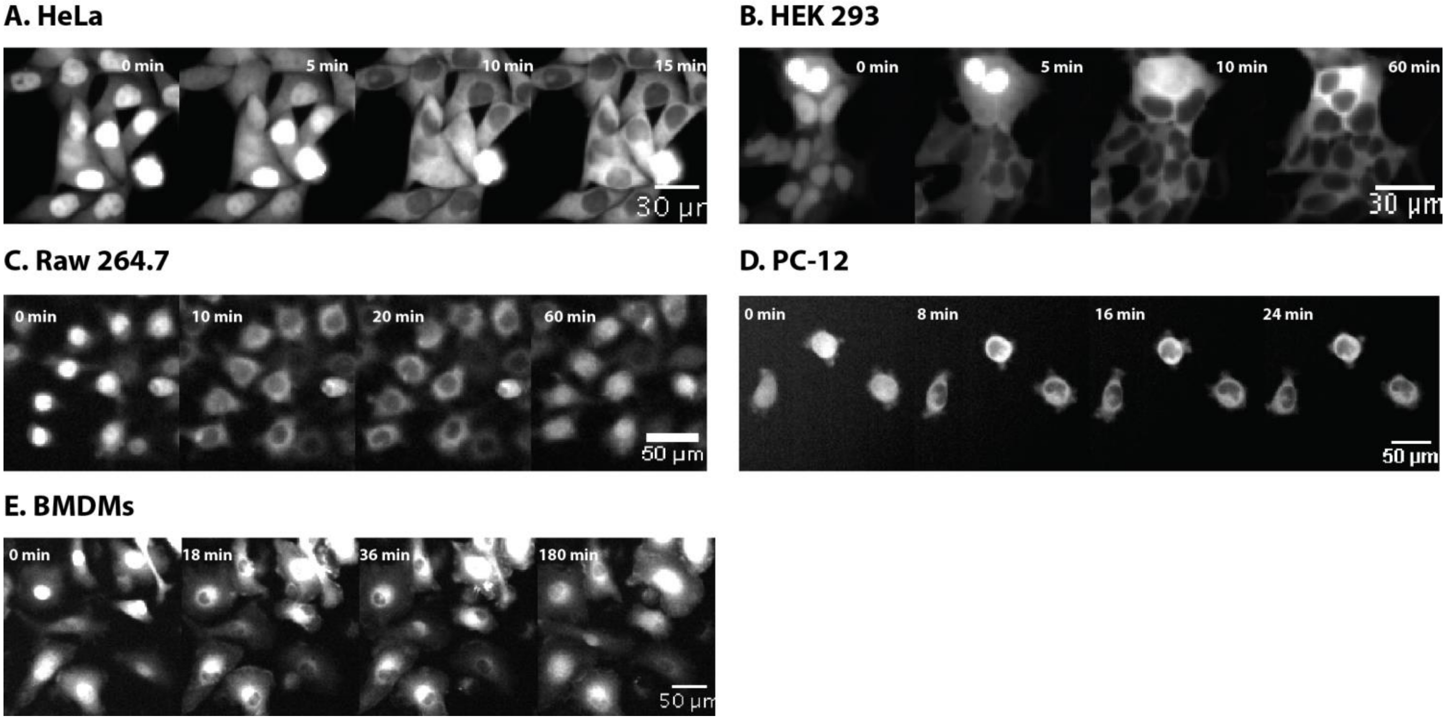
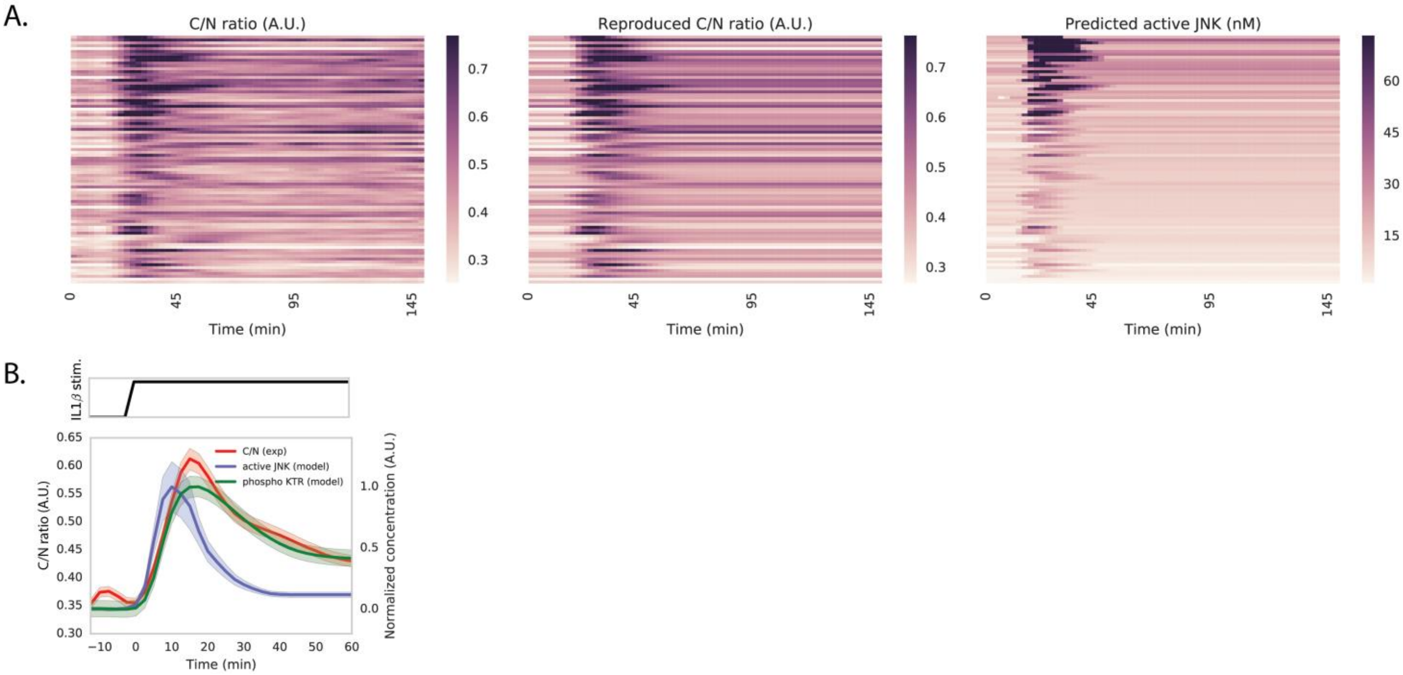


Supplementary Figure 1

KTR translocation in different cell lines.
Time relative to stimulation is shown on each images.
(A) HeLa cells expressing JNK KTR were stimulated with anisomycin 50 ng/ml.
(B) HEK293 cells expressing JNK KTR were stimulated with anisomycin 50 ng/ml.
(C) Raw 264.7 cells expressing JNK KTR were stimulated with LPS 100 ng/ml.
(D) PC-12 cells expressing ERK KTR were stimulated with NGF 25 ng/ml.
(E) Primary Bone Marrow-derived Macrophages (BMDMs) isolated from mice expressing JNK KTR were stimulated with LPS 100 ng/ml.



Supplementary Figure 2

KTR translocation and the estimated active JNK concentration.

(A) Heatmap illustrating the experimental C/N ratio, simulated C/N ratio and predicted active JNK concentration upon IL-1β stimulation. Each row denotes a color coded individual time course of an indicated property for a single cell. This is the final output from the jupyter notebook demonstration. (B) The population average traces of the C/N ratio (red), active JNK concentration (blue) and total phosphorylated KTR concentration (green). Note that the C/N ratio is acquired by direct analysis of the KTR image data, while the JNK and KTR concentrations are calculated by further analysis using a computational mechanistic model of KTR phosphorylation and dephosphorylation as well as nuclear or cytoplasmic translocation. The concentrations are normalized by minimum and maximum values.

Supplementary Methods

Subsequent analysis for the estimation of absolute kinase concentration in single cells

In the following sections, we compile a working example for the estimation of absolute active kinase concentrations in single cells using the JNK KTR. We demonstrate this process using *covertrace* for cleaning, visualization and modeling of the extracted data. In order to demonstrate the computational analysis flows in a more understandable and broadly usable way, some parts of the pipeline have been modified from that described in our previous work[1]. We provide a Docker image so that users can easily reproduce our analysis and potentially adapt it for their own application (see Equipment Setup).

<u>Data cleaning and Modeling</u>

In this section, we use *covertrace* in a jupyter notebook for handling multi-dimensional time-series data. The software is also available at https://github.com/CovertLab/covertrace.

1. Run the container by the following commands:

    docker run -it -p 8888:8888 -v $WORKDIR:/home/ braysia/ktrprotocol

2. Download and extract jupyter notebook files.

    wget http://archive.simtk.org/ktrprotocol/ktr_notebooks.zip && unzip ktr_notebooks.zip

3. After tracking, initialize a jupyter notebook.

    jupyter notebook

Copy and paste the link (http://localhost:8888/?token=~) in a browser.

4. Open the extracted "/home/ktr_datacleaning.ipynb". Execute all the analysis steps contained in cells. See http://jupyter.readthedocs.io/en/latest/index.html for how to execute jupyter notebooks.

5. Open "/home/ktr_modeling.ipynb". The single-cell level active JNK concentration will be obtained as a final output (Figure S2).

Optional: We also provide the notebooks in a PDF format (Supplementary Note 1 for data cleaning and Supplementary Note 2 for modeling) but these are not runnable with Jupyter.

<u>Considerations</u>

1. The estimation of the k_d and K_{md} are probably the most difficult parameters to estimate since these two parameters are highly dependent on each other and are hard to constrain. The estimation becomes more robust if one parameter can be obtained experimentally -- using, for example, phosphospecific antibodies or mass spectrometry[5,6].
2. Although we used a fixed population-level value for r_{total}, it can be measured experimentally at a single-cell level using a combination of absolute protein quantification and calibration with fluorescent beads[7,8].

Nature Protocols: doi:10.1038/nprot.2017.128

Supplementary Table 1: KTRs and H2B fused to a set of fluorescent proteins available at Addgene

	Cyan (mCerulean3)	Yellow (mClover)	Red (mRuby2)	Far-red (iRFP670)
ERK KTR	Addgene #90229	Addgene #90227	Addgene #90231	
JNK KTR	Addgene #90232	Addgene #59151	Addgene #59154	
p38 KTR	Addgene #59155	Addgene #90233		
PKA KTR		Addgene #59153		
H2B FP (nuclear marker)	Addgene #90234	Addgene #90235	Addgene #90236	Addgene #90237
JNK KTR AA		Addgene #90238		
JNK KTR EE		Addgene #90239		
JNK KTR AE		Addgene #90240		

Supplementary Table 2: Description of sample image files

The “?” is a placeholder for which any value can be substituted.

Directory name	File names	Reporters	Treatment	Time interval
LMB/Pos004	img_0000000??_YFP_000.png	JNK KTR AA mutant fused to mClover	LMB treatment at 2 min	30 sec
	img_0000000??_DAPI_000.png	Hoechst-stained nuclei		
LMB/Pos005	img_0000000??_YFP_000.png	JNK KTR EE mutant fused to mClover	LMB treatment at 28 min	2 min before the treatment and 30 sec after the treatment
	img_0000000??_DAPI_000.png	Hoechst-stained nuclei		
AnisoInh/Pos00?	img_0000000??_YFP_000.png	JNK KTR fused to mClover	Anisomycin treatment at 20 min and JNK inhibitor VIII treatment at 90 min	2 min
	img_0000000??_DAPI_000.png	Hoechst-stained nuclei		
IL1B/Pos00?	img_0000000??_TRITC_000.png	JNK KTR fused to mRuby2	IL1β treatment at 17.5 min	2 min
	img_0000000??_YFP_000.png	JNK KTR AE mutant fused to mClover		
	img_0000000??_DAPI_000.png	Hoechst-stained nuclei		

Supplementary Table 3: Image processing pipelines in CellTK

Nature Protocols: doi:10.1038/nprot.2017.128

If there are several functions in one process, they will be applied in the order given here.

Process	Function name	Brief description
preprocess	flatfield_references	Take median of several empty images and subtract it from the raw images.This accounts for background subtraction and illumination bias correction.
	align	Calculate image jitter over time and crop images for alignment. (optional for some experiments)
	histogram_match	Normalize pixel values between subsequent nuclear images using histogram matching[9]. This accounts for sudden changes in overall intensity due to microscope error or changing the media (optional for some experiments).
	curvature_anisotropic_smooth	Noise removal filter using anisotropic diffusion is applied to nuclear images[10] (optional for some experiments).
	gaussian_laplace	Compute the laplacian of gaussian image. When NEG=True, it will makes an image negative so that nuclear regions have positive pixels.
segment	adaptive_thres	Each pixel is classified as nuclear if its intensity is above the blurred image's pixel intensity. These process enable adaptive thresholding which is suitable for detecting nuclei with different brightness.
subdetect	propagate_multisnakes	Propagate from the segmented foreground using a custom Active Contours Without Edges[11].
tracking	run_lap	Nuclei in consecutive frames are linked by minimizing a global linking cost based on squared distances between nuclei[12]. Linking is also thresholded by nuclear intensity changes.
	track_neck_cut	For each pixel on the boundary of unlinked nuclear objects, local concavities is calculated based on the interior angle between two vectors directed to distant pixels on the same boundaries[13]. Objects are iteratively separated at two points with high concavities until one object is linked to a nucleus in a previous frame based on changes in intensity and distance.
	nearest_neighbor	Attempt to connect unlinked nuclear objects based on changes in intensity and distance. It accounts for capturing remained objects with a large movement.
postprocess	gap_closing	Nuclei which are not present in several frames and then reappeared are linked based on proximity in intensity and distance. This accounts for tracking of cells in non-consecutive frames.
	cut_short_traces	Cells are discarded if they are not tracked for at least a given number of frames.
subdetect	ring_dilation_above_offset_buffer	Cytoplasmic regions are segmented by dilating a few pixels away from nuclear segmentation. Furthermore, the pixel is classified into background if its intensity is below the offsetted background intensity estimated by the regional minimum filter.

Nature Protocols: doi:10.1038/nprot.2017.128

1. Regot, S. *et al.* High-Sensitivity Measurements of Multiple Kinase Activities in Live Single Cells. *Cell* **157,** 1724–1734 (2014).
2. Lowekamp, B. C., Chen, D. T., Ibáñez, L. & Blezek, D. The Design of SimpleITK. *Front. Neuroinform.* **7,** 45 (2013).
3. Kamentsky, L. *et al.* Improved structure, function and compatibility for CellProfiler: modular high-throughput image analysis software. *Bioinformatics* **27,** 1179–1180 (2011).
4. Regot, S., Hughey, J. J., Bajar, B. T., Carrasco, S. & Covert, M. W. High-sensitivity measurements of multiple kinase activities in live single cells. *Cell* **157,** 1724–1734 (2014).
5. Eissler, C. L. *et al.* A general strategy for studying multisite protein phosphorylation using label-free selected reaction monitoring mass spectrometry. *Anal. Biochem.* **418,** 267–275 (2011).
6. Cundell, M. J. *et al.* A PP2A-B55 recognition signal controls substrate dephosphorylation kinetics during mitotic exit. *J. Cell Biol.* **214,** 539–554 (2016).
7. Lawrimore, J., Bloom, K. S. & Salmon, E. D. Point centromeres contain more than a single centromere-specific Cse4 (CENP-A) nucleosome. *J. Cell Biol.* **195,** 573–582 (2011).
8. Graham, J. S., Johnson, R. C. & Marko, J. F. Counting proteins bound to a single DNA molecule. *Biochem. Biophys. Res. Commun.* **415,** 131–134 (2011).
9. Nyúl, L. G., Udupa, J. K. & Zhang, X. New variants of a method of MRI scale standardization. *IEEE Trans. Med. Imaging* **19,** 143–150 (2000).
10. Perona, P. & Malik, J. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.* **12,** 629–639 (1990).
11. Márquez-Neila, P., Baumela, L. & Alvarez, L. A morphological approach to curvature-based evolution of curves and surfaces. *IEEE Trans. Pattern Anal. Mach. Intell.* **36,** 2–17 (2014).
12. Jaqaman, K. *et al.* Robust single-particle tracking in live-cell time-lapse sequences. *Nat. Methods* **5,** 695–702 (2008).

