title('Mean of Pressure on the Rotated Velocity Plane');
a,b,theta = variance_ellipse(cvr.real[:,3],cvr.imag[:,3])
plot_ellipse(a,b,theta,center=np.mean(cvr[:,3]));
im.set_clim(vmin=170, vmax=240)
```



This pattern is completely different. It shows that pressure is high when the speed is high, which occurs on the downstream fringe of the velocity distribution. This is just what we expect from mooring dynamics: when speeds are large, the instruments are dragged to deeper depths through the force of horizontal drag acting on the mooring line.

Indeed, had we reflected upon what the pressure should look like, we could have anticipated such a pattern. Therefore whatever is happening with the temperature is not an artifact of towdown. Rather, it appear to be physically meaningful.

Further examination of the temperature/velocity pattern will have to wait until another time. The point of this analysis has been to illustrate how simple plots, such as two-dimensional statistics, can be used to investigate physical hypotheses.

Summary Plots Using the Variance Ellipse

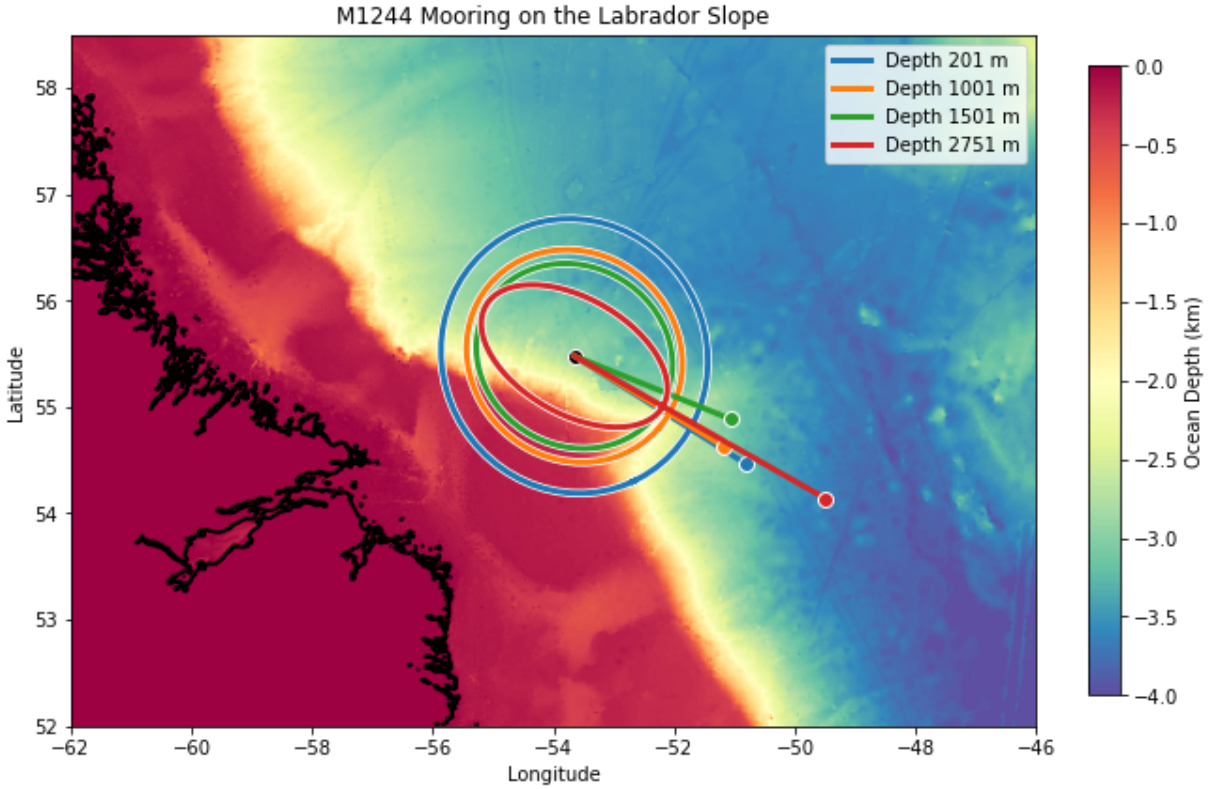
Now we will see how the variance ellipse can be combined with mean flow vectors to generate a summary plot. Here, we will plot the mean flow and variance ellipse of all four instruments at this mooring superposed on the bathymetry.

For this plot you'll need my NetCDF version version of the Smith and Sandwell one-minute global topography dataset. See here for more details.

In [24]:

