DataHigh: A Brief User's Guide (Oct 28, 2013)

DataHigh V1.0
© 2013
Ben Cowley, Carnegie Mellon University
Matt Kaufman, Cold Spring Harbor Laboratory
Zachary Butler, University of California-Irvine
Byron Yu, Carnegie Mellon University

DataHigh is a Matlab GUI for visualizing high-dimensional neural data. DataHigh can be used to study how neural population activity varies across experimental trials, as well as across experimental conditions. It can be used to visualize single-trial neural states, single-trial neural trajectories, and trial-averaged neural trajectories. The input data can either be latent variables extracted by a dimensionality-reduction method (e.g., principal component analysis, factor analysis, or Gaussian-process factor analysis) or raw spike counts.

For more information, see:

DataHigh Website link:

[www.ece.cmu.edu/~byronyu/software.shtml]

PLACE JNE DATAHIGH PAPER REFERENCE HERE

"DataHigh: Graphical user interface for visualizing and interacting with high-dimensional neural activity" by B.R. Cowley, M.T. Kaufman, M.M. Churchland, S.I. Ryu, K.V. Shenoy, and B.M. Yu. Submitted to EMBS Annual Meeting 2012.

Contact:

All questions, comments, and suggestions should be sent to:

DataHigh@gmail.com

==========
How to get started with example data
==========

1. Navigate to the `examples` folder in the main DataHigh directory.
2. Enter the following commands into the Matlab console to try different examples:
   Visualize neural states: `examples2_neuralstates`
   Visualize single-trial neural trajectories: `examples3_singletrial`
   Visualize trial-averaged neural trajectories: `examples4_trialaveraged`
   Dimensionality reduction: `examples1_dimreduce`

==========
How to get quickly started with your own data
==========

1. Format raw spike trains into a Matlab struct, `D`. `D(i)` refers to the `i`th trial. The `D(i).data` field contains a matrix of size *number of neurons* x *number of timepoints*.
2. Call `DataHigh(D,'DimReduce');` in the Matlab command line.
3. Use the **DimReduce** tool to apply a dimensionality reduction technique (e.g., GPFA). Upload to **DataHigh**.
4. Click and hold on a preview panel to rotate the main 2-d projection.
5. Use the **3-d Projection/Evolve** tools under the **Analysis Tools** tab to view the neural trajectories play out over time.
6. Capture and save any noteworthy projections for further analysis.

==========
User Guide Outline
==========

Data demos (with steps)
What to do if the trajectories look like spaghetti
Analysis Tools
Data format for DimReduce
Data format for DataHigh
Noted problems
FAQ

==========
Data demos (with steps)
==========
This section provides detailed tutorials about how to use DimReduce and DataHigh. Please see `examples.m` for more information. It may first be advantageous to view the instructional videos that go step-by-step for each example on the DataHigh website (`www.ece.cmu.edu/~byronyu/software/DataHigh/`), and then try the steps yourself with the text reference.

Example 1: Extract neural states from spike count vectors

data file: 'ex1_spikecounts.mat'
description: This is an example dataset to get familiar with dimensionality reduction. The dataset contains spike trains recorded simultaneously from 61 neurons in the premotor cortex during a standard delayed reaching task. There are 30 reaches to each of seven reach targets. The user is instructed to perform dimensionality reduction and automatically upload the extracted neural states to DataHigh.

Santhanam G, Yu BM, Gilja V, Afshar A, Ryu SI, Sahani M, Shenoy KV (2009) Factor-analysis methods for higher-performance neural prostheses. Journal of Neurophysiology. 102:614-635.

Detailed Tutorial:
The same tutorial and an instructional video can be found on the DataHigh website.

1. Navigate to the examples folder in the main DataHigh directory. Enter **ex1_dimreduce** into the Matlab command console to input raw spike trains into DimReduce. The example dataset contains seven different experimental conditions. Each condition has 30 trials, and each trial has spike trains from 61 neurons. Each spike train is 400ms long and binned at 1ms resolution.
2. The DimReduce figure will pop up. The top right corner has Next Step instructions, so that the user can walk through the steps of performing dimensionality reduction. Note that clicking **Next Step** will move the large red number to the next location where a choice is needed by the user. Each option also has a **?** button nearby to provide more information about that step.
3. For this example dataset, in the **time bin width** box, enter 400ms. This will change the "Type" to "States." For neural states, you should choose the maximally allowed time bin width, which is the length of the shortest trial. This ensures one bin is used per trial---longer trials are truncated. If you need to align your data to a stimulus event (such as a go cue), align the data first before inputting it into DimReduce. Also, to avoid possible confusion, ensure that each trial has the same length.
4. For the **mean spikes/sec threshold**, enter 1.0 spike/sec. DataHigh removes 4 neurons from the analyses. Neurons with firing rates less than 1 spike/sec can sometimes be problematic, so we leave them out.

