Loading libraries:

```
addpath 'fromMarksLibraries' -END
addpath 'CircStat2010d' -END
```

The first is a library of functions that I use often. Most of these are plotting functions. They are used by some of the other functions called below. You may also find 'blankFigure' & 'AxisMMC' to be generally useful for making publication quality plots.

'CircStat2010d' is a library I did not write. It is used by jPCA to compute some summary statistics related to the strength of the rotational tendency of the data. If you cite such numbers in papers, you must mention that you used this library and must credit the authors.

Loading the example data:

```
load exampleData
```

Inspect the structure 'Data'. Each of the 108 elements is one 'condition': one reach type that our monkey performed. The t x n matrix 'A' contains the mean firing rate (averaged across trials) as a function of time for each of 218 neurons. The 't.imes' field is optional but very useful. In our case it indicates that time starts 50 ms before a key event (described below) and ends 550 ms after that event. We sampled the firing rate every 10 ms. Time zero is the 'neural movement onset': the time when neural activity began changing rapidly after the go cue was given[1].

You can of course make any choice for the timebase (e.g., you could measure the firing rate every 1 ms, 10 ms , or whatever). You can also make any choice for what is time zero (e.g., it could be the first time, or the time a movement or stimulus starts). Make an intelligent choice, and instantiate it in the '.times' field. This will make future time indexing easy.

Setting some basic parameters:

```
% these will be used for everything below
jPCA_params.softenNorm = 5;  % how each neuron's rate is normized, see below
jPCA_params.suppressBWrosettes = true;  % these are useful sanity plots, but lets ignore them for now
jPCA_params.suppressHistograms = true;  % these are useful sanity plots, but lets ignore them for now
```

Ex 1: Plotting the first jPCA plane

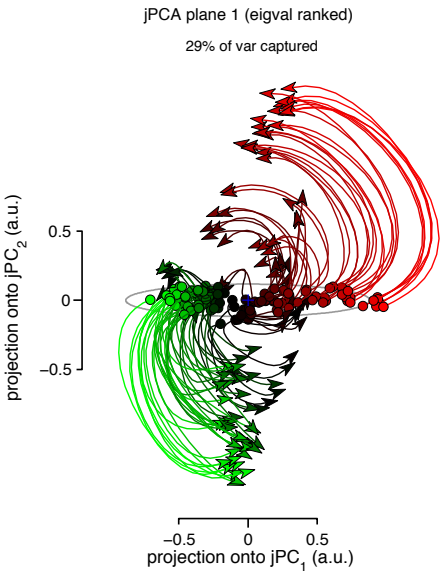
For 200 ms of data, using 6 PCs (the default)

```
times = -50:10:150;  % -50 to 150 ms w.r.t 'neural movement onset'
jPCA_params.numPCs = 6;  % default anyway, but best to be specific
[Projection, Summary] = jPCA(Data, times, jPCA_params);

phaseSpace(Projection, Summary);  % makes the plot
printFigs(gcf, '.', '-dpdf', 'Basic jPCA plot');
% prints in the current directory as a PDF
```

Open the PDF. The plot should look like that to the right.

You can pass parameters to 'phaseSpace' to control things like arrow size, the starting circle size, whether there are axes, etc.



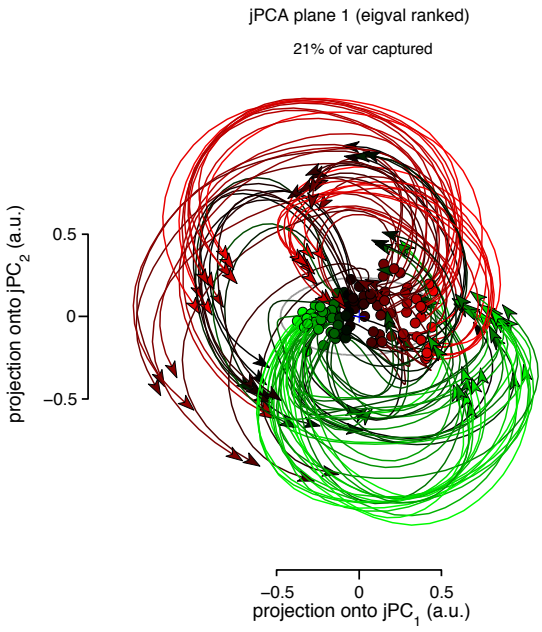
[1] Example data are from monkey N, recorded using a pair of arrays on 9-23-2010. The data contain both single units and good multi-unit isolations. These data were used to produce Fig. 3f of Churchland et al., 2012. However, for reasons of file size I'm including only the key range of times here (the range you actually want to analyze). Normalization (a step performed by jPCA) is thus acting across a shorter range of times than in the paper. So the results here are *very* slightly different. But you will have to look close to tell.