Nature Protocols: doi:10.1038/nprot.2017.128

13. Neves, J. C., Castro, H., Tomás, A., Coimbra, M. & Proença, H. Detection and separation of overlapping cells based on contour concavity for Leishmania images. *Cytometry A* **85,** 491–500 (2014).

Nature Protocols: doi:10.1038/nprot.2017.128

ktr_datacleaning

1 Supplementary Notes 1

1.1 Scope:

Single-cell imaging data is often noisy. In order to make an accurate estimate of the active kinase concentration, we first want to remove outlier cells from the dataset. This example notebook will walk you through how to interact with your dataset and remove outlier data using the functions provided. We will clean the three data sets that we need using this process.

This method relies heavily on Python classes. The documentation is here for users who wish to learn more about how classes work.

First, we need to import the necessary Python packages to access all of the functions we need:

```
In [1]: from __future__ import division
        from covertrace.data_array import Sites, DataArray
        import matplotlib.pyplot as plt
        import numpy as np
        import os
        from os.path import abspath, dirname, join
        %matplotlib inline
```

Next, we import the functions that will help us access and manipulate the data:

```
In [2]: from covertrace import ops_filter
        from covertrace import ops_plotter
        from covertrace import ops_bool
```

Finally, we save the location of the data as a variable that we can access later:

```
In [3]: data_folder = '/home/output/'
```

1.1.1 Clean the first dataset: IL1β stimulation

The *Sites* object provides a convenient interface for handling complex live-cell data. Please take a look at the Github repository for details. Briefly, it stores all of the single cell traces for various properties and conditions for multiple positions.

Let's first look at the dataset where the JNK KTR was activated with IL1β. We load the position we want and specify the experimental conditions by setting the variables *sub_folders* and *conditions*. We load these data by instantiating a *Sites* object which we name *sites*. The file name 'df.npz' is the default file name from running *CellTK*, but modified data sets can be saved under different file names and loaded using this command.

```
In [4]: parent_folder = join(data_folder, 'IL1B')
        sub_folders = ['Pos005', 'Pos006', 'Pos007', 'Pos008',]
        conditions = ['IL1B', 'IL1B', 'IL1B', 'IL1B']
        sites = Sites(parent_folder, sub_folders, conditions, file_name='df.npz')
```

The *sites* object stores the data from each position, so typing *sites.Pos005* allows you to access various data for that specific position.

Here, for all the positions, we set the time stamps from 0 min to 147.5 min at 2.5 min intervals.

In addition, we run *sites.add_median_ratio*. This operation will add ratio of median intensity

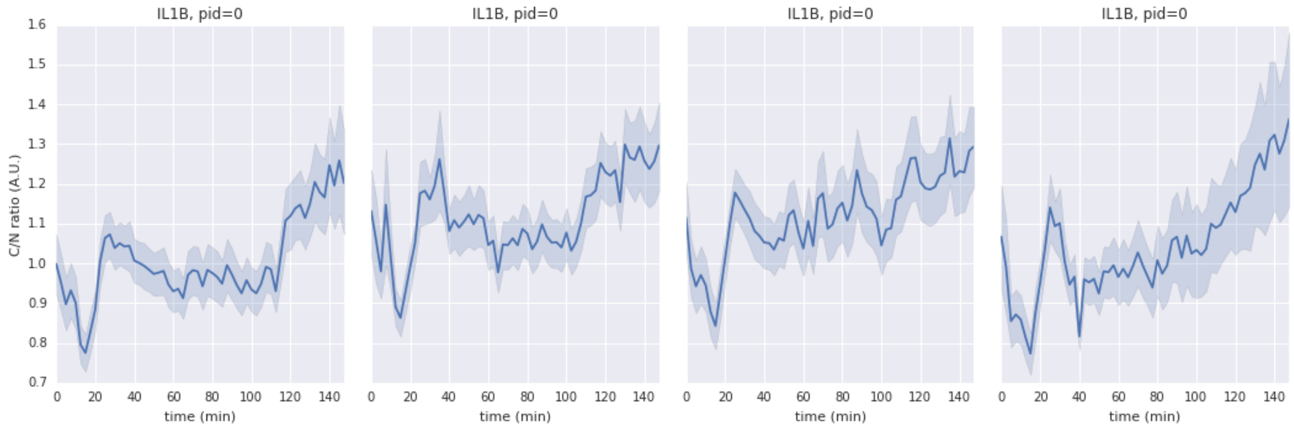
Nature Protocols: doi:10.1038/nprot.2017.128

```
In [5]: for key, site in sites.iteritems():
            site.time = np.arange(0, 150, 2.5)  # in minutes
        sites.add_median_ratio()
```

Sites class will allow ops_* functions to be applied to all the positions automatically. The functions most often used are those in the *ops_filter*, *ops_bool*, and *ops_plotter* modules that we imported above. In this case, we use *plot_tsplot function* from the *ops_plotter* module.

(Note: type `ops_plotter.plot_tsplot?` to see a description of what this function does.)

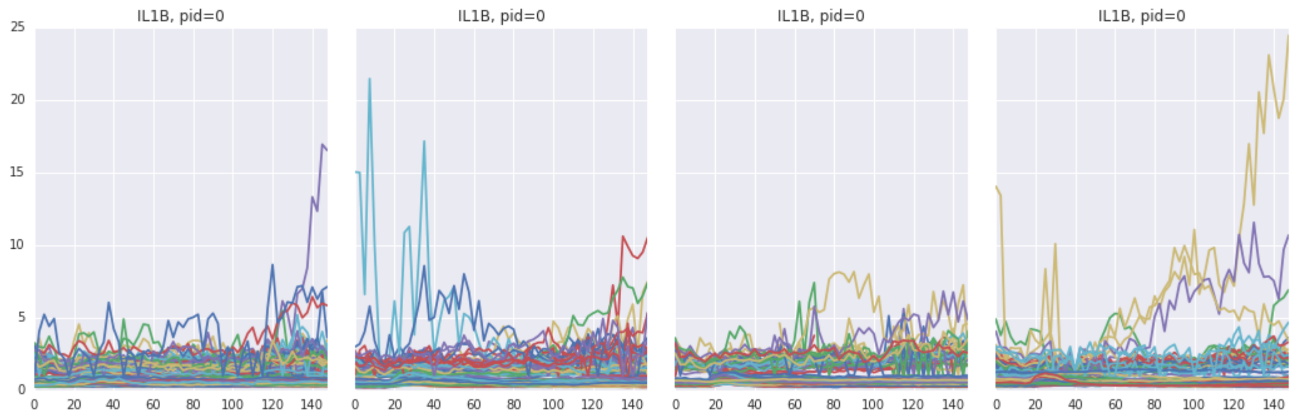
```
In [6]: fig, axes = ops_plotter.plot_tsplot(sites['cyto', 'TRITC', 'median_ratio'])
        [ax.set_xlabel('time (min)') for ax in axes];
        axes[0].set_ylabel('C/N ratio (A.U.)')

Out[6]: <matplotlib.text.Text at 0x7f8d84285f10>
```



Here is the plot generated by *ops_plotter.plot_tsplot*. It is a plot of the ratio of cytoplasmic to nuclear (C/N) median intensity of the TRITC signal. You can see the activation of JNK KTR upon IL1β stimulation.

Now take a look at the single-cell traces using the function *ops_plotter.plot_all*.

```
In [7]: fig, axes = ops_plotter.plot_all(sites['cyto', 'TRITC', 'median_ratio'])
```

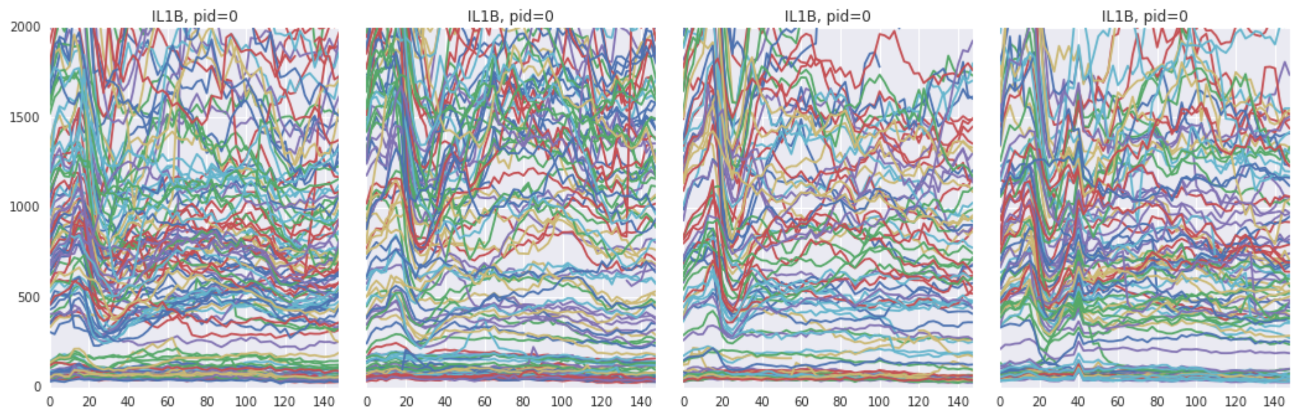


You can see that the single-cell traces are much noisier than the averaged traces above.

One typical cause of outliers is having cells that express very low amounts of the reporter. Let's take a look at the nuclear signal intensity to identify these cells.

```
In [8]: fig, axes = ops_plotter.plot_all(sites['nuc', 'TRITC', 'median_intensity'])
        [ax.set_ylim([0, 2000]) for ax in axes];
```


Nature Protocols: doi:10.1038/nprot.2017.128



Indeed, you can see two distinct populations in terms of the reporter levels in the nuclei. We want to exclude the cells that aren't expressing high amounts of the reporter. The process for doing this is first to assign a 'Property ID' (*pid*) to these cells and then remove all cells with that *pid*.

Let's assign *pid = 1* to all cells with a median intensity less than 250. We use **ops_bool.filter_frames_by_range** to mark frames with a median intensity less than the specified lower bound. All *ops_bool* functions are used to mark frames that meet certain criteria, in this case having a value lower than a cutoff threshold.

Now when we plot data, it separates the plots based on *pid*.

```
In [9]: ops_bool.filter_frames_by_range(sites['nuc', 'TRITC', 'median_intensity'], LOWER=250)
```

We can also check that cells with *pid = 1* have a noisier C/N ratio. We'll plot all position and *pid* combinations to see this.

Sites class has *canvas* as an attribute, and this can be used to modify subplot configuration. Set a number of rows to 2.

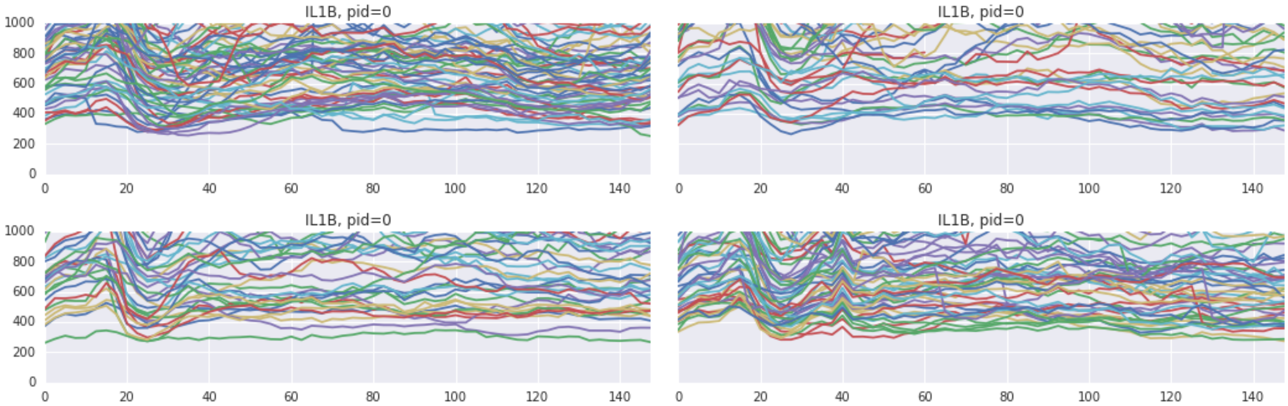
```
In [10]: sites.canvas.num_row = 2
         fig, axes = ops_plotter.plot_all(sites['nuc', 'TRITC', 'median_ratio'])
```

Notice that the plots above with *pid=1* in their title have a much noisier C/N ratio.

We can now remove the cells with *pid=1* and confirm that they were removed from the data set. The function *sites.drop_prop* removes all cells containing the given *pid*.

```
In [11]: sites.drop_prop(1)
         fig, axes = ops_plotter.plot_all(sites['nuc', 'TRITC', 'median_intensity'])
         [ax.set_ylim([0, 1000]) for ax in axes];
```


Nature Protocols: doi:10.1038/nprot.2017.128



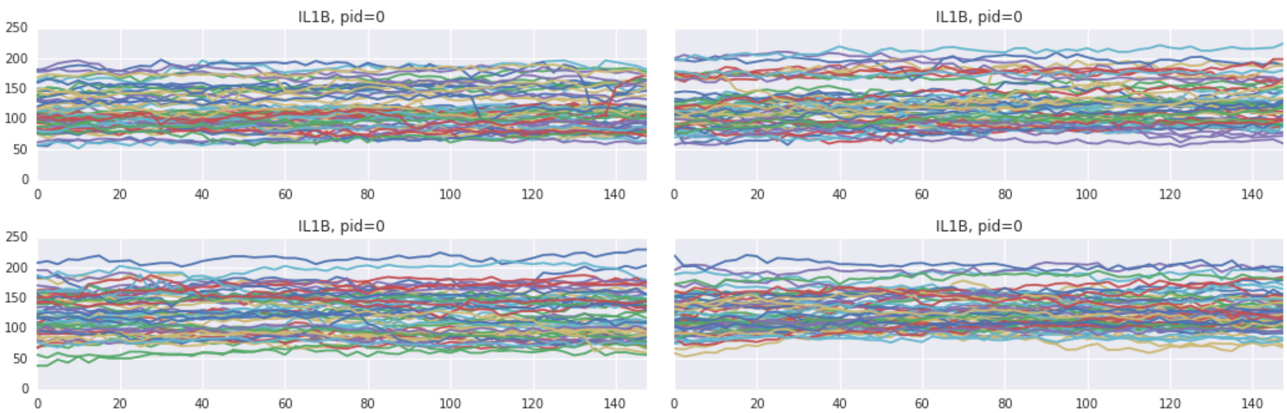
Another frequent cause of noisy data is errors in the image segmentation. Particularly when cells are confluent, it is sometimes hard to segment many pixels from the cytoplasm, thus the data extracted may not be reliable. Therefore, we remove cells that have too few pixels.

```
In [12]: fig, axes = ops_plotter.plot_all(sites['cyto', 'TRITC', 'area'])
```


```
In [13]: ops_bool.filter_frames_by_range(sites['cyto', 'TRITC', 'area'], LOWER=60, UPPER=130)
         sites.drop_prop(1)
```

Run a similar operation for nuclear area. Too small or too big nuclei often mean segmentation error or unhealthy cells.

```
In [14]: fig, axes = ops_plotter.plot_all(sites['nuc', 'TRITC', 'area'])
```

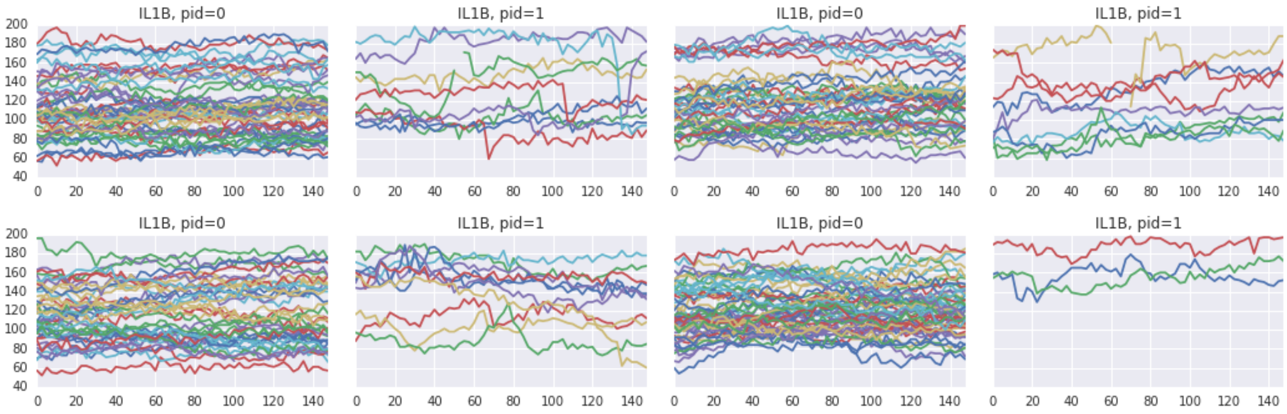


Nature Protocols: doi:10.1038/nprot.2017.128

```
In [15]: ops_bool.filter_frames_by_range(sites['nuc', 'TRITC', 'area'], LOWER=50, UPPER=200)
         sites.drop_prop(1)
```