```
#set this to the location where you put the sandwell.nc file
datadir = '/Users/lilly/Desktop/Dropbox/Matlab/jdata/sandwell/'
topo_ds = nc.Dataset(datadir+'sandwell.nc')

lona_index=np.argmax(np.array(topo_ds["lon"])>= -62.0) #index into westernmost longit
lonb_index=np.argmax(np.array(topo_ds["lon"])>  -46.0) #index into easternmost longit
lata_index=np.argmax(np.array(topo_ds["lat"])>=  52.0) #index into southernmost latit
latb_index=np.argmax(np.array(topo_ds["lat"])>   58.5) #index into northernmost latit

lons=topo_ds["lon"][lona_index:lonb_index]
lats=topo_ds["lat"][lata_index:latb_index]
topos=topo_ds["topo"][lona_index:lonb_index,lata_index:latb_index]

#Also need to interpolate lat and lon to the cell edges, since sandwell.nc is cell-cer
#while pcolormesh is expecting grid-centered.

M=np.size(lons)
flon=interp1d(np.linspace(0,M-1,M),lons,fill_value="extrapolate")
lons_gc=flon(np.linspace(-1/2,M-1/2,M+1)) #longitude values at cell edges

N=np.size(lats)
flat=interp1d(np.linspace(0,N-1,N),lats,fill_value="extrapolate")
lats_gc=flat(np.linspace(-1/2,N-1/2,N+1)) #latitude values at cell edges

#latitude and longitude of mooring location
lono=ds["lon"].data
lato=ds["lat"].data

fig,ax = plt.subplots(1,1,figsize=(10,10))
cmap=sns.color_palette("Spectral_r", as_cmap=True)
im=ax.pcolormesh(lons_gc,lats_gc,np.transpose(topos.data),shading="flat", vmin=-4, vm
plt.contour(lons,lats,np.transpose(topos.data),0,colors="k",linewidths=2)

aspect=1/np.cos(np.radians(lato))  #Aspect ratio; don't forget to convert latitude to
ax.set_aspect(aspect)

plt.plot(lono,lato,"wo", markerfacecolor="k", markersize=8) #plot mooring location

colors=('tab:blue','tab:orange','tab:green','tab:red') #colors for lines and ellipses
factor=1/5 #scaling factor for converting cm/s into degrees

#Next plot the ellipses and the mean flow vectors
h=[]
for n in range(np.shape(cv.data)[1]):
    a,b,theta = variance_ellipse(cv.data.real[:,n],cv.data.imag[:,n])

    plot_ellipse(a*factor,b*factor,theta,center=(lono,lato),aspect=aspect,color=color

    cvbar=factor*np.mean(cv[:,n])
    cvbar=cvbar.real+1j*cvbar.imag/aspect  #adjust vector for aspect ratio
```