Ex 2: Plotting the first jPCA plane: a greater range of time

```
times = -50:10:300;  % 50 ms before 'neural movement onset' until 300 ms after
jPCA_params.numPCs = 6;  % sticking with 6 for now, but will move up to 10 below
[Projection, Summary] = jPCA(Data, times, jPCA_params);
phaseSpace(Projection, Summary);  % makes the plot
```

jPCA tries to find the best plane, among the top PCs, for the range of time being analyzed. That means that the projection changes when we ask for more time. The jPCA plane is being 'tilted' a bit to find the projection that works best for this broader range of times. In doing so, slightly less variance is captured.

This is the reason why the plot at the right is not just an extended version of the plot found when using -50 to 150 ms.



Ex 3: Plotting the first jPCA plane: more time and more dimensions

In Churchland et al., 2012, most analyses were restricted so that jPCA searched for a 'good' plane within only the top 6 PCs. This was done to be conservative: the more PCs one allows jPCA to search through, the greater the concern that rotations are found 'by chance' (e.g., could have been found for random data). In practice this is not a large concern. If the data contain many conditions, and if we restrict ourselves to some reasonable number of PCs, jPCA will not erroneously find rotations.

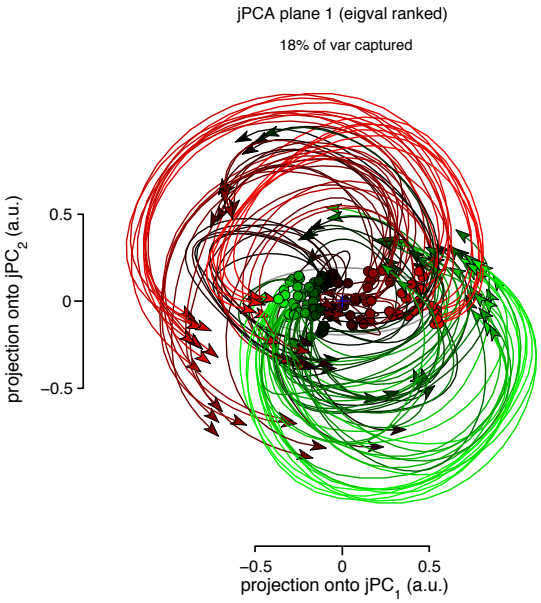
```
times = -50:10:300;
jPCA_params.numPCs = 12;  % use the top 12 PCs
[Projection, Summary] = jPCA(Data, times, jPCA_params);
phaseSpace(Projection, Summary);  % makes the plot
```

The real tradeoff is this: by using more PCs we allow jPCA to find a slightly better plane. However, that plane will typically capture slightly less data variance. Often this is worth it. Especially if we wish to find more than one jPCA plane (see below).

However, one would not wish to go overboard. Currently the code limits you to 20 unless you override it. I don't see any reason to go above 12, because 12 PCs captures most of the data variance anyway.

Allowing too many planes will eventually allow jPCA to find 'spurious' rotations. However, that error would be fairly obvious: the data variance captured would be small (e.g., 3%).

Note that '.numPCs' must always be an even number.