Another frequent cause of noise is a sudden change in nuclear area. This may be caused by cell death, cell division, or mis-segmentation. Use the same process as above to remove cells with a sudden nuclear area change. Let's filter out cells if nuclear area changes more than 15 pixels. The function *ops_bool.filter_frames_by_diff* can be used to find cells with sudden area changes.

```
In [16]: ops_bool.filter_frames_by_diff(sites['nuc', 'TRITC', 'area'], THRES=15)
         fig, axes = ops_plotter.plot_all(sites['nuc', 'TRITC', 'area']);
```

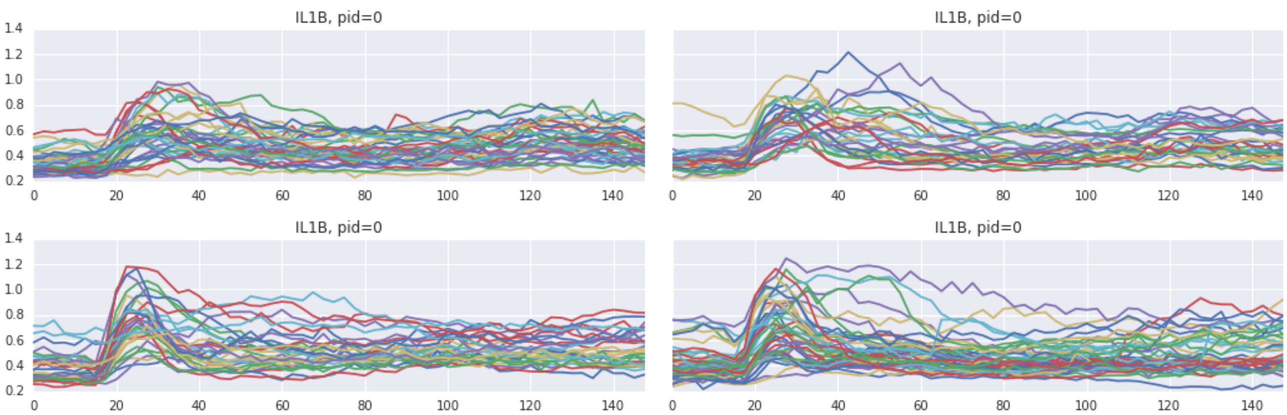



```
In [17]: sites.drop_prop(1)
```

When the signal-to-noise ratio is low, noise in the measurements can drastically affect the normalization of normalize nuclear intensity by cytoplasmic intensity. For simplicity, let's exclude cells which have expression levels outside of the 10th and 95th percentile in either nucleus or cytoplasm. This can be accomplished with the function *ops_bool.filter_frames_by_percentile_stats*.
We also use ops_bool.cut_short_traces to remove cells containing NaN.

```
In [18]: ops_bool.filter_frames_by_percentile_stats(sites['cyto', 'TRITC', 'median_intensity'], LOWER=10, UPPER=95)
         ops_bool.filter_frames_by_percentile_stats(sites['nuc', 'TRITC', 'median_intensity'], LOWER=10, UPPER=95)
         ops_bool.cut_short_traces(sites['cyto', 'TRITC', 'median_ratio'], MINFRAME=60)
         sites.drop_prop(1)
```

```
In [19]: ops_plotter.plot_all(sites['cyto', 'TRITC', 'median_ratio']);
```



Nature Protocols: doi:10.1038/nprot.2017.128

The cleaned dataset can now be saved as *df_cleaned.npz* by using *sites.save*.

```
In [20]: sites.save('df_cleaned.npz')
```

Now the single-cell traces are significantly less noisy. You can use the Bokeh notebook below to zoom and pan to investigate the data closer.

```
In [21]: import bokeh.io
         bokeh.io.output_notebook()
         import bokeh.mpl
         import bokeh.plotting
```

```
/opt/conda/lib/python2.7/site-packages/bokeh/util/deprecation.py:34: BokehDeprecationWarning: MPL compatibility can no long
  warn(message)
```

```
In [22]: fig, axes = ops_plotter.plot_all(sites['cyto', 'TRITC', 'median_ratio'])
         fig.set_size_inches((4, 4))
         bokeh.plotting.show(bokeh.mpl.to_bokeh())  # Add this line after you plot
```

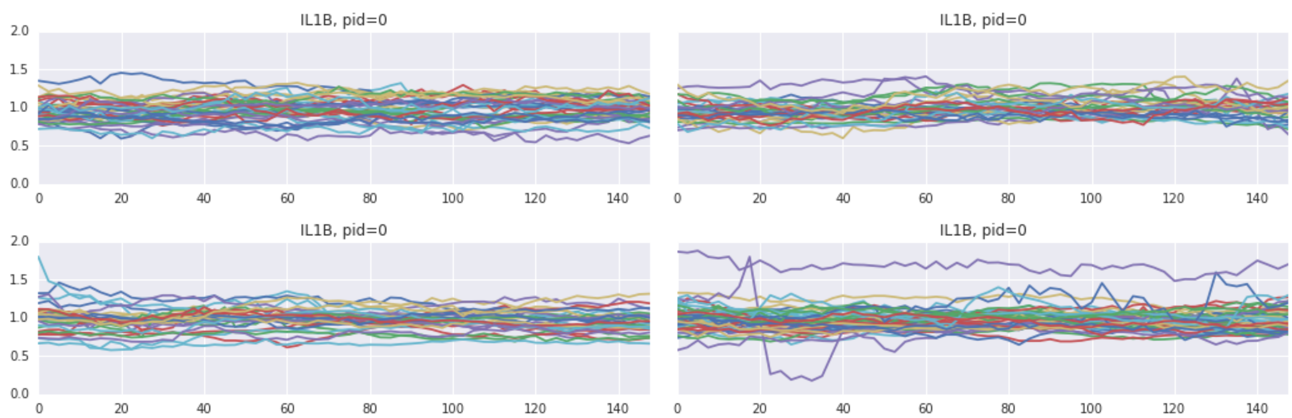
Congratulations!!! We've gone through one data cleaning example.

This is just one example for cleaning this data set, and different experiments require different processes. As you can see, data cleaning entails iterating through visualization and filtering steps for multiple wells.

1.1.2 Cleaning the YFP channel

We also need to do some cleaning on the YFP channel.
In this experiment, cells express both the non-responsive JNK KTR AE mutant in the YFP channel in addition to the wild-type JNK KTR in the TRITC channel.

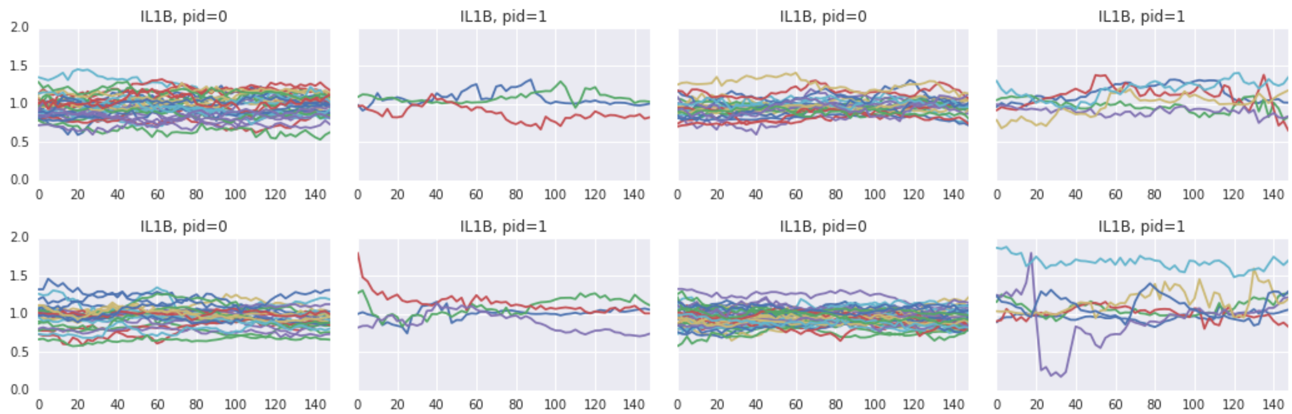
```
In [23]: ops_plotter.plot_all(sites['cyto', 'YFP', 'median_ratio']);
```



We will only be using the average trace for the JNK AE mutant when we start modeling, so filter out traces with outliers and drastic changes that will skew the average.

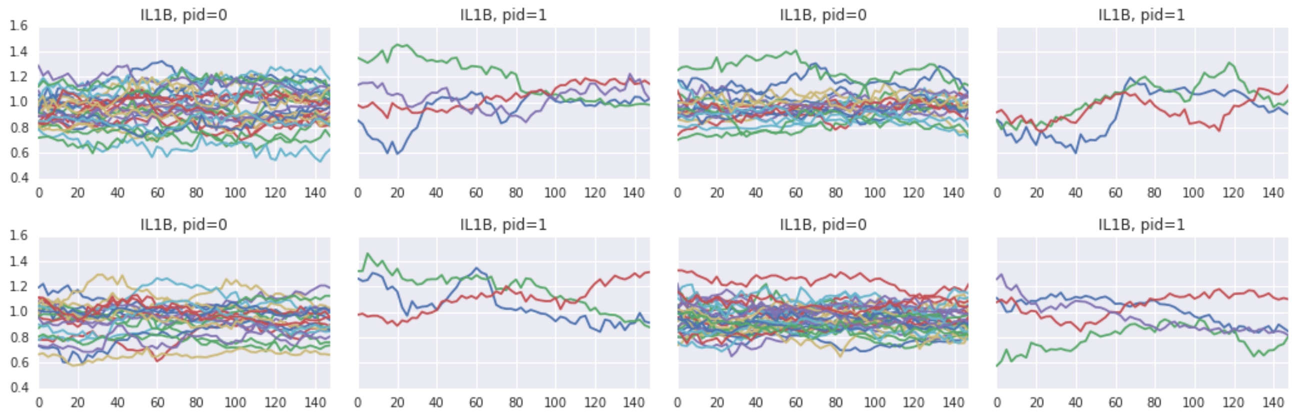
```
In [24]: ops_bool.filter_frames_by_diff(sites['cyto', 'YFP', 'median_ratio'], THRES=0.15)
         fig, axes = ops_plotter.plot_all(sites['cyto', 'YFP', 'median_ratio'])
```


Nature Protocols: doi:10.1038/nprot.2017.128



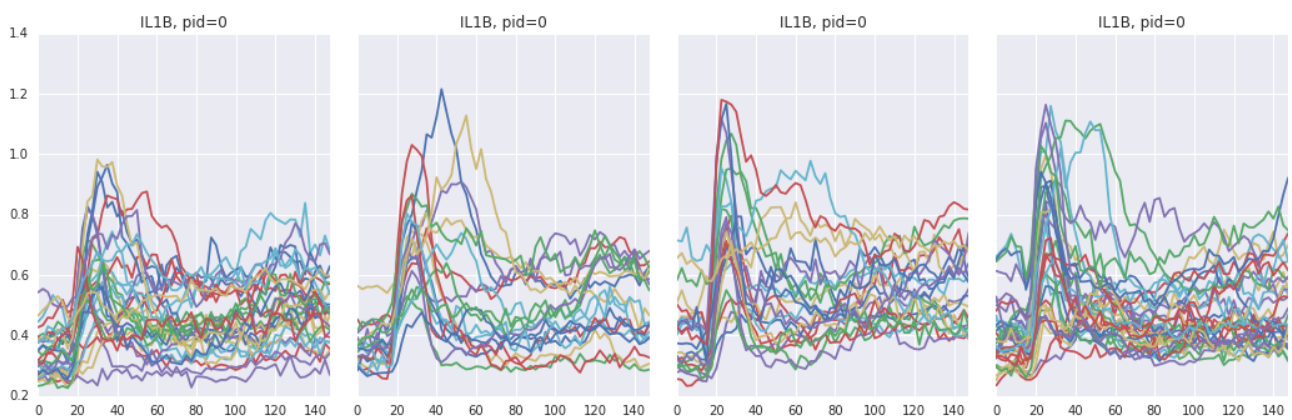

```
In [25]: sites.drop_prop(1)
```

```
In [26]: ops_bool.filter_frames_by_percentile_stats(sites['cyto', 'YFP', 'median_ratio'], func=np.nanstd, UPPER=90)
         ops_plotter.plot_all(sites['cyto', 'YFP', 'median_ratio']);
```

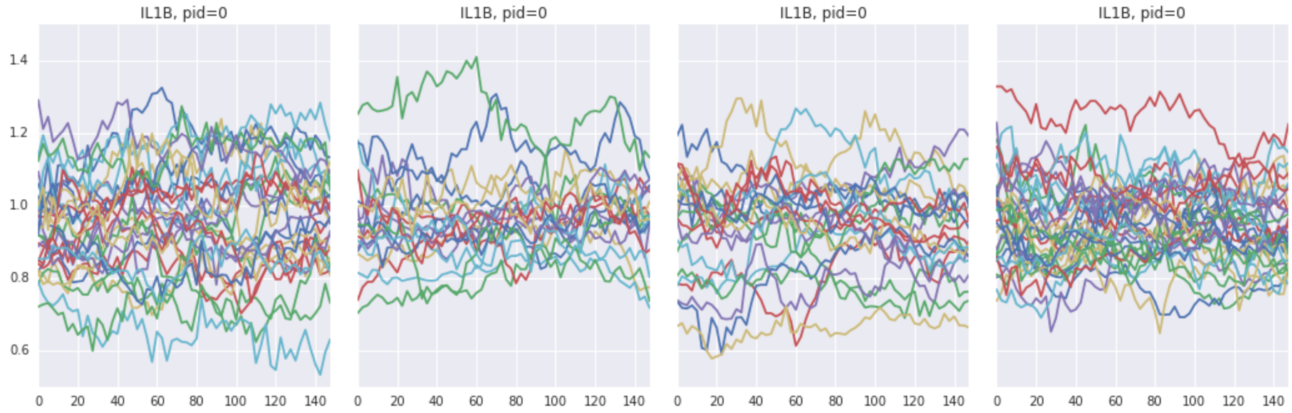



```
In [27]: sites.drop_prop(1)
```

```
In [28]: sites.canvas.num_row = 1
         ops_plotter.plot_all(sites['cyto', 'TRITC', 'median_ratio']);
         ops_plotter.plot_all(sites['cyto', 'YFP', 'median_ratio']);
         ops_bool.cut_short_traces(sites['cyto', 'TRITC', 'median_ratio'], MINFRAME=60)
         sites.save('df_cleaned.npz')
```