5. Under **Method**, choose "FA" for Factor Analysis. For neural states, we suggest FA because it allows each neuron to have a different amount of Poisson-like spiking variability. Since FA uses an iterative algorithm to fit the model parameters, it can take seconds to minutes to complete cross-validation.
6. For the **Candidate dimensionality** set, input "[1:15 20 30 40 50]". This input is in Matlab vector notation. Note that the terms “dimensionality” and “number of latent variables” are interchangeable, and by choosing a candidate dimensionality set we are trying to find how many latent variables best explain the data. For this range, DimReduce will perform cross-validation on candidate dimensionalities 1 through 15, 20, 30, 40, and 50. Trying all possible candidate dimensionalities (1 through 57) is computationally expensive, and typically unnecessary for visualization purposes, so we sample from the set of possible candidate dimensionalities.
7. Click **Perform cross-validation** to perform cross-validation on the set of candidate dimensionalities. Cross-validation tells us how many dimensions are needed to describe the data. The dataset is split into three folds. Factor analysis trains its parameters on two of the folds, and then tests on the remaining fold. This is done for each candidate dimensionality. A progress bar will pop up. Wait until it is completed.
8. Cross-validation has been performed, and we can view metrics to select an optimal dimensionality for visualization. Look at the LL metric by clicking the **LL** button in Dimensionality Plots. LL plots the cross-validated log-likelihood versus the candidate dimensionality. For this dataset, the log-likelihood has its peak at a dimensionality of 6, denoted by the star.
9. Click the **Proj** button to plot the first two factors. We see some clustering of the datapoints (which makes sense, since the data are from different experimental conditions, and some neurons are more active for some conditions than for other conditions).
10. Slide the **Select dimensionality** scrollbar until it shows 6. Typically, you should select the optimal dimensionality as determined by the cross-validated log-likelihood.
11. Click **Perform dim reduction**. DimReduce will perform dimensionality reduction for the chosen number of dimensions (i.e., factors). The data's dimensionality is reduced from 57 dimensions (i.e., neurons) to 6 latent dimensions. A PostDimReduce figure pops up. This figure allows you to make a more informed decision about your dimensionality selection. If the selected dimensionality is not appropriate, you can close the PostDimReduce figure to go back to the original DimReduce figure.
12. In the PostDimReduce figure, click **Upload to DataHigh**. The main DataHigh interface pops up. You may change the colors of the clusters by clicking the **Analysis Tools** tab in the Toolbox and then click the **Update Colors** button. In the UpdateColors figure, change the Epoch Colors field of the condition "reach1" from [0 0 1] to [1 0.5 0]. This will change the neural states of that condition from blue to orange. Click **Upload to DataHigh** to upload the changes.

Example 1: Visualize neural states

data file: 'ex1_neuralstates.mat'

description: On each trial, spike counts were taken in a single 400ms bin during the delay period in a standard delayed reaching task. We applied factor analysis (FA) to all trials together to reduce the 61-d count vectors (61 simultaneously-recorded neurons) to 6-d. We use DataHigh to visualize the 6-d space. Each color corresponds to one of 7 reach directions.

Santhanam G, Yu BM, Gilja V, Afshar A, Ryu SI, Sahani M, Shenoy KV (2009) Factor-analysis methods for higher-performance neural prostheses. Journal of Neurophysiology. 102:614-635.

Detailed Tutorial:

This tutorial walks through Example 3.1 in the JNE paper, and you can follow an instructional video on the DataHigh website. These steps continue from Example 1: Extract neural states from spike count vectors (see above).

1. If you are continuing the steps for Example 1, the DataHigh interface should fill the screen with neural states, and you can skip to the next step. If you jumped to this visualization tutorial, navigate to the **examples** folder in the main DataHigh directory. Enter **ex1_visualize** into the Matlab command line to upload the neural state data to DataHigh. We applied factor analysis to the 61-dimensional spike count vectors and determined the dimensionality to be six. The dataset includes seven experimental conditions, and each condition has 30 trials (i.e., 30 neural states).
2. The main DataHigh interface fills the screen. The central panel shows a 2-d projection of the 6-d neural states. Each color corresponds to one of seven experimental conditions. Each point corresponds to one experimental trial. Each condition has multiple trials, and the population activity is separable based on condition (the clusters have little overlap).
3. Save the current projection in a queue so that we can access it later. Click **Saved Projections** in the Toolbox, and then click **Capture Projection**.
4. We can also add descriptive annotations to the data for better visualization. Click **Annotations** in the Toolbox, and click **Cov Ellipses**. This plots an ellipse that describes the covariance matrix for each condition.
5. We are also interested in visualizing in what direction the trial-to-trial variability points. Click **1st Prin. Comp.** directly below the Cov Ellipses button to plot the first principal component's direction in the 2-d projection. They appear as thick lines that intersect the clusters' means.
6. This current projection shows separable clusters with little trial-to-trial variability. However, this a 6-d space---could a large amount of trial-to-trial variability exist in other dimensions of the space? Use the preview panels (located to the left and right of the central panel) to rotate the current 2-d projection plane in the 6-d space. The first principal component vectors increase in length. This means that most of the trial-to-trial variability exists outside of the 2-d projection we began with.
7. Click **Randomize** under the central panel a few times, which randomizes the orientation of the 2-d projection plane within the 6-d space. Most of the projections will look like a "bowl of Fruitloops," without separable clusters. Use the preview panels to find a projection in which some clusters do not overlap with others.
8. There are pre-defined cost-functions that can be used to also find projections in which the clusters are separable. Click **Find Projection** under the **Analysis Tools** tab in the Toolbox. Click the **PCA** button to view a projection defined by the top two directions of greatest scatter found