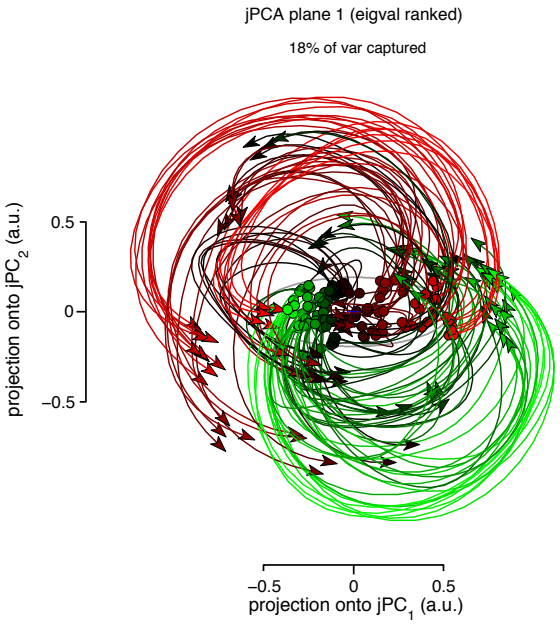


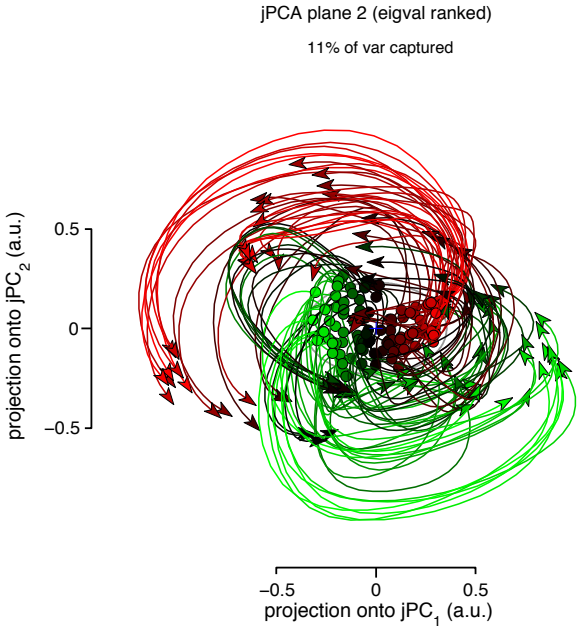
Ex 4: Plotting multiple jPCA planes

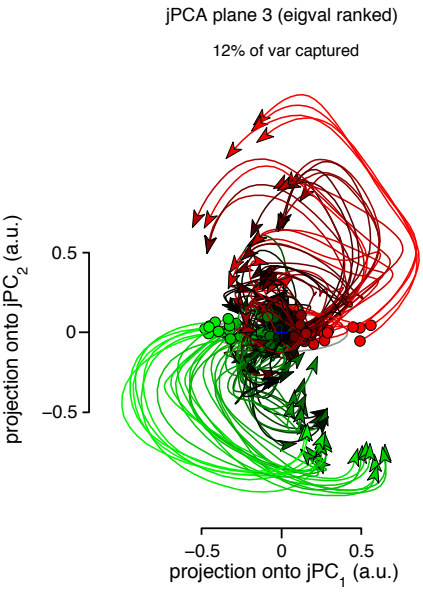
As described in Churchland et al., 2012, the data contain not just one plane with rotations, but two or three orthogonal planes (see Supplementary Figure 4).

```
times = -50:10:300;  % first we will just run the analysis as we did above in example 3
jPCA_params.numPCs = 12;
[Projection, Summary] = jPCA(Data, times, jPCA_params);

% now we will plot all three planes
plotParams.planes2plot = [1 2 3];
phaseSpace(Projection, Summary, plotParams);  % makes all three plots
```







As you would guess, to see three planes, it is not sufficient to run the analysis using 6 PCs (what are the chances that the first 6 PCs are nicely aligned with the actual planes of rotation?). If you restrict yourself to the top 6 PCs, you typically only see 1-2 good planes. Things look good for three planes only once you employ 10-12 PCs.

These three planes together capture 41% of the variance. To see this type 'sum(Summary.varCaptEachJPC(1:6))'. This is less than for any of the analyses in the paper (those ranged from 50-70%). The reasons for this are that 1) we are normalizing slightly differently here, 2) this dataset was always at the low end, and 3) we are using 12 PCs instead of 10.