Nature Protocols: doi:10.1038/nprot.2017.128



1.1.3 Clean the second dataset: Anisomycin stimulation followed by JNK inhibitor

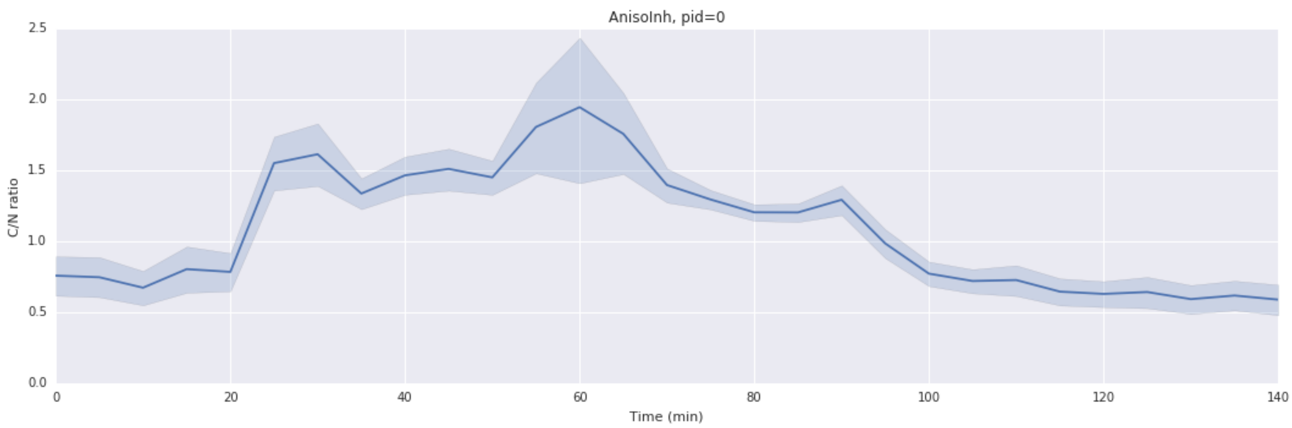
We are going to use the same process to clean the other datasets that we need.
Now we will clean the dataset where the JNK KTR was activated with anisomycin and then inactivated with JNK inhibitor.
This time we have only one position.

```
In [29]: parent_folder = join(data_folder, 'AnisoInh')
         sub_folders = ['Pos0']
         conditions = ['AnisoInh']
         sites_anis = Sites(parent_folder, sub_folders, conditions, file_name='df.npz')
```

```
In [30]: sites_anis.Pos0.time = np.arange(29) * 5  # every 5 min
         sites_anis.add_median_ratio()
```

```
In [31]: fig, axes = ops_plotter.plot_tsplot(sites_anis['cyto', 'YFP', 'median_ratio'])
         [ax.set_xlabel('Time (min)') for ax in axes]
         [ax.set_ylabel('C/N ratio') for ax in axes]
```

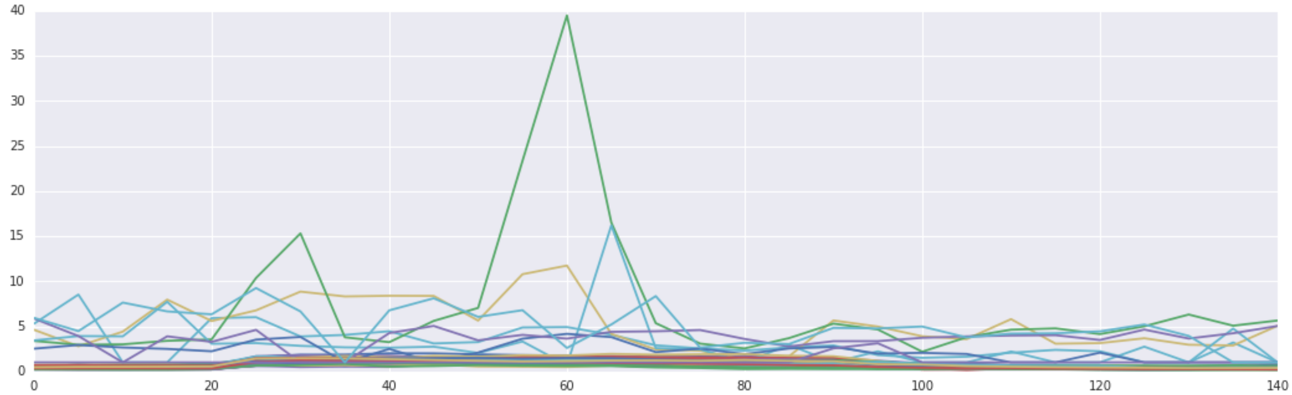
```
Out[31]: [<matplotlib.text.Text at 0x7f8d83460810>]
```



You can see the activation of JNK KTR followed by inhibition.

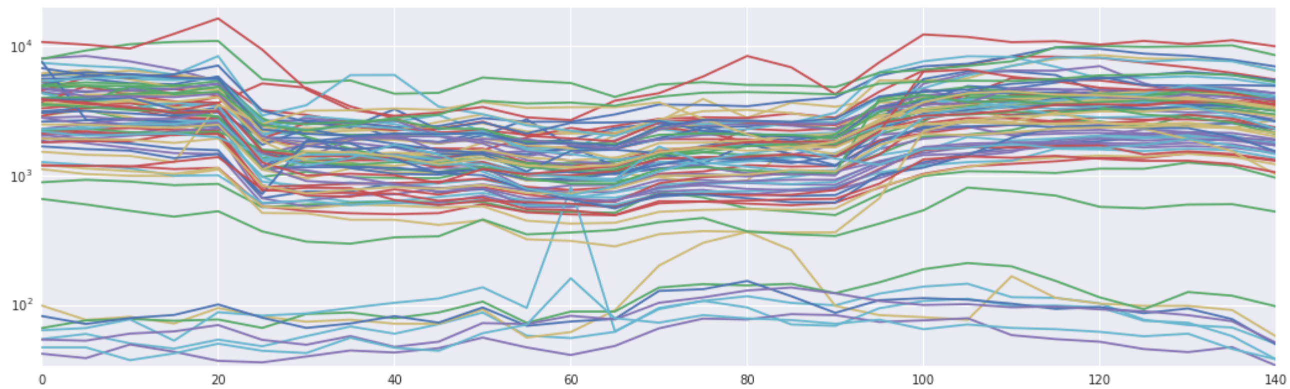
Nature Protocols: doi:10.1038/nprot.2017.128

```
In [32]: fig, axes = ops_plotter.plot_all(sites_anis.Pos0['cyto', 'YFP', 'median_ratio'])
```

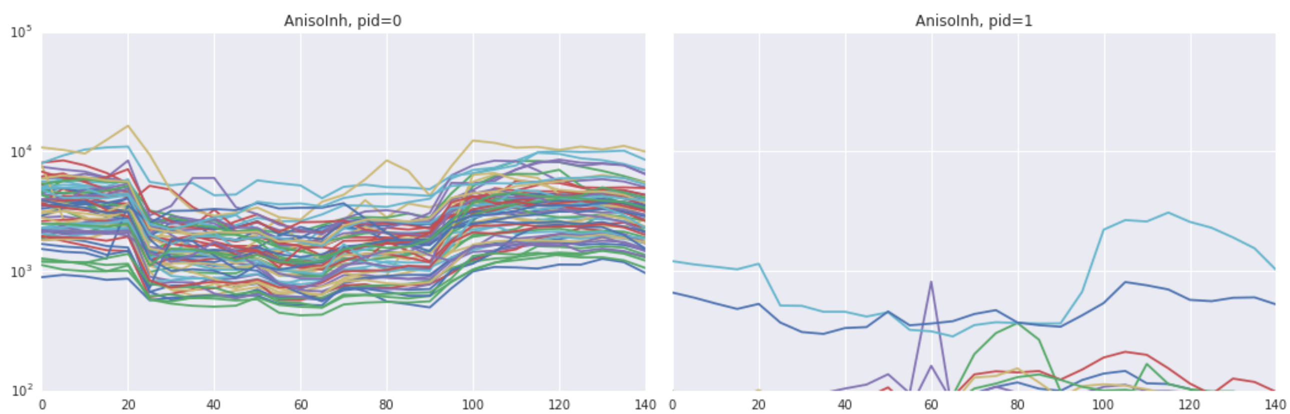


Again, you see some outliers before cleaning.
First remove cells without reporter expression.

```
In [33]: fig, axes = ops_plotter.plot_all(sites_anis.Pos0['nuc', 'YFP', 'median_intensity'], logy=True)
         [ax.set_ylim([0, 20000]) for ax in axes]

Out[33]: [(34.0, 20000)]
```



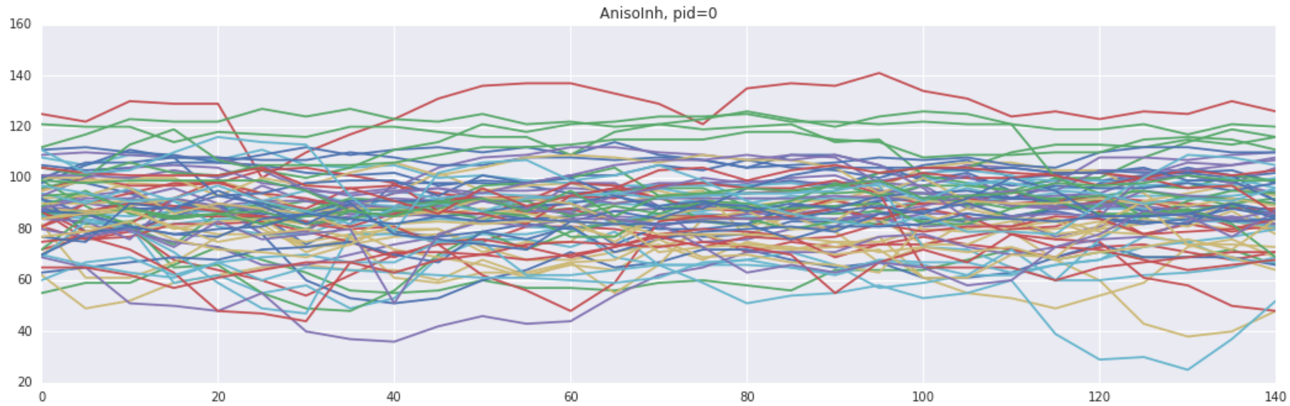
```
In [34]: ops_bool.filter_frames_by_range(sites_anis['nuc', 'YFP', 'median_intensity'], LOWER=300)
         fig, axes = ops_plotter.plot_all(sites_anis['nuc', 'YFP', 'median_intensity'], logy=True)
```



Nature Protocols: doi:10.1038/nprot.2017.128

```
In [35]: sites_anis.drop_prop(1)
```

```
In [36]: fig, axes = ops_plotter.plot_all(sites_anis['cyto', 'YFP', 'area'])
```



Drop cells which have less than 60 cytoplasmic pixels or more than 130 pixels.

```
In [37]: ops_bool.filter_frames_by_range(sites_anis['cyto', 'YFP', 'area'], LOWER=40, UPPER=130)
         sites_anis.drop_prop(1)
```

Detect sudden changes in nuclear area.

```
In [38]: ops_bool.calc_rolling_func_filter(sites_anis['nuc', 'YFP', 'area'], threshold=20, window=5)
         fig, axes = ops_plotter.plot_all(sites_anis['nuc', 'YFP', 'area'])

/opt/conda/lib/python2.7/site-packages/covertrace-0.1-py2.7.egg/covertrace/ops_bool.py:192: FutureWarning: pd.rolling_mean
        DataFrame.rolling(min_periods=0,window=5,center=True).mean()
```

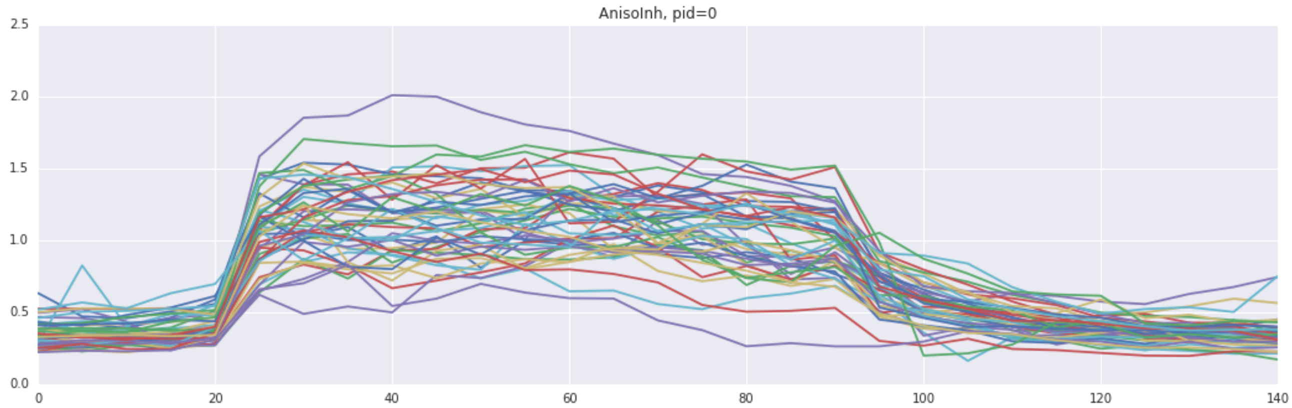

```
In [39]: sites_anis.drop_prop(1)
```

Drop cells whose median intensity of either nuclei/cytoplasm are outside of the 10 to 95 percentile.

```
In [40]: ops_bool.filter_frames_by_percentile_stats(sites_anis['cyto', 'YFP', 'median_intensity'], LOWER=10, UPPER=95)
         ops_bool.filter_frames_by_percentile_stats(sites_anis['nuc', 'YFP', 'median_intensity'], LOWER=10, UPPER=95)
         ops_bool.cut_short_traces(sites_anis['cyto', 'YFP', 'median_ratio'], MINFRAME=29)
         sites_anis.drop_prop(1)
```


Nature Protocols: doi:10.1038/nprot.2017.128

```
In [41]: ops_plotter.plot_all(sites_anis['cyto', 'YFP', 'median_ratio']);
         sites_anis.save('df_cleaned.npz')
```



1.1.4 Clean the third dataset: Leptomycin B treatment of JNK KTR AA/EE mutants

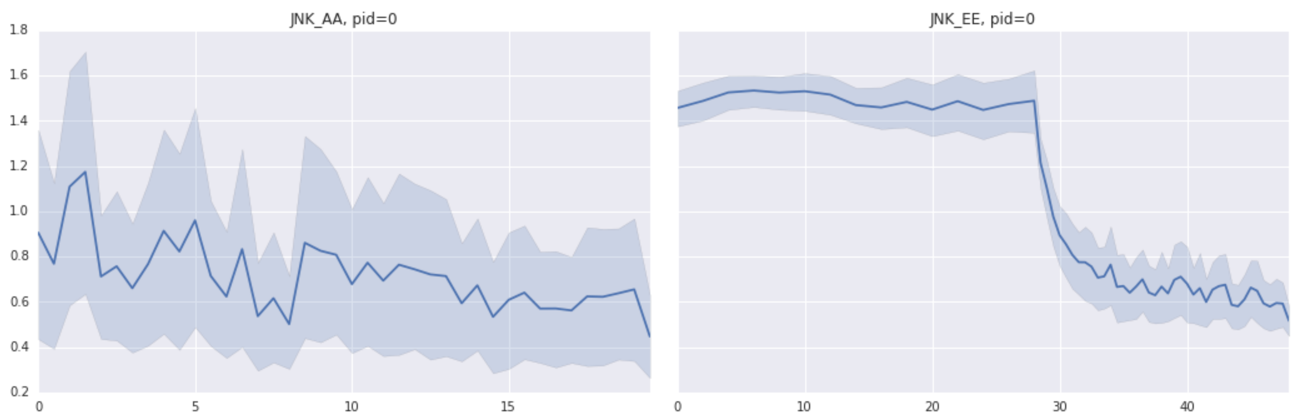
We are going to use the same process to clean the final data set that we need. In this dataset, there are two positions. One is JNK KTR AA mutants and the another is JNK KTR EE mutants.
Cells were treated with Leptomycin B (LMB) at 2 min and 28 min for JNK KTR AA mutants and JNK KTR EE mutants, respectively.

