Table of Contents

Create the analysis object

This analysis uses the 'AdaptiveDmdc' class, i.e. adaptive Dynamic Mode Decomposition with Control, and the steps of this algorithm will be explained in this document First, we want to set the options for our analysis. This is done using a struct, but all the defaults should run as is. For now we will turn off all plotting, to be explained one-by-one

```
settings = struct('to_plot_nothing',true);
```

Next we need to set the filename to be analyzed. Clearly this will depend on your folder organization.

```
filename = '../../Collaborations/Zimmer_data/WildType_adult/simplewt5/
wbdataset.mat';
```

I'm not exactly sure of the format yet, so for now let's get the data that we actually want to analyze out manually

```
Zimmer_struct = importdata(filename);
dat = Zimmer_struct.traces.';
```

We can also add the neuron names to the settings struct, so that the algorithm can use them for some plots.

```
id_struct = struct(...
    'ID', {Zimmer_struct.ID},...
    'ID2', {Zimmer_struct.ID2},...
    'ID3', {Zimmer_struct.ID3});
settings.id_struct = id_struct;
```

Finally, we create the analysis object.

```
ad_obj = AdaptiveDmdc(dat, settings);
```

```
Preprocessing...
Finished analyzing
```

This object has various properties that are explained by typing either:

```
>> help AdaptiveDmdc
```

```
>> help ad_obj
```

Note that these display a lot of text!

The algorithm

The algorithm will be explained by going through the documentation function by function, starting with the initializer

```
help AdaptiveDmdc.AdaptiveDmdc
```

```
  Creates adaptive_dmdc object using the filename or data
  matrix (neurons=rows, time=columns) in file_or_dat.
  The settings struct can have many fields, as explained in the
  full help command.

  This initializer runs the following functions:
    import_settings_to_self(settings);
    preprocess();
    calc_data_outliers();
    calc_outlier_indices();
    calc_dmd_and_errors();
    plot_using_settings();

  And optionally:
    augment_and_redo();
```

Importing the settings is pretty clear. Preprocessing has several options, all set in the above 'settings' struct, e.g. subtracting the mean. The first algorithmic piece is the function "calc_data_outliers":

```
help AdaptiveDmdc.calc_data_outliers
```

```
  Calculate which neurons have nonlinear features and should be
  separated out as controllers by various methods:

    sparsePCA: take out the nodes that have high loading on the
        first 15 pca modes

  The following methods all solve the DMD problem (x2=A*x1),
  subtract off the resulting linear fit, and then calculate
  features based on the residuals
    DMD_error: choose neurons with the highest L2 error
    DMD_error_exp: choose neurons with the highest error
        calculated using an exponential (length scale,
        'lambda', set in the initial options)
    DMD_error_normalized: choose neurons with high L2 error,
        normalized by their median activity
    DMD_error_outliers: (default) choose neurons with high
        'outlier error,' which is the L2 norm of only outlier
        points (>=3 std dev away) weighted by 1/distance to
        neighbors (i.e. clusters are weighted more strongly)

    random: randomly selects neurons; for benchmarking
    user_set: The user sets the 'x_indices' setting manually
        (Note: also requires setting 'sort_mode')

  Note: if the setting 'to_plot_cutoff' is true, then this
  function plots the errors of each neuron and the cutoffs used
  as an interactive graph
```

```
help AdaptiveDmdc.calc_outlier_indices
```

```
  Given indices that should be used for the data, calculate and
  save the controller indices as well (the complement of the
  data indices).
    Important: also sorts the data matrix so that the
    controllers are the last rows
```

```
help AdaptiveDmdc.calc_dmd_and_errors
```

```
  Actually performs dmd and calculates errors

  The basic DMD algorithm solves for _A_ in the equation:
    $$ x2 = A*x1 $$
  where the original data matrix, _X_, has been split:
    $$ x2 = X(:,2:end) $$
    $$ x1 = X(:,1:end-1) $$

  DMD with control adds an additional layer, and solves for _A_
  and _B_ in the equation:
    $$ x2 = A*x1 + B*u $$
  where _u_ is the control signal and _B_ is the connection
  between the control signal and the dynamics in the data. In
  this class, the control signal is taken to be certain rows
  of the data as identified by the setting 'sort_mode'

  TODO: implement better and less biased algorithms for DMD
```

```
help AdaptiveDmdc.plot_using_settings
```

```
  Plots several things after analysis is complete

  There are several plotting possibilities, which are all set
        in the original settings struct:
  'to_plot_data': the sorted data set (control signals on
    bottom)
  'to_plot_A_matrix_svd': the svd of the matrix which solves
    the equation $$ x_(t+1) = A*x_t $$ will be plotted
  'which_plot_data_and_filter': plots certain neurons with the
    filter used to determine error outliers
  'to_plot_data_and_outliers': plots individual neuron errors
    with outliers marked (method depends on 'sort_mode');
    interactive

  And printing options:
  'to_print_error': prints L2 error of normal fit vs. fit using
    the control signals
  'use_optdmd': also prints L2 error of an alternative dmd
    algorithm
```