by principal component analysis. Click **LDA** to see a projection found by Fisher's linear discriminant analysis, in which the cluster sizes are small, and the distances between clusters are large. Click **Upload to DataHigh**.

9. Depth Perception is another tool that can be useful to find separability between clusters and identify outlying trials. First, click **Conditions** in the Toolbox. Then deselect all conditions except for "reach1" and "reach5." Use the preview panels to rotate the main 2-d projection until the two clusters overlap. Then, click the Annotations tab, and click **DepthPercept**. One condition's datapoints will become much larger than the other condition's. Depth Perception first finds a direction that is orthogonal to the 2-d projection (e.g., pointing out of the monitor screen). Then, it sizes datapoints based on how much they "come out" of the monitor screen towards the user. Datapoints that differ greatly in size also differ in distance in the high-dimensional space.

Example 2: Extract single-trial neural trajectories from raw spike trains

data file: 'ex2_rawspiketrains.mat'
description: This is an example dataset to get familiar with dimensionality reduction. The dataset contains spike trains recorded simultaneously from 61 neurons in the premotor cortex during a standard delayed reaching task. There are 56 reaches to each of two reach targets. The user is instructed to perform dimensionality reduction and automatically upload the extracted neural trajectories to DataHigh.

B. M. Yu, J. P. Cunningham, G. Santhanam, S. I. Ryu, K. V. Shenoy, and M. Sahani. "Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity." *J. Neurophysiol*, vol. 102, 2009, pp. 614-635.

Detailed Tutorial:
The same tutorial and an instructional video can be found on the DataHigh website.

1. Navigate to the examples folder in the main DataHigh directory. Enter **ex2_dimreduce** into the Matlab command console to input raw spike trains into DimReduce. The example dataset contains two different experimental conditions. Each condition has 56 trials, and each trial has spike trains from 61 neurons. Each spike train has a length between 1018ms and 1526ms and binned at 1ms resolution.
2. The DimReduce figure will pop up. The top right corner has Next Step instructions, so that the user can walk through the steps of performing dimensionality reduction. Note that clicking **Next Step** will move the large red number to the next location where a choice is needed by the user. Each option also has a **?** button nearby to provide more information about that step.
3. For this example dataset, in the **time bin width** box, enter 20ms (if not there by default). The "Type" will read "Trajs" for neural trajectories. DimReduce will take 20ms non-overlapping bins over the raw spike trains. We recommend that you start with 20ms bins, as we have found this to work well with population activity recorded in the motor and visual cortices.
4. For the **mean spikes/sec threshold**, enter 1.0 spike/sec. DataHigh removes 3 neurons from the analyses. Neurons with firing rates less than 1 spike/sec can sometimes be problematic, so so we leave them out. Leave the "Trial-averaged neural trajs" box unchecked.