Here we see that the *sites* class can store multiple positions and conditions.

```
In [42]: parent_folder_mut = join(data_folder, 'LMB')
         sub_folders_mut = ['Pos004', 'Pos005']
         conditions_mut = ['JNK_AA', 'JNK_EE']
         sites_mut = Sites(parent_folder_mut, sub_folders_mut, conditions_mut, file_name='df.npz')
         # Set time stamps.
         sites_mut.Pos004.time = list(np.arange(0, 40) * 0.5)
         sites_mut.Pos005.time = list(np.concatenate((np.arange(0, 28, 2), np.arange(28, 48.5, 0.5))))
         sites_mut.add_median_ratio()
```

```
In [43]: ops_plotter.plot_tsplot(sites_mut['cyto', 'YFP', 'median_ratio']);
         axes[0].set_title('JNK KTR AA mutant')
         axes[1].set_title('JNK KTR EE mutant')
         axes[0].set_ylabel('C/N ratio')
```

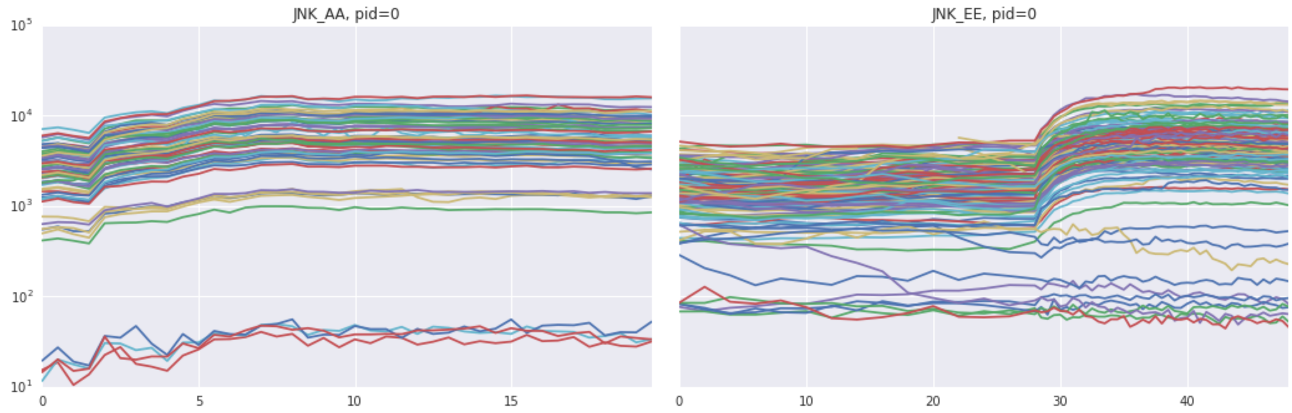
```
Out[43]: <matplotlib.text.Text at 0x7f8d8241ba10>
```



Nature Protocols: doi:10.1038/nprot.2017.128

The C/N ratio of JNK KTR EE mutants is almost equivalent to the active JNK KTR, indicating that it localizes to the cytoplasm more than JNK AA mutants do initially.
After Leptomycin B treatment, you can see that inhibition of nuclear export leads to nuclear localization for both mutants.

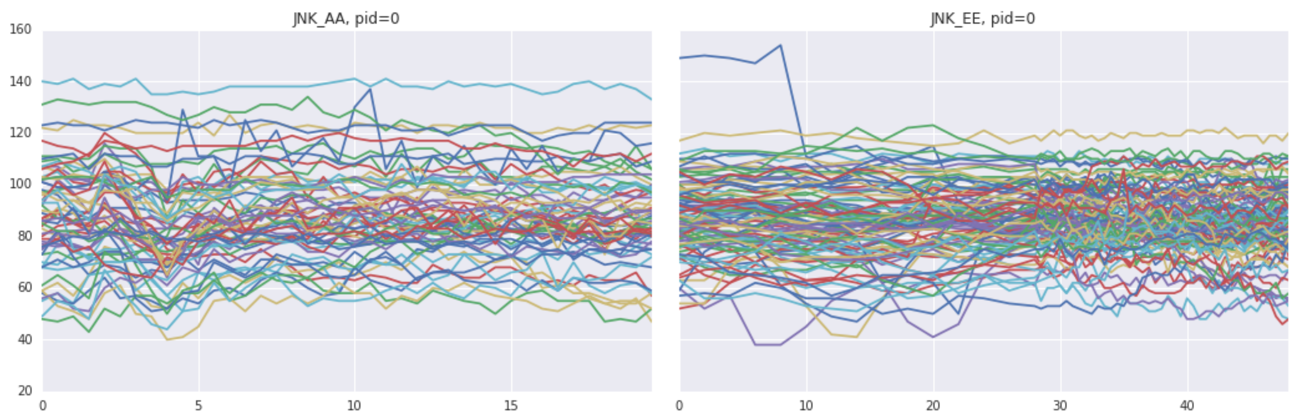
```
In [44]: ops_plotter.plot_all(sites_mut['nuc', 'YFP', 'median_intensity'], logy=True);
```



Now we can again exclude cells with low reporter expression levels, small nuclear areas, and sudden changes in nuclear area.

```
In [45]: ops_bool.filter_frames_by_range(sites_mut['nuc', 'YFP', 'median_intensity'], LOWER=750)
         sites_mut.drop_prop(1)
```

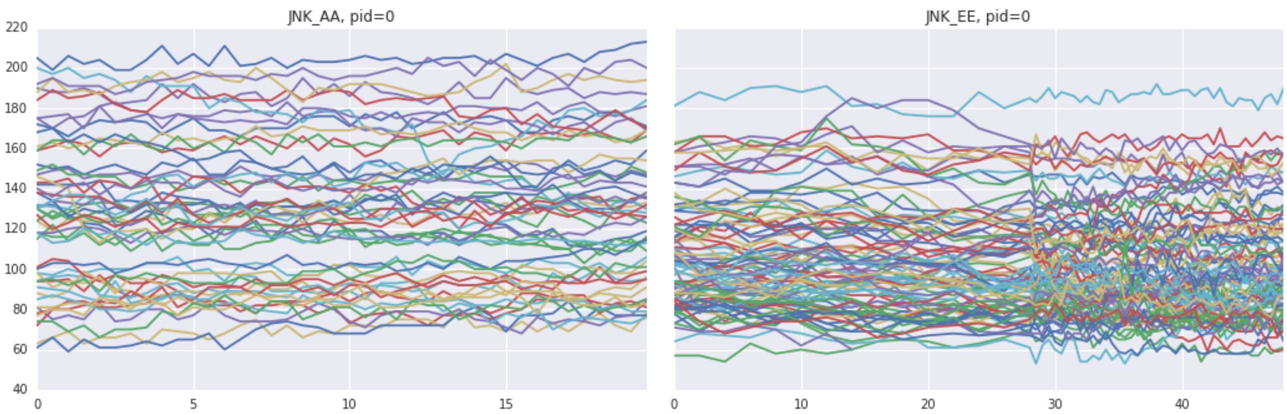
```
In [46]: ops_plotter.plot_all(sites_mut['cyto', 'YFP', 'area']);
```




```
In [47]: ops_bool.filter_frames_by_range(sites_mut['cyto', 'YFP', 'area'], LOWER=50, UPPER=130)
         sites_mut.drop_prop(1)
```

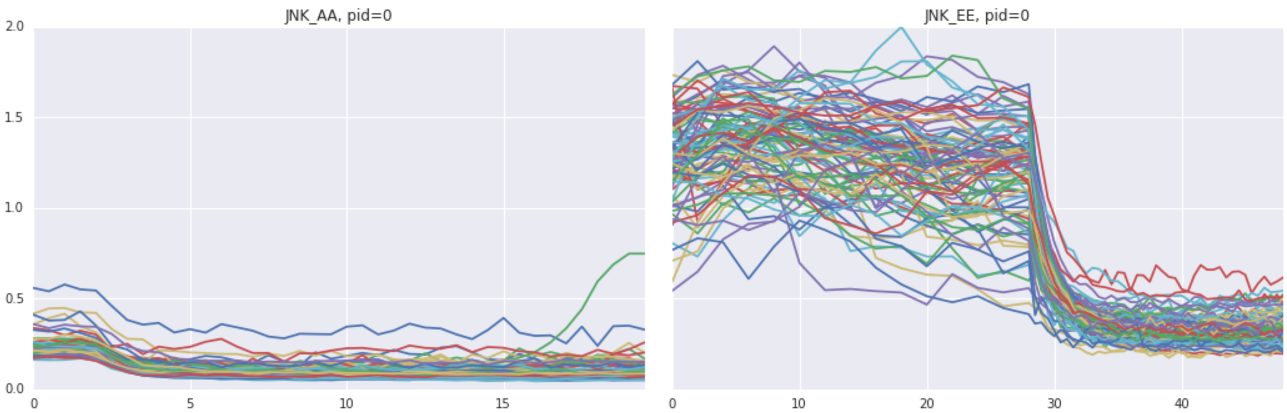
```
In [48]: ops_plotter.plot_all(sites_mut['nuc', 'YFP', 'area']);
```


Nature Protocols: doi:10.1038/nprot.2017.128




```
In [49]: ops_bool.calc_rolling_func_filter(sites_mut['cyto', 'YFP', 'area'], threshold=15, window=5)
         ops_bool.cut_short_traces(sites_mut.Pos004['cyto', 'YFP', 'median_ratio'], MINFRAME=40)
         ops_bool.cut_short_traces(sites_mut.Pos005['cyto', 'YFP', 'median_ratio'], MINFRAME=55)
         sites_mut.drop_prop(1)
```

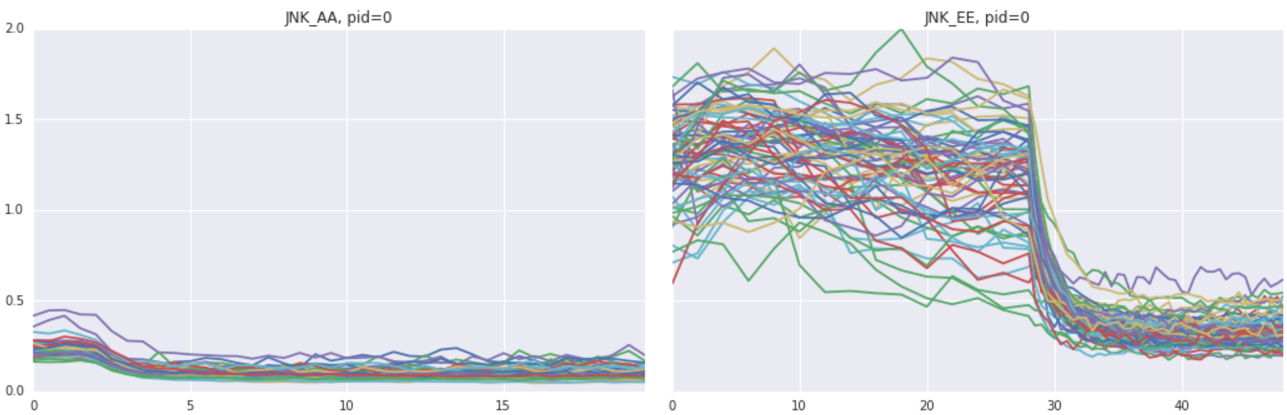
```
In [50]: ops_plotter.plot_all(sites_mut['cyto', 'YFP', 'median_ratio']);
```




```
In [51]: ops_bool.filter_frames_by_percentile_stats(sites_mut['cyto', 'YFP', 'median_intensity'], LOWER=10, UPPER=95)
         ops_bool.filter_frames_by_percentile_stats(sites_mut['nuc', 'YFP', 'median_intensity'], LOWER=10, UPPER=95)
         sites_mut.drop_prop(1)
```

The cleaned single-cell traces are noticeably less noisy than before.

```
In [52]: ops_plotter.plot_all(sites_mut['cyto', 'YFP', 'median_ratio']);
         sites_mut.save('df_cleaned.npz')
```



Nature Protocols: doi:10.1038/nprot.2017.128

ktr_modeling

1 Supplementary Notes 2

1.1 Scope:

This iPython notebook assumes that you've already used the process described in the data_cleaning notebook to exclude outlier cells and generate a clean dataset. This notebook will explain the process to parameterize the KTR model based on the cleaned imaging data.

As in the previous notebook, we begin by importing the packages we need and by defining the location of the data folder.

```
In [1]: from __future__ import division
        from covertrace.data_array import Sites, DataArray
        import matplotlib.pyplot as plt
        import numpy as np
        from functools import partial
        import os
        from os.path import abspath, dirname, join
        from scipy.integrate import odeint
        from scipy.optimize import minimize
        from scipy.signal import savgol_filter
        import seaborn as sns
        import random
        random.seed(1)
        %matplotlib inline
```

```
In [2]: from covertrace import ops_filter
        from covertrace import ops_plotter
        from covertrace import ops_bool
```

```
In [3]: data_folder = '/home/output/'
```

1.2 KTR Model

The following ordinary differential equations (ODE) describe the dynamics of the phosphorylated/dephosphorylated reporter in nucleus/cytosol.

$$\frac{dr_{cu}}{dt} = -kin_c(t) \cdot k_{cat}\frac{r_{cu}}{r_{cu}+K_m} + k_{dc} \cdot \frac{r_{cp}}{r_{cp}+K_{md}} - k_{iu} \cdot r_{cu} + k_{eu} \cdot r_{nu}$$

$$\frac{dr_{nu}}{dt} = -kin_n(t) \cdot k_{cat}\frac{r_{nu}}{r_{nu}+K_m} + k_{dn} \cdot \frac{r_{np}}{r_{np}+K_{md}} + k_v \cdot k_{iu} \cdot r_{cu} - k_v \cdot k_{eu} \cdot r_{nu}$$

$$\frac{dr_{cp}}{dt} = kin_c(t) \cdot k_{cat}\frac{r_{cu}}{r_{cu}+K_m} - k_{dc} \cdot \frac{r_{cp}}{r_{cp}+K_{md}} - k_{ip} \cdot r_{cp} + k_{ep} \cdot r_{np}$$

$$\frac{dr_{np}}{dt} = kin_n(t) \cdot k_{cat}\frac{r_{nu}}{r_{nu}+K_m} - k_{dn} \cdot \frac{r_{np}}{r_{np}+K_{md}} + k_v \cdot k_{ip} \cdot r_{cp} - k_v \cdot k_{ep} \cdot r_{np}$$

$$r_{cu} + r_{cp} + \frac{1}{k_v}(r_{nu} + r_{np}) = r_{total}$$

Symbol	description	Units
r_{total}	total concentration of reporter	μM

Nature Protocols: doi:10.1038/nprot.2017.128

Symbol	description	Units
r_{cu}	cytosolic unphosphorylated reporter	μM
r_{nu}	nuclear unphosphorylated reporter	μM
r_{cp}	cytosolic phosphorylated reporter	μM
r_{np}	nuclear phosphorylated reporter	μM
k_{iu}	nucler import rate of unphosphorylated reporter	min^{-1}
k_{eu}	nucler export rate of unphosphorylated reporter	min^{-1}
k_{ip}	nucler import rate of phosphorylated reporter	min^{-1}
k_{ep}	nucler export rate of phosphorylated reporter	min^{-1}
k_{cat}	catalytic rate constant of kinase phosphorylating the reporter	min^{-1}
K_m	Michaelis constant for kinase and reporter	μM
k_{dc}	dephosphorylation V_{max} of reporter in cytosol	$\mu M \cdot min^{-1}$
K_{md}	Michaelis constant for dephosphorylation of reporter	μM
$kin_c(t)$	time dependent concentrations of active kinase in cytosol	μM
$kin_n(t)$	time dependent concentrations of active kinase in nucleus	μM

We take some of these parameters from published literature. The rest will be estimated based on the collected data.

```
In [4]: r_total = 0.4   # uM
        k_cat = 20.0   # 1/min
        Km = 3.0   # uM
```

1.3 Estimation of cytosolic to nuclear volume ratio: kv

First obtain the parameter kv, the ratio of cytosolic volume to nuclear volume.
We assume that the total reporter concentration is constant for the time-scale of our experiments.

Therefore, $r_{c,1} + \frac{r_{n,1}}{k_v} = r_{c,2} + \frac{r_{n,2}}{k_v}$ where $r_{c,i}$ and $r_{n,i}$ are the cytosolic and nuclear concentrations of the reporter at condition i.