I've been using the last, optional, function as a lead for a way to understand the latent states and transition signals, but it's exploratory more than anything

```
help AdaptiveDmdc.augment_and_redo
```

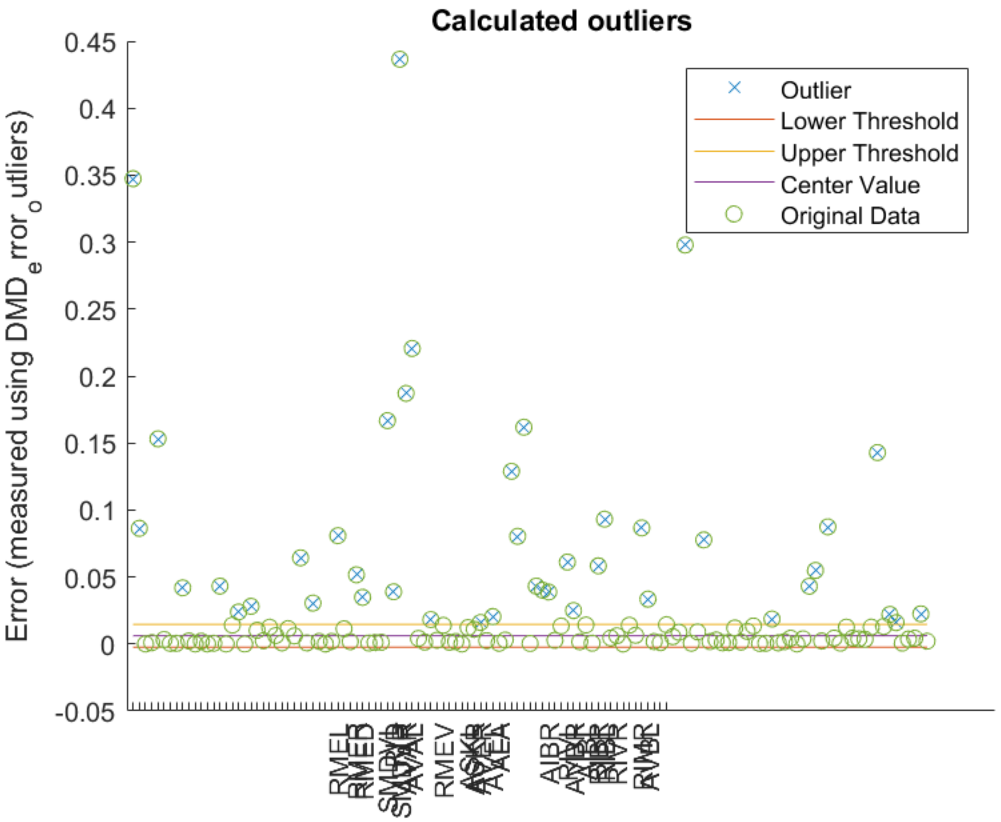
```
  Augments the data with the error signals from the first
  run-through, and then redoes the analysis
```

Plotting

There are several plotting functions that are useful for visualizing what is going on, and one that is interactive. Some sorting methods (e.g. sparsePCA) do not use this type of error detection, but all of the 'DMD_*' methods do. This command by default plots how the signals were used to sort the neurons. Each data point can be clicked to show the residual (original data - DMD fit), with data points marked if they contributed to the counted error.