5. Under **Method**, choose "GPFA" for Gaussian Process Factor Analysis. GPFA reduces the high-dimensional population activity to a smaller number of latent variables that vary smoothly over time, where the amount of smoothing is determined by the data.
6. GPFA typically takes less than a minute to fit to data from tens of neurons and hundreds of trials. However, when we do cross-validation, we may need to fit GPFA hundreds of times (depending on the number of candidate dimensionalities and cross-validation folds). For the purposes of visualization, we can avoid the long wait by fitting a GPFA model with a large number of dimensions, looking at the parameters returned by the fitting procedure, and removing the dimensions that appear to not be needed. Slide the **Select dimensionality** scrollbar to 40. Click **Upload Results**, which performs dimensionality reduction for the selected dimensionality. A progress bar pops up. It should take about 30 seconds to fit the parameters and extract the neural trajectories.
7. After the progress bar finishes, the PostDimreduce figure then pops up. We will now decide how many of the 40 dimensions are actually needed to describe the data.
8. Click **View Loading Matrix**. The loading matrix defines the linear mapping between latent variables (columns) and the neural activity (rows). Check to see whether any of the columns of the loading matrix show contributions from only a small number (e.g., 2 or 3) of neurons. This indicates that the spike counts for those neurons are highly correlated, which could be an indication of electrode cross-talk. Under normal conditions, we expect each column to have contributions from many of the neurons, which is the case here. Close the figure.
9. Click the **View Each Dim** button. DimReduce automatically orthonormalized the latent variables, and ordered them based on the amount of variability they explain. In the SingleDim figure, you can see each latent variable plotted individually. Each trace corresponds to a single trial. Click the **Next** button to scan through the latent variables. As you scan, you will notice that past a certain latent variable, the neural trajectories do not vary across time or experimental condition. These latent variables are not contributing much to describing the data, and will be removed for visualization. Close the SingleDim figure.
10. Click the **Eigenspectrum** button. This plots the cumulative percent shared variance explained by the number of latent variables. A good selection of the dimensionality would explain around 90% of the shared variance and have temporal fluctuations for each single dimension. However, the selection can be subjective, unlike the optimal dimensionality identified with cross-validation. Close the Eigenspectrum figure.
11. In the **Select final dimensionality** menu list, select a dimensionality of 15. DimReduce will remove latent variables 16 through 40 from the analyses, and keep the top fifteen latent variables. Thus, we reduced the original 58-d space to 15 dimensions. Click **Upload to DataHigh**. The DataHigh interface fills the screen.

Example 2: Visualize single-trial neural trajectories

data file: 'ex2_singletrialtrajs.mat'

description: Spike counts were taken in non-overlapping 20ms bins during a standard delayed reaching task. We applied Gaussian-process factor analysis (GPFA) to reduce the 61-d count vectors (61 simultaneously-recorded neurons) to 15-d. We use DataHigh to visualize the 15-d space. Trajectories include the time before the reach target is presented, reach planning, and reach execution. Each color corresponds to one of two reach conditions.

B. M. Yu, J. P. Cunningham, G. Santhanam, S. I. Ryu, K. V. Shenoy, and M. Sahani. "Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity." *J. Neurophysiol*, vol. 102, 2009, pp. 614-635.

Detailed Tutorial:

This tutorial walks through Example 3.2 in the JNE paper, and you can follow an instructional video on the DataHigh website. These steps continue from Example 2: Extract single-trial neural trajectories from raw spike trains (see above).

1. If you are continuing the steps from the Example 2 DimReduce tutorial, we could visualize all of the extracted neural trajectories, which may look cluttered. For visual clarity, we suggest initially visualizing a small subset of trajectories to orient ourselves in the latent space. For your convenience, we have included example data in which the neural trajectories were extracted in the same manner as the previous tutorial, but only 15 trajectories were kept for each condition. To load the data, click the **Analysis Tools** tab and then click **Load Data**. Navigate back one folder to the main **DataHigh** directory, then navigate to the **data** folder. Load **ex2_singletrialtrajs.mat**. Continue to the next step.

   Otherwise, if you have jumped to this tutorial, navigate to the examples folder in the main DataHigh directory. Enter **ex2_visualize** into the Matlab command line console to upload the single-trial neural trajectory data to DataHigh. Beforehand, we applied Gaussian-process factor analysis (GPFA) to the 61-dimensional raw spike trains and determined the optimal dimensionality to be 15 with cross-validation. The data consist of two experimental conditions, and each condition has 15 trials.
2. The main DataHigh interface fills the screen. The colors (green and blue) represent the two different experimental conditions. Each neural trajectory represents the population timecourse for one trial. The start of the neural trajectories (i.e., the first timepoints) are in light grey in the center of the projection, and time increases as you follow along the trajectories. Save this projection by clicking the **Saved Projections** tab and then **Capture Projection**.
3. Now click **Randomize**. Try to use the preview panels (located to the left and right of the central panel) to find a nice projection that separates the neural trajectories based on their corresponding experimental conditions (green and blue). If you feel you are stuck at a certain type of projection (e.g., a local optima of visualization), you can start over with another random projection. In truth, you are not searching for the "best projection," but rather viewing many projections that give you intuition for the data. In the process of trying to find a nice projection, you can get a sense of how ugly or beautiful the data are. You can also click the **Freeroll** button to see a smooth sequence of projections---it runs like a screensaver.
4. There are other tools if you are having trouble finding a nice projection (sometimes the data will not have a "nice" projection!). You could use the Find Projection tool, but the standard cost

functions tend to not find meaningful projections. Another option is Genetic Search. Click **Analysis Tools** and click the **Genetic Search** button. The Genetic Search figure starts with fifteen random projections. Click on projections that look of interest. Then, click on **Next Generation**. DataHigh has an algorithm that finds fifteen new projections that are similar to the selected projections. Try some iterations of GeneticSearch. You can upload an interesting projection by clicking on it (only one projection should be selected) and clicking **Upload to DataHigh**.