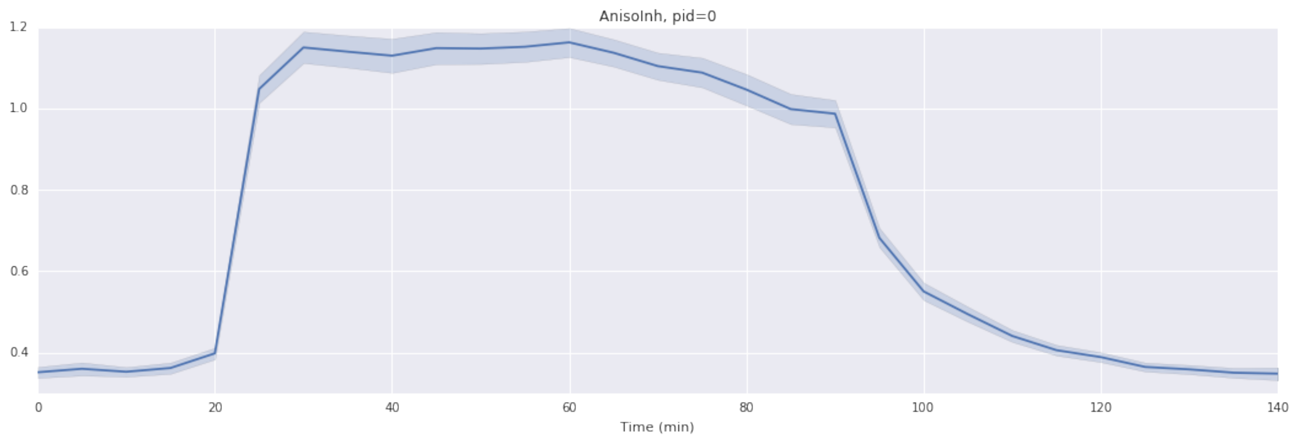
Solving for k_v, we get:
$k_v = \frac{r_{n,2} - r_{n,1}}{r_{c,1} - r_{c,2}}$.
A larger dynamic range in your data will give you a better signal to noise ratio, and thus a more accurate estimation of k_v. For this reason, we use the condition where the JNK KTR is activated by anisomycin and then inhibited by a JNK inhibitor.

```
In [5]: parent_folder = join(data_folder, 'AnisoInh')
        sub_folders = ['Pos0']
        conditions = ['AnisoInh']
        sites = Sites(parent_folder, sub_folders, conditions, file_name='df_cleaned.npz')
```

Take a look at the average traces for the median intensity of the cytoplasmic/nuclear (C/N) ratio. This dataset has already been cleaned.

```
In [6]: fig, axes = ops_plotter.plot_tsplot(sites['cyto', 'YFP', 'median_ratio'])
        axes[0].set_xlabel('Time (min)')

Out[6]: <matplotlib.text.Text at 0x7f15522bfe10>
```


Nature Protocols: doi:10.1038/nprot.2017.128



You can see the cytoplasmic translocation of the JNK KTR at 20 min when the cells are stimulated with anisomycin, followed by nuclear translocation at frame 90 min when the cells are inhibited with the JNK VIII inhibitor.
This combination of activation and inactivation will provide a large dynamic range, allowing robust parameter estimation.

Again, we use $k_v = \dfrac{r_{n,2} - r_{n,1}}{r_{c,1} - r_{c,2}}$.
Let's use the frames between 50 min to 80 min and 90 min to the end for our activated and inhibited conditions, respectively. We can access these data using *site.data* and take the average using the NumPy function *nanmean*.

```
In [7]: site = sites.Pos0
        time1 = np.array(site.time)
        n0m = np.nanmean(site['nuc', 'YFP', 'median_intensity'][:, (time1 >= 30) * (time1 <= 40)], axis=1)
        n1m = np.nanmean(site['nuc', 'YFP', 'median_intensity'][:, time1 >= 110], axis=1)
        c0m = np.nanmean(site['cyto', 'YFP', 'median_intensity'][:, (time1 >= 30) * (time1 <= 40)], axis=1)
        c1m = np.nanmean(site['cyto', 'YFP', 'median_intensity'][:, time1 >= 110], axis=1)
```

```
In [8]: estimated_k_v = float(np.nanmedian(np.array((n1m - n0m)/(c0m - c1m))))
        print 'k_v is {0:1.2f}'.format(estimated_k_v)
```

```
k_v is 4.00
```

Using these conditions, we estimate the average k_v to be 4.00.

1.4 Estimation of import and export rate constants: kiu, keu, kip, kep

Cytoplasmic translocation of the KTR happens because phosphorylation both decreases the nuclear export rate and increases the nuclear import rate. So now we need to find the rate constants for the import (k_i) and export (k_e) of phosphorylated (k_p) and unphosphorylated (k_u) reporters.

To estimate these constants, we utilize the fact that the JNK KTR AA and EE mutants approximate the unphosphorylated and phosphorylated states of the wild-type JNK KTR, respectively. Since AA and EE mutants are not phosphorylated, we assume that ke and ki for AA mutants is equal to keu and kiu. Similarly, ke and ki for EE mutants is equal to kep and kip.

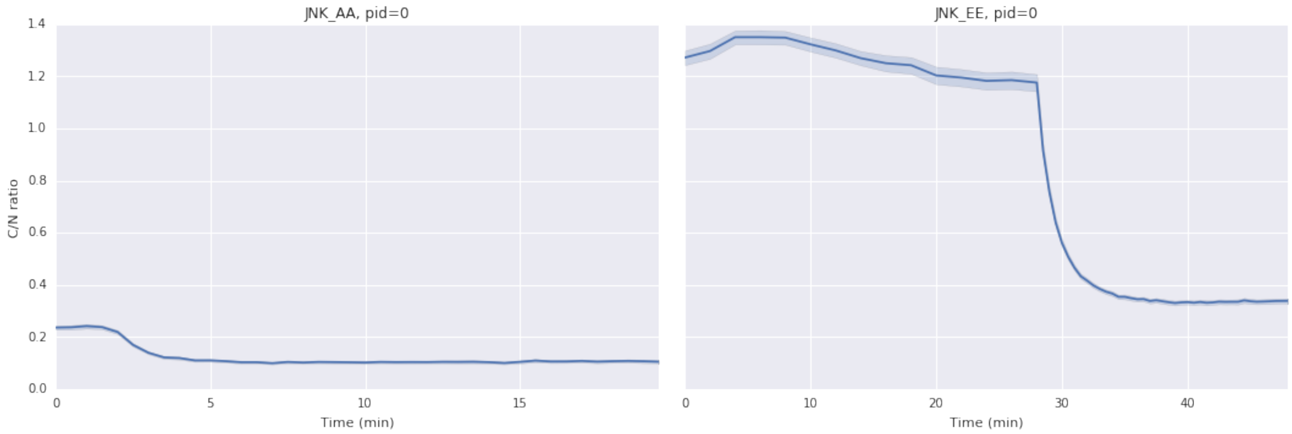
```
In [9]: parent_folder_mut = join(data_folder, 'LMB')
        sub_folders_mut = ['Pos004', 'Pos005']
        conditions_mut = ['JNK_AA', 'JNK_EE']
        sites_mut = Sites(parent_folder_mut, sub_folders_mut, conditions_mut, file_name='df_cleaned.npz')
```

Leptomycin B (LMB) is a potent inhibitor of nuclear export. The cells are treated with LMB at 2 min for Pos004 and at 28 min for Pos005, leading to a net nuclear translocation in both cases.

Nature Protocols: doi:10.1038/nprot.2017.128

```
In [10]: fig, axes = ops_plotter.plot_tsplot(sites_mut['cyto', 'YFP', 'median_ratio'])
         axes[0].set_ylabel('C/N ratio')
         [ax.set_xlabel('Time (min)') for ax in axes]
```

```
Out[10]: [<matplotlib.text.Text at 0x7f154ef6f0d0>,
          <matplotlib.text.Text at 0x7f154eaa96d0>]
```




```
In [11]: aa_ratio = np.array(sites_mut.Pos004['cyto', 'YFP', 'median_ratio'])
         ee_ratio = np.array(sites_mut.Pos005['cyto', 'YFP', 'median_ratio'])
```

We use the following system of differential equations to decribe reporter concentration in the nucleus (rn) and cytoplasm (rc).

$$\frac{dr_c}{dt} = -k_i r_c + k_e r_n$$

$$\frac{dr_n}{dt} = k_v k_i r_c - k_v k_e r_n$$

We create a function that stores these two differential equations and returns the solution given the input parameters. We will use a built-in ODE solver later to solve this function.

```
In [12]: def ode_mutant_model(y, t, *args):
             k_v, k_i, k_e = args[0], args[1], args[2]
             r_c, r_n = y[0], y[1]
             d_r_c = -k_i * r_c + k_e * r_n
             d_r_n = k_v * k_i * r_c - k_v * k_e * r_n
             return [d_r_c, d_r_n]
```

We assume the efficiency of LMB inhibition varies from cell-to-cell. This is expressed by parameter h; upon inhibition at time *tinh*, the export rate changes from k_e to hk_e.

The initial steady state of the system is $\frac{k_e}{k_i}$, and the steady state after the inhibition will be $\frac{hk_e}{k_i}$. We can use these values to estimate h for each cell.

```
In [13]: h_aa, h_ee = [], []
         for cell in aa_ratio:
             h_aa.append(np.mean(cell[30:])/np.mean(cell[:3]))
         for cell in ee_ratio:
             h_ee.append(np.mean(cell[30:])/np.mean(cell[:12]))
```

We would like to find the parameters h, k_e and k_i such that the model reproduces the time series of the LMB treatment experiment. Write a function that receives h, k_e and k_i and returns the predicted time series of the C/N ratio.

We again define the function. We also need the SciPy function *odeint* (http://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html), which can solve a system

Nature Protocols: doi:10.1038/nprot.2017.128

of ordinary differential equations. The function containing our ODE model (*ode_mutant_model*) is given as an input argument for *odeint*.

Our function (*calc_ts_sim_ratio*) inputs the parameters from the experiment, then runs *odeint* twice (once for the data before LMB treatment and once for the data after) to solve our ODE model. The returned values are the cytoplasmic/nuclear ratios before and after treatment.

```
In [14]: def calc_ts_sim_ratio(x, time, k_v, t_inh, h):
             ei_ratio = x[0]
             k_i = x[1]
             k_e = ei_ratio * k_i

             ini_r_c = 1.0
             ini_r_n = 1.0/ei_ratio
             params = (k_v, k_i, k_e)
             # Simulate before inhibition
             ts_pre_inh = odeint(ode_mutant_model, [ini_r_c, ini_r_n], time[time<t_inh], params, rtol=1e-4)
             r_pre_inh = np.array([i[0]/i[1] for i in ts_pre_inh])  # convert reporter profile to cytoplasmic/nuclear rati
             params = (k_v, k_i, k_e * h)
             # Simulate after inhibition
             ts_post_inh = odeint(ode_mutant_model, [ts_pre_inh[-1, 0], ts_pre_inh[-1, 1]], time[time>=t_inh], params, rtol
             r_post_inh = np.array([i[0]/i[1] for i in ts_post_inh])  # convert reporter profile to cytoplasmic/nuclear ra
             return np.concatenate((r_pre_inh, r_post_inh))
```

To start, run a simulation just for a single cell with an EE mutant reporter.

Before stimulation, $\frac{k_e}{k_i} = \frac{r_c}{r_n}$.

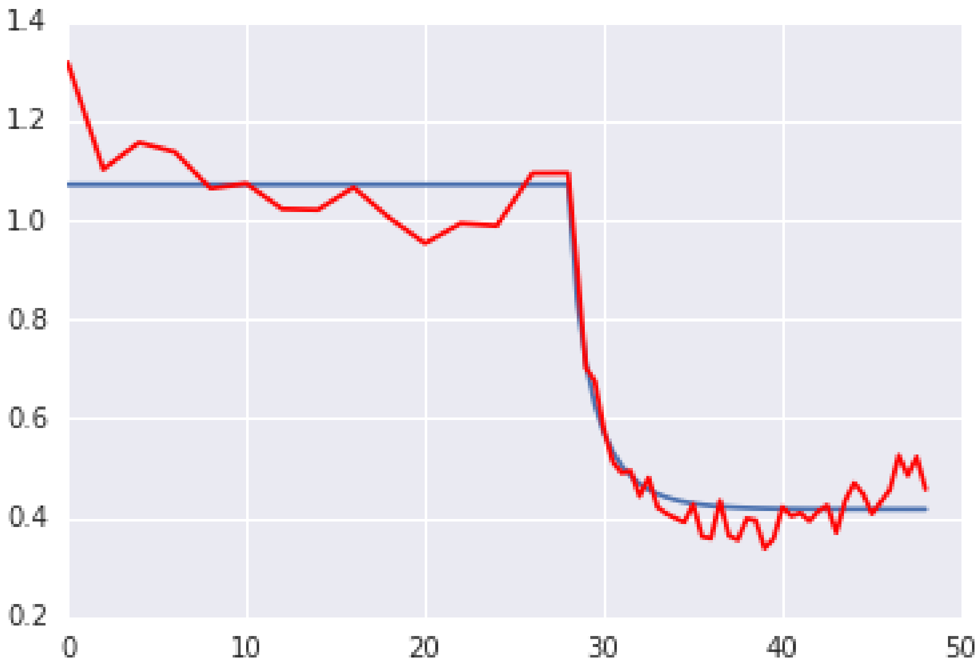
To optimize a fit to the experimental data, we define a cost function as the sum of squared distance between the experimentally measured trace and the estimated trace. We then use the SciPy function minimize to find values of k_e and k_i that minimize the cost function. Note that in this case we define a function using lambda, which allows the creation of anonymous functions (functions without a name).

```
In [15]: bnds = ((1e-6, None), (1e-6, None))
         time_ee = np.concatenate((np.arange(0, 28, 2), np.arange(28, 48.5, 0.5)))
         t_inh_ee = 27.5

         cell_id = 2
         first_cell = ee_ratio[cell_id, :]  # time-series of C/N ratio of a cell.

         func = lambda x: ((calc_ts_sim_ratio(x, time_ee, estimated_k_v, t_inh_ee, h_ee[cell_id]) - first_cell)**2).sum()
         ret = minimize(func, x0=[np.mean(first_cell[:13]), 0.1], bounds=bnds, method='L-BFGS-B')  # run the optimization
         est_ei_ratio, est_k_i = ret.x  # parameters
         est_k_e = est_ei_ratio * est_k_i
         print 'k_e and k_i are estimated to be {0:1.3f} and {1:1.3f} for this cell.'.format(est_k_e, est_k_i)

k_e and k_i are estimated to be 0.194 and 0.182 for this cell.
```

After estimating the parameters, we can compare the predicted time series and experimental time series for this cell.

```
In [16]: ts_pre_inh = odeint(ode_mutant_model, [est_k_e, est_k_i], time_ee[time_ee<=t_inh_ee],  (estimated_k_v, est_k_i, es
         ts_post_inh = odeint(ode_mutant_model, [est_k_e, est_k_i], time_ee[time_ee>t_inh_ee],  (estimated_k_v, est_k_i, h_

         plt.plot(time_ee, np.concatenate(([i[0]/i[1] for i in ts_pre_inh], [i[0]/i[1] for i in ts_post_inh])))
         # plt.hold(True)
         plt.plot(time_ee, first_cell, 'r');
```


Nature Protocols: doi:10.1038/nprot.2017.128



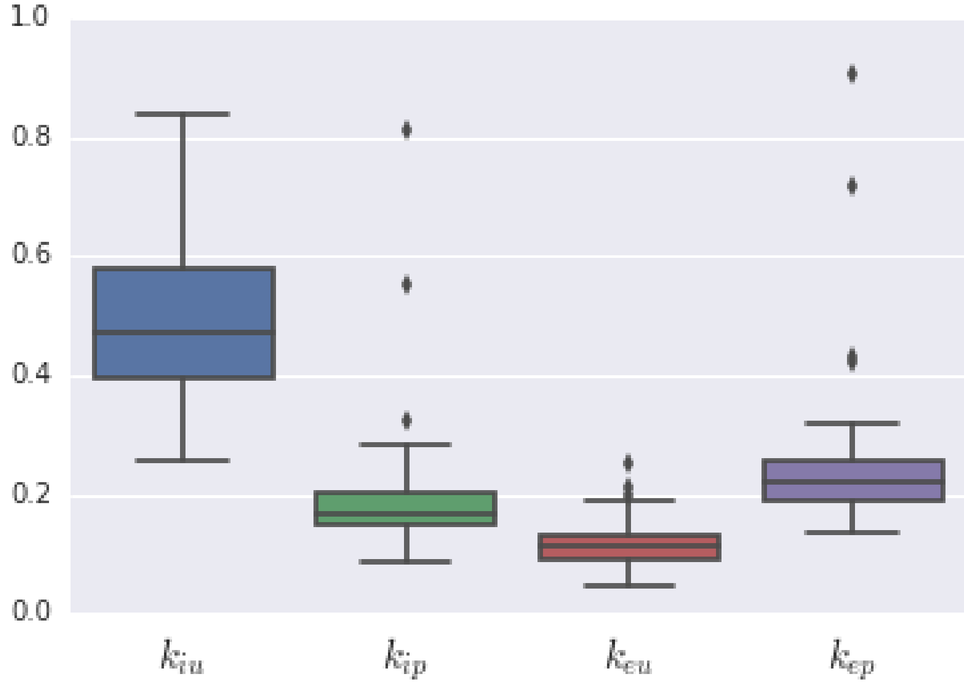
Now we expand this for all the cells in the experiement using a *for* loop to iterate over the set of cells.

```
In [17]: param_ee_store = []
         for cell, sc_h in zip(ee_ratio, h_ee):
             func = lambda x: ((calc_ts_sim_ratio(x, time_ee, estimated_k_v, t_inh_ee, sc_h) - cell)**2).sum()
             ret = minimize(func, x0=[np.median(cell[:13]), 0.1], bounds=bnds, method='L-BFGS-B', tol=1e-6)
             if ret.success:
                 param_ee_store.append(ret.x)
         est_k_ip = [i[1] for i in param_ee_store]
         est_k_ep = [i[0]*i[1] for i in param_ee_store]
```