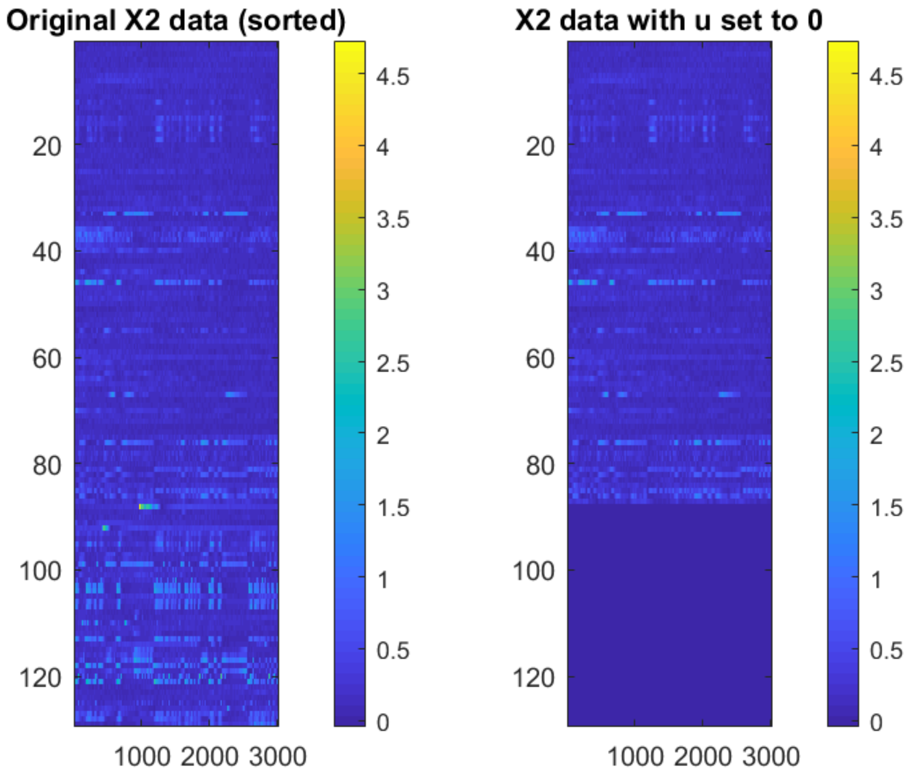
NOTE: has interactivity if plotted

```
ad_obj.plot_data_and_outliers();
```



Another useful visualization is a two-part visualization of the data + control signal the left and the data with the control signal set to 0 on the right

```
ad_obj.plot_data_and_control();
```



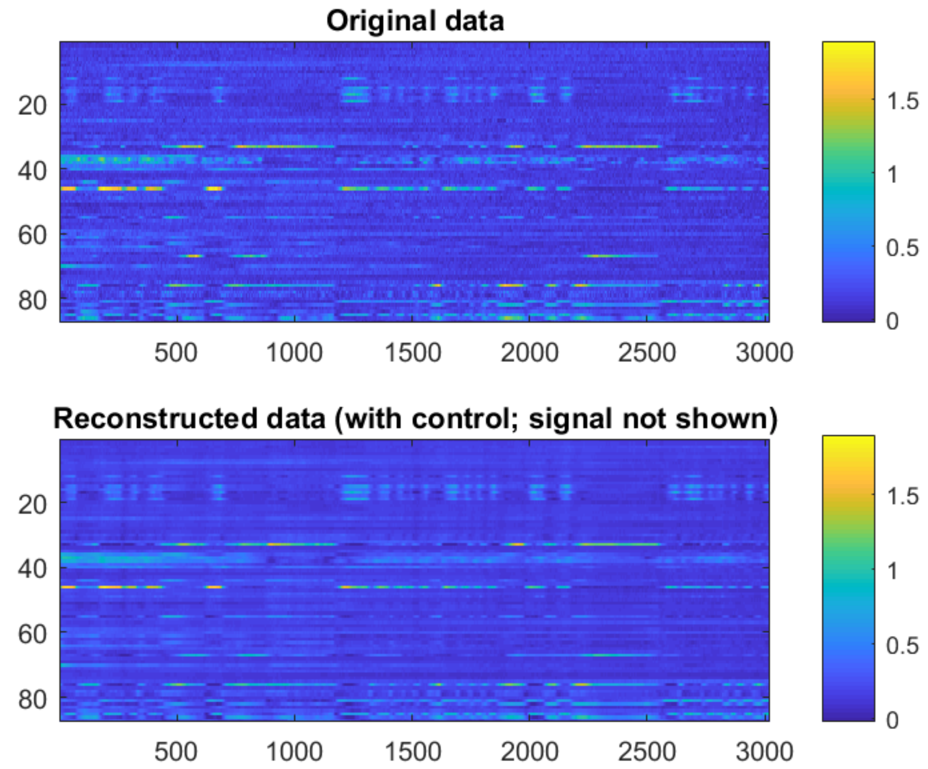
Visualizations related to the key result is are reconstructions, either with control or not. An uncontrolled linear model does not at all capture the dynamics in C elegans, and often leads to NaN or 0 value predictions. Just to reiterate: this algorithm takes values out of the passed data and uses them as predictors (the control signal). This means that the entire dataset is not reconstructed, only part of it.

```
help AdaptiveDmdc.plot_reconstruction
```

```
  Plots a reconstruction of the data using the stored linear
  model. Options (defaults in parentheses):
    use_control (false): reconstruct using control, i.e.
        $$ x_(t+1) = A*x_t + B*u $$
        or without control (the last _B_*_u_ term)
    include_control_signal (false): plot the control signal as
        well as the reconstruction
    to_compare_raw (true): also plot the raw data
    neuron_ind (0): if >0, plots a single neuron instead of the
        entire dataset
```