As indicated above, these three planes reflect the usual tradeoff that comes with using more PCs: projections that capture cleaner dynamic structure, but less data variance. Try running the above analysis with numPCs = 10. The projections still look quite good and the total variance captured by the three planes rises to 47% (this is still very much on the low end, but that is just the nature of this particular dataset).

We hope to devise a method that naturally finds the right balance between clean structure captured and overall structure captured. But until then, you have to strike that balance yourself by picking a sensible number of PCs (typically 6-12).

Ex 5: Movies of the neural trajectory evolving

Inspecting rotational structure across a long range of times becomes annoying in a static plot, as the traces start to all overlap. Fortunately, you can easily make a movie.

```
times = -50:10:300;  % This is just what we did for examples 3 & 4 above
jPCA_params.numPCs = 12;
[Projection, Summary] = jPCA(Data, times, jPCA_params);

phaseMovie(Projection, Summary);  % plots the movie
```

Your movie should exactly trace out the static figure.

You can also plot more times without changing the projection. Lets say you analyze -50:10:300 ms. This is a good range, because the neural state is in flux pretty much the whole time. This is what you want to best allow jPCA to find a good projection. But perhaps you later wish to plot the end of the neural trajectory, as it is decaying back to 'baseline'. You probably wish to do this without changing the projection. jPCA handles this situation easily.

```
% for a greater range of times
movParams.times = -50:10:550;  % a range of times that is broader than that used to find the jPCA
projection
phaseMovie(Projection, Summary, movParams);
```

phaseMovie will show you all the times that match the times you ask for. E.g., if you asked for -1000:1:550 you would just get `-50:10:550;`

Ex 6: Saving movies

Here are instructions on how to export and view your movies. This gets a little annoying. But not too much.

```
movParams.times = -50:10:550;
movParams.pixelsToGet = [25 35 280 280]; % These WILL vary depending on your monitor and layout.
MV = phaseMovie(Projection, Summary, movParams);

figure; movie(MV);  % shows the movie in a matlab figure window

movie2avi(MV, 'jPCA movie', 'FPS', 12, 'compression', 'none'); % 'MV' now contains the movie
```

You will have to play with '.pixelsToGet' until you have the movie framed right. You may get a warning if you ask for too large a span of pixels.

Somewhat strangely, 'movie2avi' uses an old codec. The movie will not open in the most recent Quicktime. It will open in Quicktime 7 pro or in VLC. You can then save it in a new codec, and compress it to a reasonable size. There may be a better solution to this – I have only looked a little. One more odd compatibility issue: even though the most recent Quicktime will not play this codec, Keynote plays is nicely. Go figure.

Ex 7: Rotating from the PCA basis to the jPCA basis

This is a recent, and not particularly well-documented feature. Have fun with it.

```
% going back to the original 6D analysis over a short period of time.
times = -50:10:150;  % 50 ms before 'neural movement onset' until 150 ms after
jPCA_params.numPCs = 6;  % default anyway, but best to be specific
[Projection, Summary] = jPCA(Data, times, jPCA_params);

phaseSpace(Projection, Summary);  % Static plot as a sanity check

rotationMovie(Projection, Summary, -50:10:150, 70, 70);  % 70 steps to full rotation.  Show all 70.
```

Ex 8: The benefit (and pitfalls) of removing the cross-condition mean

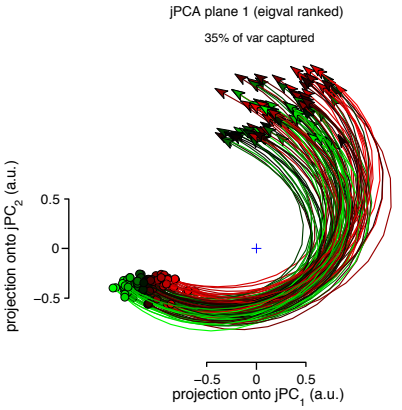
All the jPCA plots in Churchland et al. 2012 employed the following pre-processing step: before further analysis (PCA, jPCA or whatever) the cross-condition mean was first removed. This was done on a per-cell basis, and in the same way for recorded and simulated data. See supplementary figure 11 for a full explanation regarding how and why this was done. The brief explanation is that in M1, there is a large, slow, condition-independent component to the response, with the condition-specific component riding on top. Thus, if one considers the neural trajectories projected onto the first few principal components, they are very often similar for all conditions.