Repeat the process and run a simulation for all the cells with AA mutant reporters.

```
In [18]: param_aa_store = []

         time_aa = np.arange(0, 40) * 0.5 # /min, Imaged every 30 sec
         t_inh_aa = 1.5    # /min, timing of inihibiton
         for cell, sc_h in zip(aa_ratio, h_aa):
             func = lambda x: ((calc_ts_sim_ratio(x, time_aa, estimated_k_v, t_inh_aa, sc_h) - cell)**2).sum()
             ret = minimize(func, x0=[np.mean(cell[:3]), 0.1], bounds=bnds, method='L-BFGS-B', tol=1e-6)
             if ret.success:
                 param_aa_store.append(ret.x)

         est_k_eu = [i[0]*i[1] for i in param_aa_store]
         est_k_iu = [i[1] for i in param_aa_store]

In [19]: ax = sns.boxplot(data=[est_k_iu, est_k_ip, est_k_eu, est_k_ep])
         ax.set_xticklabels(['$k_{iu}$', '$k_{ip}$', '$k_{eu}$', '$k_{ep}$'], fontsize=15);
         print "k_iu:{0:1.3}\tk_ip:{1:1.3}\nk_eu:{2:1.3}\tk_ep:{3:1.3}".format(np.nanmedian(est_k_iu), np.nanmedian(est_k_i
         ax.set_ylim([0, 1.0])

k_iu:0.471        k_ip:0.166
k_eu:0.112        k_ep:0.22


Out[19]: (0, 1.0)
```


Nature Protocols: doi:10.1038/nprot.2017.128

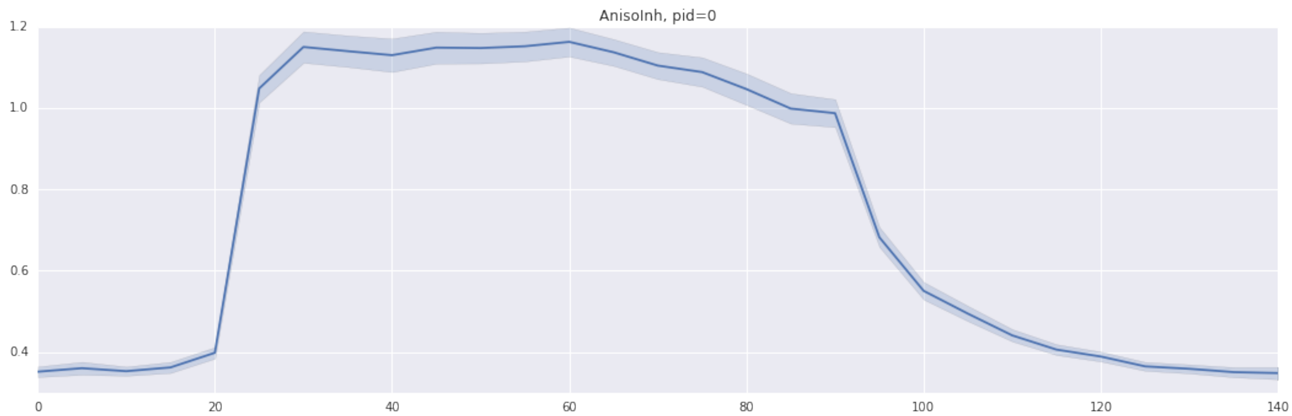


As expected, you can see that the estimated import rate is larger for unphosphorylated reporters, whereas the export rate is larger for phosphorylated reporters.
The combination of these effects causes the cytoplasmic translocation of KTRs upon phosphorylation.

1.5 Estimation of dephosphorylation parameters: k_d and K_{md}

After a reporter is phosphorylated, it has to get dephosphorylated by enzymes already present in the cell. k_d and K_{md} are the maximum dephosphorylation rate and Michaelis-Menten constant of reporter dephosphorylation, respectively. Here we assume k_d is equal in the cytosol and the nucleus.

```
In [20]: fig, axes = ops_plotter.plot_tsplot(sites['cyto', 'YFP', 'median_ratio'])
         site = sites.Pos0
```



We fit k_d and K_{md} to a model such that it reproduces the decay of the C/N ratio after kinase inhibitor treatment at 90 min. We assume that the JNK inhibitor begins inhibition instantly after it is applied to the cells, and that the extent of inhibition varies from cell to cell.

Nature Protocols: doi:10.1038/nprot.2017.128

We are going to use the average import/export rate constants estimated in the previous section. First, make a dictionary containing the parameter set.

```
In [21]: param_set = dict(k_v=estimated_k_v,
                          k_iu=np.nanmedian(est_k_iu), k_eu=np.nanmedian(est_k_eu),
                          k_ip=np.nanmedian(est_k_ip), k_ep=np.nanmedian(est_k_ep),
                          k_cat=k_cat, Km=Km, r_total=r_total)
```

Now make a model for the KTR system describing the phosphorylation, dephosphorylation, and nuclear-cytosolic shuttling of the KTR.

$$\frac{dr_{cu}}{dt} = -kin_c(t) \cdot k_{cat}\frac{r_{cu}}{r_{cu}+K_m} + k_{dc} \cdot \frac{r_{cp}}{r_{cp}+K_{md}} - k_{iu} \cdot r_{cu} + k_{eu} \cdot r_{nu}$$

$$\frac{dr_{nu}}{dt} = -kin_n(t) \cdot k_{cat}\frac{r_{nu}}{r_{nu}+K_m} + k_{dn} \cdot \frac{r_{np}}{r_{np}+K_{md}} + k_v \cdot k_{iu} \cdot r_{cu} - k_v \cdot k_{eu} \cdot r_{nu}$$

$$\frac{dr_{cp}}{dt} = kin_c(t) \cdot k_{cat}\frac{r_{cu}}{r_{cu}+K_m} - k_{dc} \cdot \frac{r_{cp}}{r_{cp}+K_{md}} - k_{ip} \cdot r_{cp} + k_{ep} \cdot r_{np}$$

$$\frac{dr_{np}}{dt} = kin_n(t) \cdot k_{cat}\frac{r_{nu}}{r_{nu}+K_m} - k_{dn} \cdot \frac{r_{np}}{r_{np}+K_{md}} + k_v \cdot k_{ip} \cdot r_{cp} - k_v \cdot k_{ep} \cdot r_{np}$$

$$r_{cu} + r_{cp} + \tfrac{1}{k_v}(r_{nu} + r_{np}) = r_{total}$$

As before, we specify the ODE model in a function.

```
In [22]: def main_ode(y, t, p):
             """kin_c_func and kin_n_func are the functions that return the active kinase concentration at time t.
             """
             c_u, n_u, c_p, n_p = y[0], y[1], y[2], y[3]

             k_v = p['k_v']
             k_iu, k_eu, k_ip, k_ep = p['k_iu'], p['k_eu'], p['k_ip'], p['k_ep']
             k_cat, Km, r_total = p['k_cat'], p['Km'], p['r_total']
             k_dc, k_dn, Kmd = p['k_d'], p['k_d'], p['Kmd']
             kin_c_func, kin_n_func = p['kin_c_func'], p['kin_n_func']

             d_c_u = -kin_c_func(t) * k_cat * c_u/(c_u + Km) + k_dc * c_p/(c_p + Kmd) - k_iu * c_u + k_eu * n_u
             d_n_u = -kin_n_func(t) * k_cat * n_u/(n_u + Km) + k_dn * n_p/(n_p + Kmd) + k_v * k_iu * c_u - k_v * k_eu * n_u
             d_c_p = kin_c_func(t) * k_cat * c_u/(c_u + Km) - k_dc * c_p/(c_p + Kmd) - k_ip * c_p + k_ep * n_p
             d_n_p = r_total - c_u - n_u/k_v - c_p - n_p/k_v
             return [d_c_u, d_n_u, d_c_p, d_n_p]
```

We assume the system is in steady state before inhibiton, thus at a given C/N ratio, we can calculate the reporter profile ($r_{cu}, r_{nu}, r_{cp}, r_{np}$) such that $\frac{dr}{dt}$ is 0.

First, write a function that will calculate the reporter profile given an active kinase concentration (*calc_rep_profile_at_steady state*). The function estimates the cytosolic and nuclear concentrations of the phosphorylated and unphosphorylated reporters to fit the experimental results at steady state ($\frac{dr}{dt} = 0$). We will use the NumPy function interp to do a linear interpolation at the steady-state values to use as our steady-state function.

We also write a second function (*calc_rcn_at_steady_state*) to return the C/N ratio at steady-state.

```
In [23]: def calc_rep_profile_at_steady_state(kin, pset):
             """At given kinase concentration and parameters,
             simulate the reporter profile (c_u, n_u, c_p, n_p)
             such that it minimizes sum of squared dy, assuming the pseudo-steady state.

             Output: reporter profile
             """
             ub = pset['r_total']
             pset['kin_c_func'] = lambda t: np.interp(t, [0, 1], [float(kin), float(kin)])
             pset['kin_n_func'] = lambda t: np.interp(t, [0, 1], [float(kin), float(kin)])
             x0 = [ub, 0, 0, 0]

             func = lambda y: (np.array(main_ode(y, 0, pset))**2).sum()
             ret = minimize(func, x0=x0, method='Powell', tol=1e-3)
```


Nature Protocols: doi:10.1038/nprot.2017.128

```
        return ret.x

    def calc_rcn_at_steady_state(kin, pset):
        """At given kinase concentration and parameters, simulate the C/N ratio at a steady-state.
        """
        x = calc_rep_profile_at_steady_state(kin, pset)
        return (x[0] + x[2])/(x[1] + x[3])
```

Next, we write a function (*estim_steady_state_kinase*) that inputs a steady-state C/N ratio and estimates a total kinase concentration (using *calc_rcn_at_steady_state*) that recreates the given steady state C/N ratio.

```
In [24]: def estim_steady_state_kinase(rcn, pset, bnd_min=0, bnd_max=None):
             """Given C/N ratio at the steady state, return active kinase concentration.
             """
             func1 = lambda kin1: ((calc_rcn_at_steady_state(kin1, pset) - rcn)**2).sum()
             ret1 = minimize(func1, x0=[0.01], bounds=((bnd_min, bnd_max), ), tol=1e-4)
             return ret1.x[0]
```

Finally, using the functions that we have just written, we define one more function that will be used to fit the parameters k_d and K_{md}. The function needs to estimate the intial (pre-inhibition) and final (post-inhibition) steady state kinase concentrations (using *estim_steady_state_kinase*) and the pre-inhibition reporter profile (using *calc_rep_profile_at_steady_state*). The function then returns an estimate of how the reporter profile changes post inhibition.

```
In [25]: KIN_MAX = 3.0  # ad hoc. Kinase concentration should not be as this high.
         def inhibitor_ode(x, time, rcn_init, rcn_final, pset):
             """
             Given k_d, Kmd and C/N ratio at the beginning and at the end, it will simulate the timecourse of C/N ratio.
             """
             pset['k_d'] = x[0]
             pset['Kmd'] = x[1]
             # Calculate initial kinase concentration such that it matches with given C/N ratio (rcn_init).
             kin_ini = estim_steady_state_kinase(rcn_init, pset, bnd_max=KIN_MAX)
             # At given kinase concentration, calculate reporter profiles.
             y0 = calc_rep_profile_at_steady_state(kin_ini, pset)
             # At given rcn_final, calculate kinase concentration at the final steady state.
             kin_after_inh = estim_steady_state_kinase(rcn_final, pset, bnd_max=kin_ini)

             # At time 0, inihibition started. kinase is inactive so it follows kinase (rcn_final)
             # but reporter profile at time 0 follows y0. Calculate cytoplasmic to nuclear times series.
             pset['kin_c_func'] = lambda t: np.interp(t, [time[0], time[-1]], [kin_after_inh, kin_after_inh])
             pset['kin_n_func'] = lambda t: np.interp(t, [time[0], time[-1]], [kin_after_inh, kin_after_inh])
             ts = odeint(main_ode, y0, time, (pset, ), rtol=1e-4)
             return (ts[:, 0] + ts[:, 2])/(ts[:, 1] + ts[:, 3])
```

Estimating k_d and K_{md} with this model is a difficult task, as these two parameters are heavily dependent on each other. For simplicity, we will fit with the population average time-series as opposed to each single-cell trace in order to save computation time.

To help generate a good fit, we will first guess the initial values by looping over a log-based parameter range and then subsequently fine tune the intial guess. We use a log-based range to avoid getting trapped in local solutions. Note, that we bound the upper limit of K_{md} as r_{total}. We use the NumPy function logspace to generate a distribution of initial values and then evaluate the squared distance from the calculated C/N ratio to the actual C/N ratio. The initial values that produce the best result are saved.

```
In [26]: med_rcn = np.array(np.nanmean(site['cyto', 'YFP', 'median_ratio'], axis=0))
         rcn_init = np.array(np.nanmean(med_rcn[18]))
         rcn_final = med_rcn[-1]
         decay_rcn = med_rcn[18:]

         time = np.arange(len(decay_rcn)) * 5  # every 5 min
         func = lambda x: ((inhibitor_ode(x, time, rcn_init, rcn_final, param_set) - decay_rcn)**2).sum()  # cost function
```


Nature Protocols: doi:10.1038/nprot.2017.128

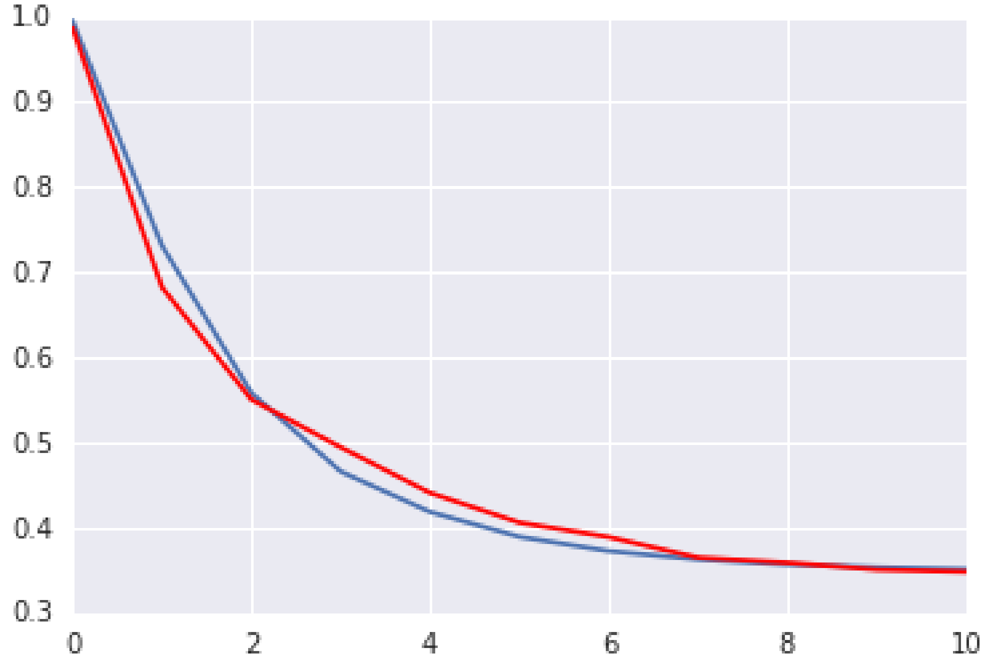
```
kd_range, kmd_range = np.logspace(-2, 0, 5), np.logspace(-2, np.log10(param_set['r_total']), 5)

store = [np.Inf, np.Inf, np.Inf]
for kdi in kd_range:
    for kmdi in kmd_range:
        res = func([kdi, kmdi])
        if store[0] > res:  # if an output from the cost function is smaller than previous ones, update store.
            store = [res, kdi, kmdi]
ini_kd, ini_kmd = store[1], store[2]
print ini_kd, ini_kmd
```

```
/opt/conda/lib/python2.7/site-packages/scipy/integrate/odepack.py:218: ODEintWarning: Excess work done on this call (perhap
  warnings.warn(warning_msg, ODEintWarning)


0.0316227766017 0.159054145753
```