5. We can do more analysis on the previously found projection. Upload the captured projection to the central panel by clicking the **Saved Projections** tab and clicking on the previously captured projection's thumbnail. Now click the **Analysis Tools** tab and click **3d Projection**. The Projection3D figure appears. The user can use this projection as a Matlab 3-d viewer (as if we used plot3) by dragging the figure around.
6. Note that this is a static 3-d projection (with the same two projection vectors used by the central panel and a third random projection vector). Clicking **Randomize** may alter the 3-d projection, as it randomizes the third projection vector. Click on the **Evolve** button to view the population activity play out over time. You may drag on the figure while Evolve is executing, and also save a movie. After Evolve has finished, close the Projection3D figure.
7. Click the **Annotations** tab and then click on **Average Trajs**. This displays average neural trajectories for each condition, which can be helpful in analysis. You can also view the epoch boundaries by clicking on the **Epoch Boundaries** button. For this example dataset, the two epoch boundaries correspond to target onset and to when the go cue was given. The starting epoch in grey corresponds to pre-stimulus activity, and the last timepoint corresponds to movement onset. You can quickly change epoch colors with the Update Colors tool in Analysis Tools.
8. DataHigh is also useful to detect "outlying" trials (i.e., a trial in which the population activity does not resemble that of other trials from the same condition). This can be useful for quickly triaging large data sets, and identifying which trials are out of the ordinary, which may be interesting in their own right. For example, an outlying trajectory might correspond to a trial with a long reaction time. We previously found an outlying neural trajectory with DataHigh. Click the **Analysis Tools** tab, and click **Update Colors**. In the top left of the UpdateColors figure, click the **Update Trial** button. Under **Select Trial**, select trial 7. Under Epoch Colors, change rows 2 and 3 from [0 .7 0] to [1 0 0]. This changes the trajectory's color to red. Click **Upload to DataHigh**. Click the **Conditions** tab and deselect condition 2. Click **Recenter Data** at the very bottom of the Conditions panel. You can now rotate the 2-d projection plane to see how the red neural trajectory (trial 7) deviates from the others. You can also find differences using the Genetic Search tool.

Example 3: Extract trial-averaged neural trajectories from PSTHs

data file: 'ex3_psths.mat'
description: This is an example dataset to get familiar with dimensionality reduction. The dataset contains peri-stimulus time histograms (PSTHs) recorded from 61 neurons in the

premotor cortex during a standard delayed reaching task. The PSTHs are averaged over trials corresponding to one of two reach targets. The user is instructed to perform dimensionality reduction and automatically upload the extracted neural trajectories to DataHigh.

B. M. Yu, J. P. Cunningham, G. Santhanam, S. I. Ryu, K. V. Shenoy, and M. Sahani. "Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity." *J. Neurophysiol*, vol. 102, 2009, pp. 614-635.

Detailed Tutorial:

The same tutorial and an instructional video can be found on the DataHigh website.

1. Navigate to the examples folder in the main DataHigh directory. Enter **ex3_dimreduce** into the Matlab command console to input PSTHs into DimReduce. The example dataset contains two different experimental conditions whose PSTHs have already been computed. Each PSTH has 61 neurons and is binned with 1ms resolution. The same computation can be achieved by inputting raw spike trains into DimReduce and selecting the "Trial-averaged neural trajectories" check box. If you decide to input raw spike trains, make sure the spike trains have first been properly aligned.
2. The DimReduce figure will pop up. The top right corner has Next Step instructions, so that the user can walk through the steps of performing dimensionality reduction. Note that clicking **Next Step** will move the large red number to the next location where a choice is needed by the user. Each option also has a **?** button nearby to provide more information about that step.
3. For this example dataset, in the **time bin width** box, enter 20ms (if not there by default). The "Type" will read "Trajs" for neural trajectories. DimReduce will convert the PSTHs to 20ms resolution by locally summing within non-overlapping 20ms bins.
4. For the **mean spikes/sec threshold**, enter 1.0 spikes/sec. DataHigh removes 4 neurons from the analyses. Neurons with firing rates less than 1 spikes/sec can sometimes be problematic, so we remove them. Leave the "Trial-averaged neural trajs" box unchecked, since we already trial-averaged the data.
5. Under **Method**, choose "PCA" for Principal Component Analysis. By averaging across trials, we presumably remove much of the Poisson-like spiking variability. Thus, we can apply PCA, which has no concept of observation noise, to the PSTHs. Leave the "Trial-averaged neural trajs" and "Square Root Transform" boxes unchecked, and leave the smoothing kernel width at 0ms.
6. We can estimate the "optimal" dimensionality using PCA by viewing the eigenspectrum. Slide the **Select dimensionality** scrollbar on the right side of DimReduce to the maximum dimensionality (57). Click **Perform Dim Reduction** to perform dimensionality reduction. The PostDimReduce figure pops up.
7. Click the **View Each Dim** button. The SingleDim figure pops up. It displays the individual orthonormalized latent variables for all conditions. Orthonormalization has the advantage of ordering the latent variables such that latent variable 1 describes the most variance in the data while latent variable 57 describes the least variance. Click **Next** to scan through the latent variables. One noticeable feature is that the traces in time are not smooth. This is probably due to a limited number of trials to estimate the PSTHs.
8. We can go back and smooth the PSTHs before applying dimensionality reduction. Close the SingleDim and PostDimReduce figures. On the left side of DimReduce, slide the **Smoothing kernel width** to 25ms, which convolves each PSTH with a Gaussian with standard deviation of