This is interesting scientifically: the largest component of the response is often not movement specific, and seems to just care about whether a movement (any movement) is being generated. Yet for present purposes we have a more practical problem: the large condition-independent component is an impediment to applying jPCA in a useful way. Without mean subtraction, the first jPCA plane is dominated by a big loop that is similar for all conditions. Such a loop is certainly consistent with rotational dynamics, but makes only a very weak argument for such dynamics. One really wants multiple views of how the neural trajectory unfolds with time. When all the conditions do the same thing, one gets only a single view.

There are two ways around this problem. The first is to look past the first plane:

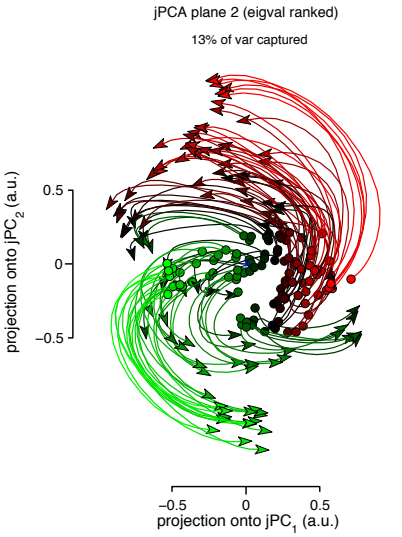
```
% turning off mean subtraction
times = -50:10:150;  % 200 ms of time, as in supp fig. 7
jPCA_params.meanSubtract = false;  % no mean subtraction
jPCA_params.numPCs = 10;  % as in supp fig. 7
[Projection, Summary] = jPCA(Data, times, jPCA_params);

plotParams.plotPlanEllipse = false;
plotParams.planes2plot = [1 2 3];
phaseSpace(Projection, Summary, plotParams);
```



The three planes are shown to the right. The first plane captures a largely non-condition-specific component. The next two planes capture response aspects that differ more strongly across conditions. These resemble what one would have got in the first plane, if one had used mean subtraction.

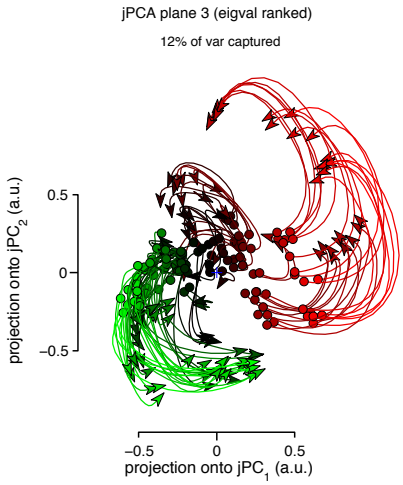
Note that when the mean is not subtracted out, a large chunk of variance is consumed by aspects of the data that do not differ across conditions. Necessarily, the projections that do capture condition-specific components capture less of the data variance.



Again, one typically wants projections that capture condition-specific aspects of the underlying response. The first solution is that above: look beyond the first jPCA plane. The second is to subtract the cross-condition mean, as in all the examples above. Then all the remaining data variance is condition specific. Usually this is the better choice, though not always. An obvious contra-indication would be if different conditions were different in some fundamental way (e.g., eye movements versus arm movements) such that subtracting off a single cross-condition mean might not make sense.

Still, lets assume that you will usually keep the default setting, and subtract the cross-condition mean before finding the projections. You might still wish to see what that mean would have done in the projection you found. To illustrate, let's return to our very first example, but plot a cyan trace to show how the cross-condition mean travels in this plane

```
times = -50:10:150;
jPCA_params.meanSubtract = true;  % the default
jPCA_params.numPCs = 6;
[Projection, Summary] = jPCA(Data, times, jPCA_params);
plotParams.crossCondMean = true;
plotParams.planes2plot = 1;
phaseSpace(Projection, Summary, plotParams);
```



(resulting fig not reproduced here).