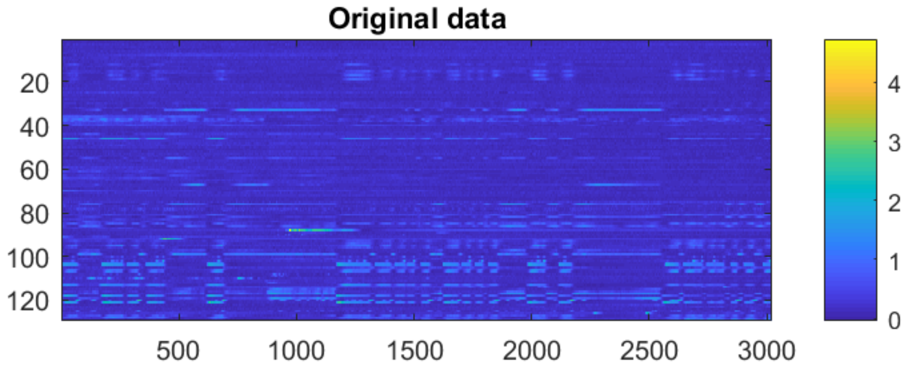
This is using control, and displays only part of the data:

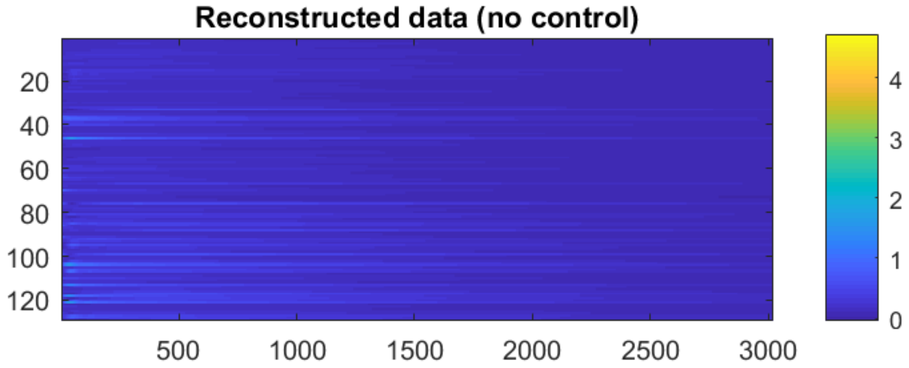
```
ad_obj.plot_reconstruction(true);
```



This plot is without control, and shows all the data

```
ad_obj.plot_reconstruction();
```