25ms. Then click on **Perform dim reduction** again to perform dimensionality reduction. The PostDimReduce figure pops up.

9. Again, click on the **View Each Dim** button, and scan through the latent variables. The neural trajectories are now smooth. After the top ten, the latent variables do not vary much across time or experimental condition, so we can probably safely disregard them. Close the SingleDim figure.
10. Click the **Eigenspectrum** button, which pops up the Eigenspectrum figure. We see that the top six dimensions explain about 93% of the variance, which agrees with what we found in the individual latent variables with SingleDim. We could select the top ten latent variables, but it appears that the last four do not help much to describe the data. Close the Eigenspectrum figure.
11. In the **Select final dimensionality** menu list, select a dimensionality of 6. DimReduce removes latent variables 7 through 57, and keeps the top six latent variables. Click **Upload to DataHigh**. The DataHigh interface fills the screen.
12. Since there are so few trajectories, the rotation speed may be fast. You can test this by clicking and holding on one of the preview panels (to the left and right of the central panel), which rotates the central panel's 2-d projection. To change the rotation speed, click the **Analysis Tools** tab and then the **Zoom/Rotate** button to pop up the Zoom/Rotate figure. Slide the middle scrollbar (named Rotation Speed) down to a slower value. Close the Zoom/Rotate figure. The rotation speed has now been lowered, and can be tested by clicking and holding on a preview panel.

Example 3: Visualize trial-averaged neural trajectories

data file: 'ex3_trialavgtrajs.mat'
description: Monkeys made curved reaches around barriers. This neural data is from the movement epoch. For each of the 27 reach conditions, we first created PSTHs for each of the 118 sequentially-recorded neurons. This is the trial-averaged population response for that reach condition. Pre-processing included PC-smoothing, subtraction of across-condition mean, and jPCA. Each trajectory corresponds to one reach condition.

M.M. Churchland, J.P. Cunningham, M.T. Kaufman, J.D. Foster, P. Nuyujukian, S.I. Ryu, and K.V. Shenoy (2012). "Neural population dynamics during reaching," Nature 487(7405), 51-6.

Detailed Tutorial:
This tutorial walks through Example 3.3 in the JNE paper, and you can follow an instructional video on the DataHigh website. These steps continue from Example 3: Extract trial-averaged neural trajectories from PSTHs (see above).

1. If you are continuing the steps from the Example 3 DimReduce tutorial, we will load a more interesting dataset, so that we can look at trial-averaged neural trajectories from many experimental conditions, rather than just 2 conditions. To load the data, click the **Analysis Tools** tab, and then click **Load Data**. Navigate back one folder to the main DataHigh directory, then navigate to the data folder. Load **ex3_trialavgtrajs.mat**. Continue to the next step.

Otherwise, if you have jumped to this tutorial, navigate to the **examples** folder in the main DataHigh directory. Enter **ex3_visualize** into the command line console to upload the trial-averaged neural trajectory data to DataHigh.

2. The example dataset comes from the Churchland et al. 2012 paper, in which a subject made reaches to different targets. The PSTHs were computed for 118 neurons and 27 experimental conditions. We performed dimensionality reduction on the PSTHs to reduce the 118-dimensional space to 6-d, using a dimensionality reduction method called jPCA. jPCA extracts latent variables that reveal any rotational structure in the data.
3. Each blue trace is a trial-averaged neural trajectory of one reach condition. As the preview panels show, even a 6-d space can have a lot of variety! Scale the projection so that the neural trajectories are not clipped off in the central panel. Click the **Analysis Tools** tab, and then click **Zoom/Rotate**. Slide the **Zoom** scrollbar to "out," which immediately shrinks the projection. Close the ZoomRotate figure.
4. Save the current projection for later viewing by clicking on the **Saved Projections** tab and then **Capture Projection**.
5. Rotate around in the 6-d space with the preview panels (to the left and right of the central panel). You will find that the rotational structure is a salient feature in the data, but other interesting features do exist. For example, the trajectories seem to be in pairs. Each pair contains two experimental conditions that had the same reach endpoint but differed in the path of the reach.
6. Reload the original projection by clicking the **Saved Projections** tab and then the thumbnail of the original projection. Click the **Analysis Tools** tab and then click **3d Projection**. You can see the rotational structure in the 3-d projection. Click **Evolve** to see the timecourse evolve. Close the Projection3d figure.
7. Under **Analysis Tools**, click **SingleDim**. Each latent variable is plotted versus time. From these individual plots, it is not obvious that the trial-averaged neural trajectories would contain rotational structure. This is an example of how looking at the population activity in a multi-dimenisonal firing rate space can yield insights that would be difficult to obtain by looking at the PSTHs of individual neurons (or the timecourse of individual latent variables).