```
    #the comma is essential in the output argument below, as without it the output of
    #will be a tuple rather than a line object, see https://matplotlib.org/2.0.2/users
    hn, = ax.plot([lono,lono+cvbar.real],[lato,lato+cvbar.imag],color=colors[n],linew
    h.append(hn)
    ax.plot(lono+cvbar.real,lato+cvbar.imag,"wo", markerfacecolor=colors[n], markersi

labels=[]
for depth in ds["depths"].data:
    labels.append('Depth ' + str(int(depth)) + ' m')

ax.legend(h,labels)

cb=fig.colorbar(im, ax=ax, fraction=0.03)
cb.set_label('Ocean Depth (km)')

plt.xlabel('Longitude')
plt.ylabel('Latitude')

ax.set_title("M1244 Mooring on the Labrador Slope");
```



Note that the mean flow is more or less aligned with the direction of the isobaths at the mooring location. This is a not uncommon result, particularly at higher latitudes where the stratification tends to be weaker and flows are more guided by bathymetry.

Then as we progress in depth variance ellipses are seen to become increasingly constrained to be parallel to the local isobaths as well, matching the orientation of the mean flow vectors. Again, it is fairly common (for oceanic currents, anyway) for the variance ellipse orientation to match the mean flow direction.

This plot is a good example of how the simple statitics of means and variances can be used to generate useful summaries of a whole dataset. It also shows how those statitics become more informative when placed into some kind of context, in this case, the context of the local topography.

Using Cartopy

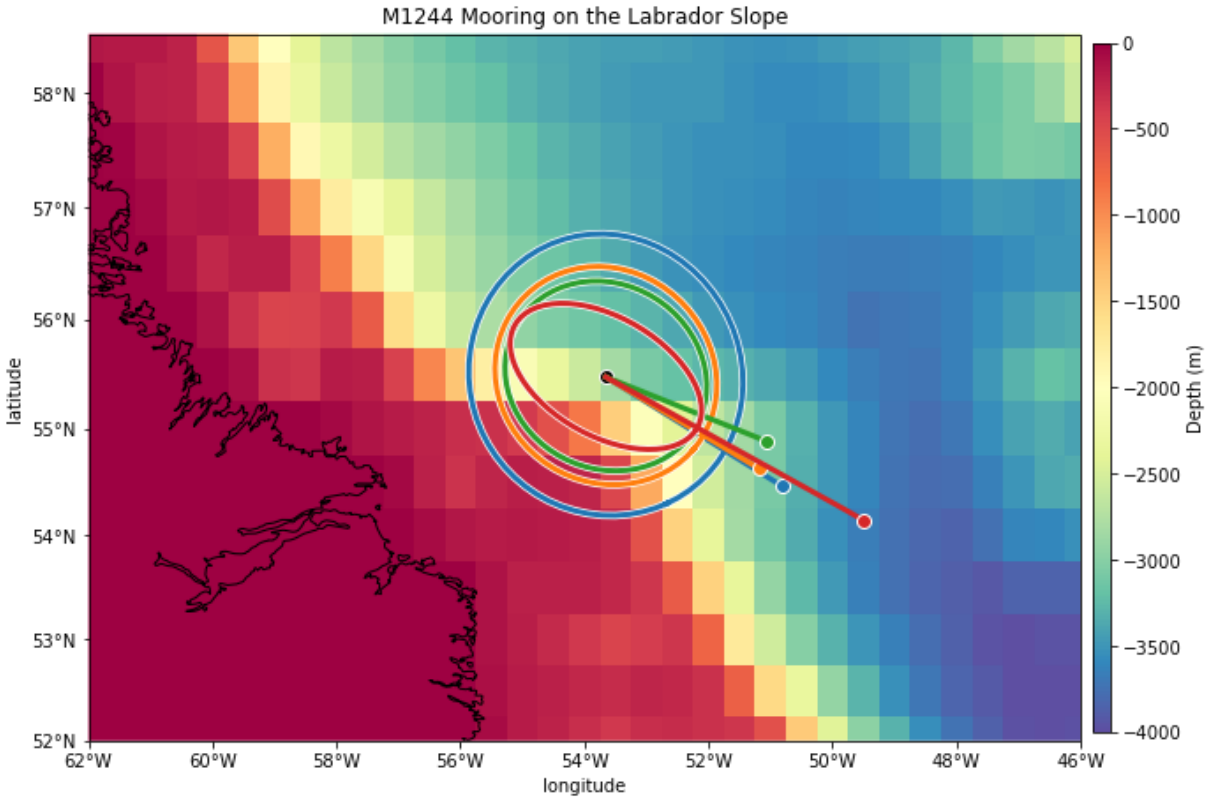
In case you are working with Cartopy for map projections, the example below makes the same plot using Cartopy. It's a bit of overkill for this particular example since we are dealing with such as small region. Also, it doesn't look as good because the bathymetry is coarser.

This example is included only for those who are already working with Cartopy and want to see how the ellipse plotting etc. is handled. Otherwise, feel free to skip it.

In [25]:

```
#Thanks to the examples at https://www.fatiando.org/harmonica/latest/sample_data/earth
#and https://uoftcoders.github.io/studyGroup/lessons/python/cartography/lesson/
#and https://scitools.org.uk/cartopy/docs/v0.16/gallery/tick_labels.html
#print(topodata)

import harmonica as hm  #for bathymetery data
topodata = hm.datasets.fetch_topography_earth() #fetch topograph data
from cartopy.mpl.ticker import LongitudeFormatter, LatitudeFormatter

plt.figure(figsize=(12, 12))

#first we set the map projection, in this case, a transverse Mercator
ax = plt.axes(projection=ccrs.Mercator(central_longitude=ds["lon"].data, \
    min_latitude=52, max_latitude=58.5, globe=None, latitude_true_scale=0.0))
ax.set_extent([-62, -46, 52, 58.5])

#add shading for topography
pc =  topodata.topography.plot.pcolormesh(
    ax=ax, transform=ccrs.PlateCarree(), add_colorbar=False, \
    cmap=sns.color_palette("Spectral_r", as_cmap=True), vmin=-4000, vmax=0
)

#set tick marks and make sure these are converted to be appropriate for our projection
ax.set_xticks(np.arange(-62,-45,2), crs=ccrs.PlateCarree())
ax.set_yticks(np.arange(52,59,1), crs=ccrs.PlateCarree())
lon_formatter = LongitudeFormatter(zero_direction_label=True)
lat_formatter = LatitudeFormatter()
ax.xaxis.set_major_formatter(lon_formatter)
ax.yaxis.set_major_formatter(lat_formatter)

#plot colorbar
plt.colorbar(
    pc, label="Depth (m)", orientation="vertical", aspect=40, pad=0.01, shrink=0.6
)

#plot latitude and longitude of mooring location
lono=ds["lon"].data
lato=ds["lat"].data
plt.plot(lono,lato,"wo", markerfacecolor="k", markersize=8, \
    transform=ccrs.Geodetic())

aspect=1/np.cos(np.radians(lato))  #Aspect ratio; don't forget to convert latitude to
colors=('tab:blue','tab:orange','tab:green','tab:red') #colors for lines and ellipses
factor=1/5 #scaling factor for converting cm/s into degrees longitude

#Next plot the ellipses and the mean flow vectors
h=[]
for n in range(np.shape(cv.data)[1]):
    a,b,theta = variance_ellipse(cv.data.real[:,n],cv.data.imag[:,n])
    plot_ellipse(a*factor,b*factor,theta,center=(lono,lato),aspect=aspect,transform=c
```

```
    cvbar=factor*np.mean(cv[:,n])
    cvbar=cvbar.real+1j*cvbar.imag/aspect

    hn, = ax.plot([lono,lono+cvbar.real],[lato,lato+cvbar.imag],transform=ccrs.Geodet
    h.append(hn)
    ax.plot(lono+cvbar.real,lato+cvbar.imag,"wo",transform=ccrs.Geodetic(), markerfac

#the labelling is not working at the moment
#labels=[]
#for depth in ds["depths"].data:
#    labels.append('Depth ' + str(int(depth)) + ' m')
#
#ax.legend(h,labels)

ax.coastlines()
ax.set_title("M1244 Mooring on the Labrador Slope")

plt.show()
```



The End