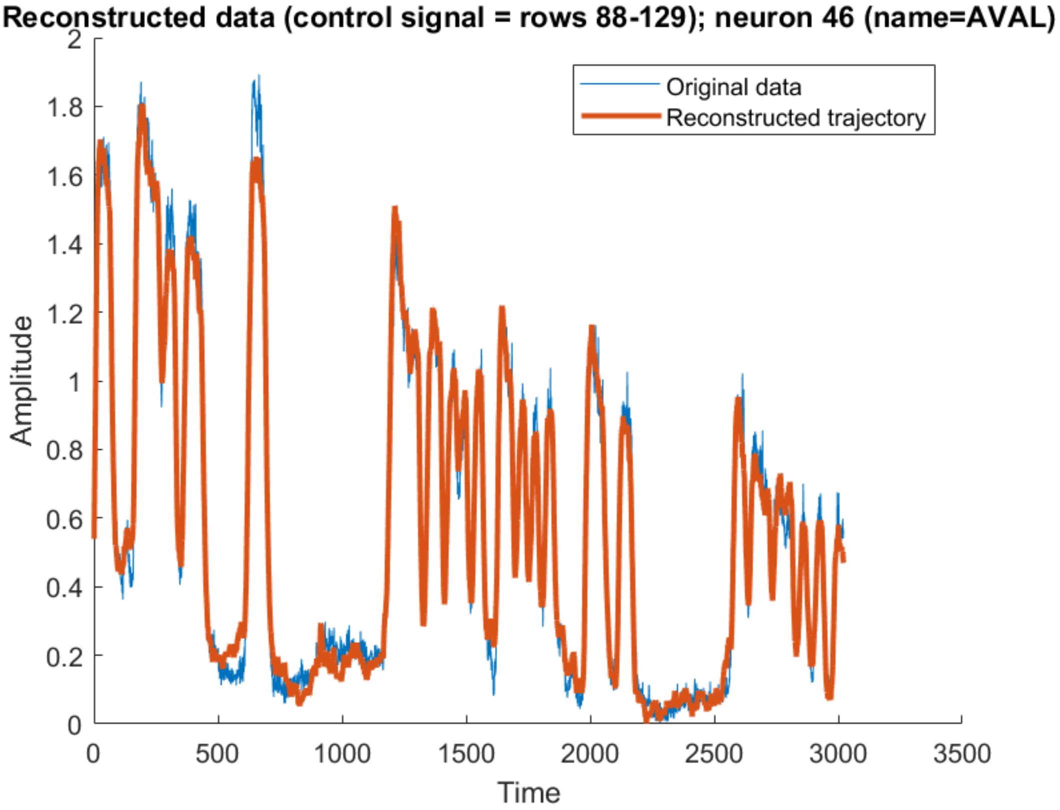

Original data


Reconstructed data (no control)

We can also display individual neuron reconstructions

```
neuron_ind = 46;
use_control = true;
include_control_signal = true;
to_compare_raw = true;
ad_obj.plot_reconstruction(...
    use_control, include_control_signal, to_compare_raw, neuron_ind);
```

```
Identifications of neuron 46: AVAL, , AVAL
```

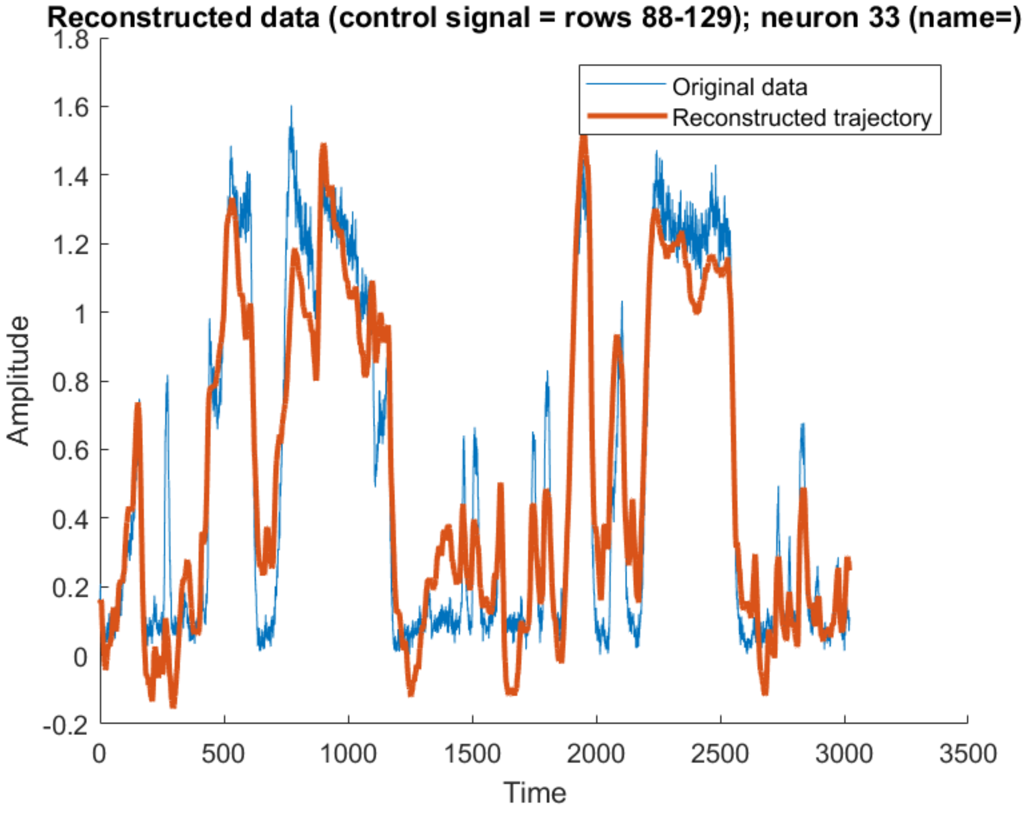


Another neuron with interesting behavior

```
neuron_ind = 33;
ad_obj.plot_reconstruction(...
    use_control, include_control_signal, to_compare_raw, neuron_ind);

%==========================================================================

Identifications of neuron 33: , ,
```



Published with MATLAB® R2017a