==========

What to do if the trajectories look like spaghetti

==========

In an arbitrary 2-d projection, the neural trajectories can appear to overlap and look tangled like "spaghetti." The following are steps to take to try to clarify what is being shown in the 2-d projections.

1. The user can repeatedly click the "Randomize" button in the main DataHigh interface to visualize many different initial random projections, which can then be rotated by clicking and holding the preview panels.
2. One can also view many random 2-d projections using the Freeroll, Find Projection, and Genetic Search tools. After an interesting 2-d projection is found, the user can upload it to the main DataHigh interface and rotate the projection using the preview panels.
3. Instead of viewing entire neural trajectories which can look cluttered, the Evolve tool can be used to incrementally plot segments of the neural trajectories.
4. The user may also use SingleDim to individually inspect each latent variable. Neural trajectories that look jumbled in a 2-d projection can look well-organized when viewing each latent variable versus time.
5. For single-trial neural trajectories, one can visualize the deviation of single trials from average trajectories by clicking on the “Average Trajs” annotation option in the DataHigh toolbox.
6. Another technique is to label each trial as a different condition, begin by displaying a single neural trajectory, and progressively add more trials using the “Conditions” tab, identifying subtle differences among neural trajectories.
7. Another possible approach when there are many conditions is to average each neural trajectory across time to obtain one point for each trajectory (i.e., the ``center of mass" for each trajectory). The user can apply PCA, LDA, or cluster PCA with Find Projection to find a 2-d projection that separates the points. Then, one can use the same 2-d projection to view the neural trajectories.

==========
Analysis Tools
==========

Conditions	Select which conditions will be displayed
Pop Figure	Creates a Matlab fig that can be saved
3d Projection	Randomly chooses a 3rd proj vec to show a 3-d projection
Evolve	Timecourse of each trajectory is played out
Find Projection	Find projection given by PCA, PCA of cluster means, or LDA
Projection Weights	View the current projection vectors’ coefficients
Smoother	Applies a Gaussian kernel to smooth the trajectories
Drag Trajectory	Change a trajectory by changing each dimension’s time-varying values
Zoom/Rotate	Change zoom, rotation speed, and in-plane rotation of 2d projection
Genetic Search	Uses a genetic algorithm to find projections with similar features
DimReduce	Perform dimensionality reduction on the input data
SingleDim	Plot each latent variable individually
Load Proj Vecs	Load any previously saved projection vectors from a .mat file
Save Proj Vecs	Save the current projection vectors into a .mat file

Load Data	Open a new dataset; assumed to have struct D
Update Colors	Modify the existing struct to change D.condition and D.epochColors
Saved Projections	Capture the current projection to return to at a later time
Average Trajectories	Computes an average trajectory for the single-trial trajectories in each condition
Epoch Boundaries	Display cues denoting the elements in epochStarts
Cov Ellipses	Display 2d covariance ellipses of the high-d data
1st Prin. Comp.	Display the first principal component, projected from the high-d data
Origin	Display dot at the origin in the latent space
Cluster Means	Display a dot for each condition's cluster mean
Depth Perception	Changes sizes of dots based on their distance between each other in the orthogonal space of the current 2-d projection
Freeroll	Acts like a screensaver, randomly rotating the projection vectors

==========
Data format for DimReduce
==========

Load `'./data/RawSpikeTrains.mat'` for an example. DimReduce accepts a struct array, D, where each element contains the spike trains for one trial. Spike trains should be binned in 1 ms timesteps.

D =
1xN struct array with fields:
data: (number of neurons x number of milliseconds) array

The data can include optional fields (type, epochStarts, epochColors, condition) as outlined in the next section, Data format for DataHigh.

==========
Data format for DataHigh
==========

See the example datasets in `'./data/'`. DataHigh accepts a struct array, D, where each element represents a cluster of neural states or a neural trajectory.

D =

1xN struct array with fields:
type: 'traj' or 'state'
epochStarts: (1 x number of epochs)
epochColors: (number of epochs x RGB)
condition: 'conditionName'
data: (number of dimensions x number of data points)

Example:

D(1):
type: 'traj'
epochStarts: [1 36 78]
epochColors: [0.7 0.7 0.7; 1 0 0; 0 1 0]
condition: 'left_reach'
data: [7x145 array]

==========
Noted problems
==========

1. In the past, we have had performance issues with Mac computers. However, we have sped up the code such that we do not see such issues currently. If the continuous rotation of the central panel seems slow, use the "Zoom/Rotate" feature to increase the rotation speed.
2. Font size on the figures has always been an issue. Even though Matlab is platform independent, the Matlab GUI software certainly is not. I have worked on both Windows and Mac laptops to optimize the font size for all figures. However, different Windows/Mac configurations may cause the font to either be too big or too small. We plan to address these issues in our next code release (and allow the user to decide the font size). However, for now, please resize the DataHigh figure to better fit your screen, and use screenshots (either from the instructional videos on the website or the DataHigh JNE Paper) to determine the text, and let us know of the problem. In general, DataHigh font size should be fine for most users (it has been tested on the 15-inch MacBook Pro and 15-inch Dell XPS 15z).

==========
FAQ

==========

Q: What is a neural trajectory?
A: A neural trajectory describes the time-evolution of neural population activity. The basic setup is a high-dimensional space, where each axis represents the firing rate of one neuron in the population. As the neural activity unfolds over time, a trajectory is traced out in the space. For more information, see Yu et al., *J Neurophysiol*, 2009.

Q: If I "get lost" in the space, can I get back to the original projection?
A: If you used the Saved Projections tool, you can easily return to a previous projection. Otherwise, if you have saved the desired original projection, go to [Toolbox→Analysis Tools→Load Proj Vecs] to load the .mat file.

Q: What do the panels actually do?
A: Each panel shows a preview of what would happen if you rotated the current projection vectors in a particular direction by 180 degrees. By holding on a panel, you are slowly incrementing the angle of plane of rotation, which continuously updates the main projection.

Q: I found a bug.
A: Please contact us at DataHigh@gmail.com. We'll apply the fly-swatter!

Q: How do I change an outlying trajectory's color?
A: Go to [Toolbox→Update Colors→Update Trial] and change the appropriate epochColors field, or change it in the data struct passed into DataHigh.

Q: Why is DataHigh useful for clusters of datapoints (spike counts)? Can't I just use LDA to find the projection of most separation?
A: It would be pointless to spend hours with DataHigh looking for a projection that separates the clusters well, when statistical techniques, such as Fisher's linear discriminant analysis (LDA), does the job quite well. However, LDA only gives one projection…what is happening in the infinite number of other projections? DataHigh can be used to analyze the shape of each cluster and the trial-to-trial variability of the data. See Example 2 for more intuition.

Q: What is % var, and how is it calculated?
A: % var displays how much variance is in the current projection, compared to the high-d data. Mathematically, we take the trace of the current projection's covariance matrix and divide it by the trace of the latent data's covariance matrix. The highest possible % var is achieved when the projection vectors are the first two principal components.

Q: I really need this awesome feature for DataHigh, but it's not there. What can I do?
A: Hack the code! Feel free to manipulate any part of the code as you see fit. However, please adhere to the open source copyright license, which states that no original source code may be deleted (you can comment it out, though) if you plan to redistribute the software package. If

you feel this would be a feature that many would also find useful, shoot us an e-mail, and we'd be happy to consider incorporating the feature in the next code release.

Q: How can I call DataHigh from another directory?
A: You need to add DataHigh's directory to the current path.
>> help addpath

Q: Where can I order my DataHigh t-shirt?
A: DataHigh tees will be available for purchase at all major commercial retailers soon.

Q: How is the average trajectory computed?
A: For visual purposes, we would like to have a reference trajectory that represents an average of the single-trial neural trajectories for different conditions. Note that this is different from, but qualitatively similar to, averaging before dimensionality reduction to produce trial-averaged neural trajectories. We would like this average trajectory to obey epoch boundaries (i.e., at each epoch boundary, we should take an average) and be continuous. We opted for a method that linearly resamples each neural trajectory based on the distance traveled by the trajectory. Thus, for an epoch, if two trajectories travel roughly the same distance but have different time durations, the average only considers the distances.

In the figure, two hypothetical single-trial neural trajectories (green) travel from an initial state of the epoch (bottom left) to a final state (top right). The leftmost green trajectory reaches the final state and stops, since its trial length is short. The rightmost green trajectory reaches the final state and hovers there, waiting for the epoch to end (which could happen when a task cue is given, etc.). Our method focuses on averaging the main structure of the trajectories (traveling from initial to final state) rather than including the small loops of waiting for the task cue. The average trajectory (red) moves to the final state in a similar fashion as both green trajectories, since our method downweights the waiting activity where the time was long but the distance traveled was small. While this method is effective for visualization, the user should take care whenever averaging trials that do not have the same epoch length across trials.