Now, we can use the *minimize* function to more accurately estimate k_d and K_{md} using the initial guesses we just generated.

```
In [27]: # The estimation of k_d and Kmd for all the single cells takes a time.
         cons = ({'type': 'ineq', 'fun': lambda x: x[0]},
                 {'type': 'ineq', 'fun': lambda x: x[1]},
                 {'type': 'ineq', 'fun': lambda x: KIN_MAX - x[0]},
                 {'type': 'ineq', 'fun': lambda x: param_set['r_total'] - x[1]})
         ret = minimize(func, x0=[ini_kd, ini_kmd], constraints=cons, method='COBYLA', tol=1e-4)
         est_k_d, est_Kmd = ret.x[0], ret.x[1]

         param_set['k_d'] = est_k_d
         param_set['Kmd'] = est_Kmd

         plt.plot(inhibitor_ode([est_k_d, est_Kmd], time, rcn_init, rcn_final, param_set))
         plt.hold(True)
         plt.plot(decay_rcn, 'r')
         print 'The estimated k_d and Kmd for this cell is {0:1.3f} and {1:1.3f}'.format(est_k_d, est_Kmd)
```

```
The estimated k_d and Kmd for this cell is 0.026 and 0.170
```



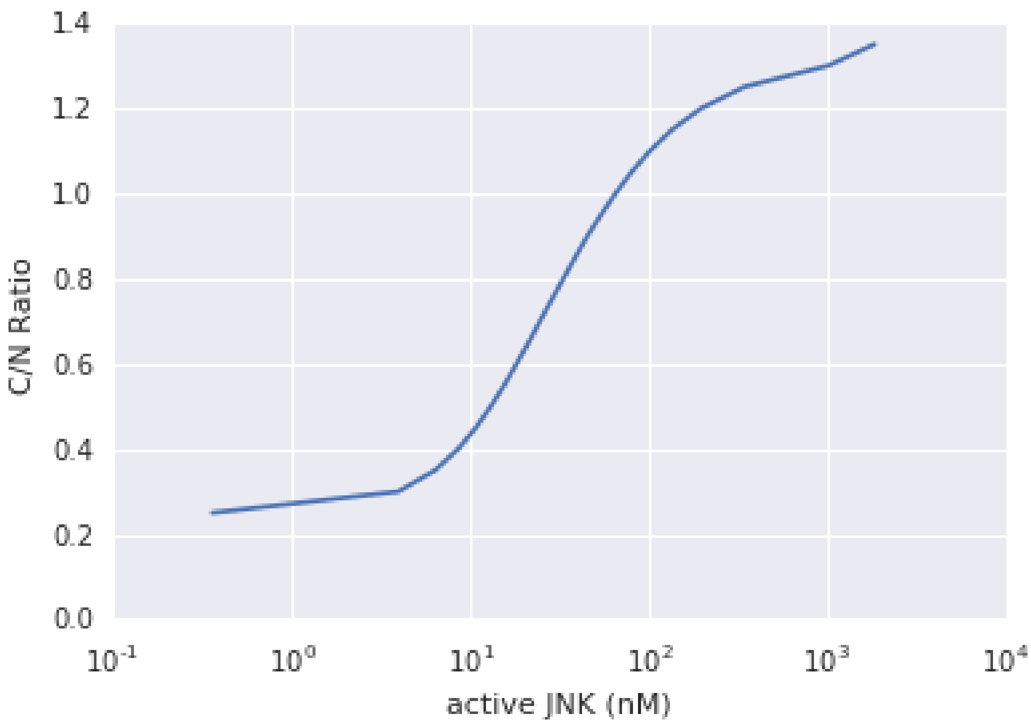
Nature Protocols: doi:10.1038/nprot.2017.128

1.6 Absolute kinase concentration

Now we have obtained all of the parameters we need to estimate absolute kinase concentration over time. First, plot a simulated relationship between the C/N ratio and active JNK at steady state.

```
In [28]: st = []
         for i in np.arange(0, 1.4, 0.05):
             st.append(estim_steady_state_kinase(i, param_set) * 1000)  # in nM
         plt.semilogx(st, np.arange(0, 1.4, 0.05))
         plt.xlabel('active JNK (nM)')
         plt.ylabel('C/N Ratio')
```

```
Out[28]: <matplotlib.text.Text at 0x7f154e9ffb50>
```



You can see that the simulation gets unstable when the amount of active JNK is above 1000 nM. Presumably, the model cannot extrapolate past the upper bound of the training data, so we will constrain the upper limit of active JNK to this value for later simulations.

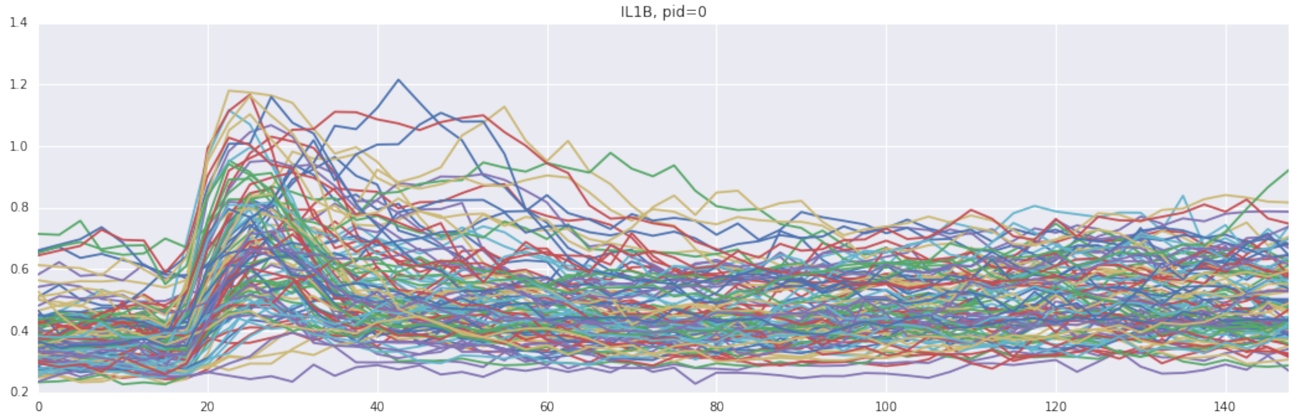
```
In [29]: KIN_MAX = 1.0  # 1uM
```

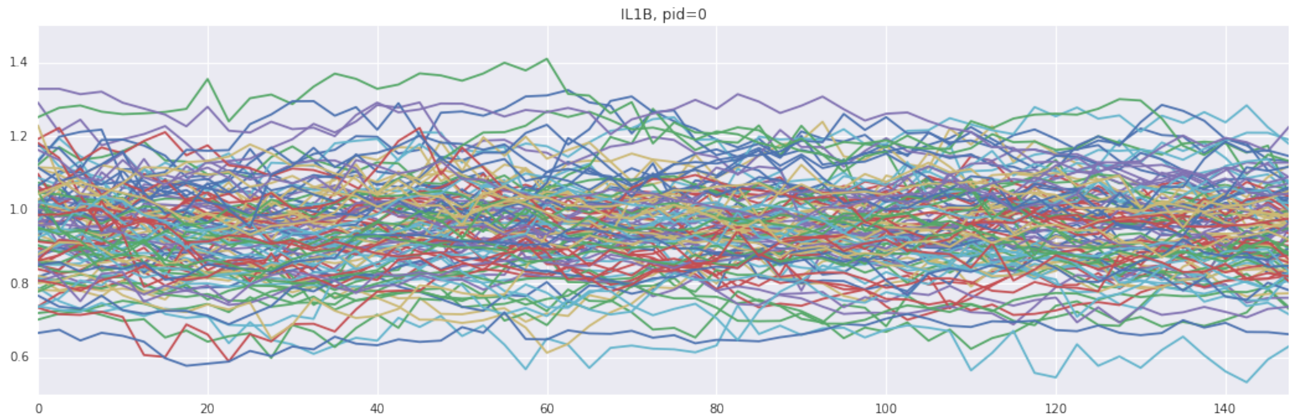
Now, let's calculate the time-series of JNK stimulated with IL-1β.

```
In [30]: parent_folder = join(data_folder, 'IL1B')
         sub_folders = ['Pos005', 'Pos006', 'Pos007', 'Pos008']
         conditions = ['IL1B', 'IL1B', 'IL1B', 'IL1B']
         sites_il1b = Sites(parent_folder, sub_folders, conditions, file_name='df_cleaned.npz')
         sites_il1b.merge_conditions()
```

```
In [31]: ops_plotter.plot_all(sites_il1b['cyto', 'TRITC', 'median_ratio']);
         ops_plotter.plot_all(sites_il1b['cyto', 'YFP', 'median_ratio']);
```


Nature Protocols: doi:10.1038/nprot.2017.128





In this experiment, cells express the JNK KTR AE mutant (right) in addition to the JNK KTR (left). The variability observed in the JNK KTR AE mutant is a reflection of the noise in the endogenous export and import rates. We can use these data to correct for the variability in import and export in individual cells.

For each cell, we calculate the average C/N ratio over time and then divide it by the average population C/N ratio. This value is q, the noise factor in export/import.
We correct the mean value of the estimated k_e and k_i using q for each cell.

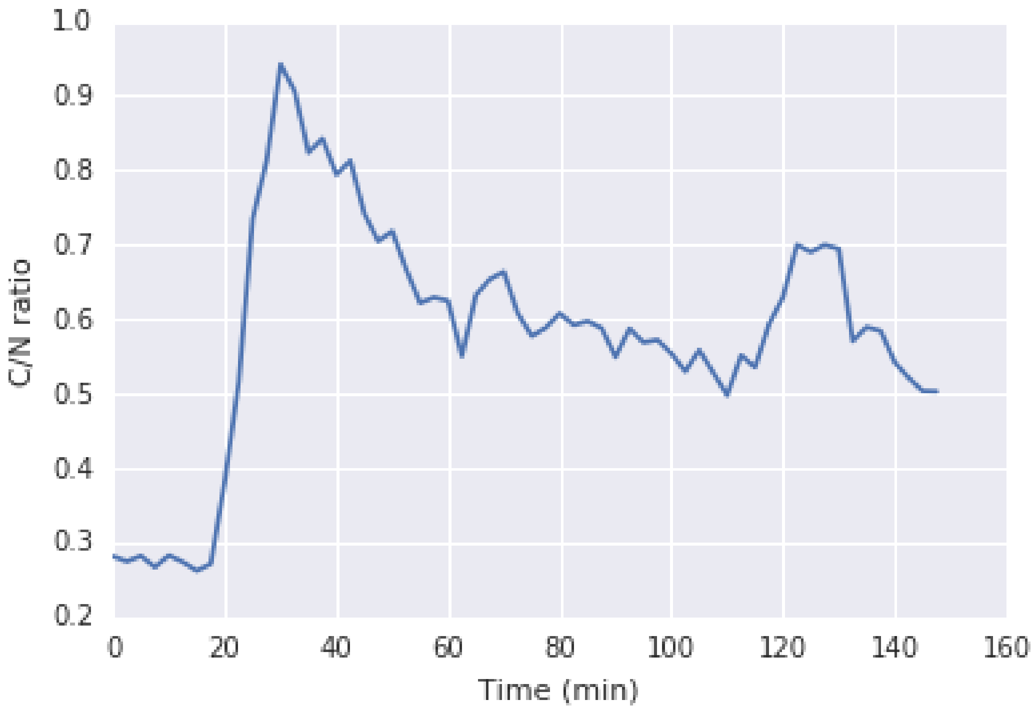
```
In [32]: site_il1b = sites_il1b.Pos005
         mean_yfp = np.nanmean(site_il1b['cyto', 'YFP', 'median_ratio'], axis=0)
         q_store = []
         for cell in site_il1b['cyto', 'YFP', 'median_ratio']:
             q_store.append(np.mean(cell/mean_yfp))

         k_iu_sc, k_ip_sc, k_eu_sc, k_ep_sc = [], [], [], []
         for q in q_store:
             k_iu_sc.append(np.nanmean(est_k_iu)/q)
             k_ip_sc.append(np.nanmean(est_k_ip)/q)
             k_eu_sc.append(np.nanmean(est_k_eu) * q)
             k_ep_sc.append(np.nanmean(est_k_ep) * q)
```

Now let's start estimation with a single cell.

```
In [33]: cell_id = 0  # randomly pick one cell
         time = np.arange(0, 150, 2.5)  # in minute
         single_rcn = np.array(site_il1b['cyto', 'TRITC', 'median_ratio'][cell_id, :])
         plt.plot(time, single_rcn)
         plt.ylabel('C/N ratio')
         plt.xlabel('Time (min)')
```


Nature Protocols: doi:10.1038/nprot.2017.128

Out[33]: <matplotlib.text.Text at 0x7f154d909210>



This is the experimental trace that we are going to fit.

First, to reduce the complexity of parameter estimation, we assume that time-course of active JNK, as well as C/N ratio with IL1-β treatment, can be modeled using the trapezoid function below.
This part might not be necessary but reduces the calculation cost.

We write a function, *trapezoid_func*, that stores the linear equation describing each portion of the trapezoid and the times where each section begins and ends.

```
In [34]: def trapezoid_func(t, ct1, ct2, ct3, ct4, c1, c2, c3):
             """Trapezoid function.
             ct1 = t1
             ct2 = t1 + t2
             ct3 = t1 + t2 + t3
             ct4 = t1 + t2 + t3 + t4

             """
             if t <= ct1:
                 return c1
             elif (t > ct1) and (t <= ct2):
                 return c1 + (c2 - c1) * (t-ct1)/(ct2-ct1)
             elif (t > ct2) and (t <= ct3):
                 return c2
             elif (t > ct3) and (t <= ct4):
                 return c2 - (c2 - c3) * (t-ct3)/(ct4-ct3)
             elif t > ct4:
                 return c3
```

Since we assume the data can be fit with the trapezoid function, we would like to capture the dynamics of first peak (mostly ~50 min) in particular.
Thus, we put some emphasis on the fitting results before 50 min in the following cost function (*trapezoid_err*). This makes fitting to noisy data more robust. The cost function is still the squared distance to the experimental values.

Nature Protocols: doi:10.1038/nprot.2017.128

```
In [35]: def trapezoid_err(params, t, y, w=5, tw=50):
             """cost function for trapezoid fitting.
             w is picked ad hoc."""
             weights = np.ones(len(t))
             weights[t <= tw] = w
             y_p = np.zeros(y.shape)
             for num, ti in enumerate(t):
                 y_p[num] = trapezoid_func(np.float(ti)/t.max(), *params)
             return ((y - y_p)**2 * weights).sum()
```

We also write a function (*fit_trapezoid*) that will fit whatever input data is given to the trapezoidal function. The function then returns the time points for each of the four stages (T_1, T_2, T_3, T_4), as well as the values at each intersection (C_1, C_2, C_3). A set of constraints stored in the variable *cons* ensures the fit produced is trapezoidal.

```
In [36]: def fit_trapezoid(t, y, p0=None, tbuf=[0.05, 0.05, 0.05]):
             if p0 is None:
                 p0 = [0.2, 0.4, 0.6, 0.8, y[0], y.max(), y[-1]]
             # This is the constraints for trapezoid parameters.
             cons = ({'type': 'ineq', 'fun': lambda x:  x[1] - x[0] - tbuf[0]},  # t1 < t2
                     {'type': 'ineq', 'fun': lambda x:  x[2] - x[1] - tbuf[1]},  # t2 < t3
                     {'type': 'ineq', 'fun': lambda x:  x[3] - x[2] - tbuf[2]},  # t3 < t4
                     {'type': 'ineq', 'fun': lambda x:  x[5] - x[4]},  # c1 < c2
                     {'type': 'ineq', 'fun': lambda x:  x[5] - x[6]},  # c3 < c2
                     {'type': 'ineq', 'fun': lambda x:  x})  # non-negative parameters
             bnds = ((0, 1), ) * 4 + ((y.min(), y.max()), ) * 3
             fun = partial(trapezoid_err, t=t, y=y)
             res = minimize(fun, p0, constraints=cons, bounds=bnds, tol=1e-12, options=dict(max_iter=10000))
             # convert Ts from relative to absolute time.
             Ts = np.interp(res.x[:4], np.linspace(0, 1, len(t)), t)
             return np.concatenate((Ts, res.x[4:]))
```

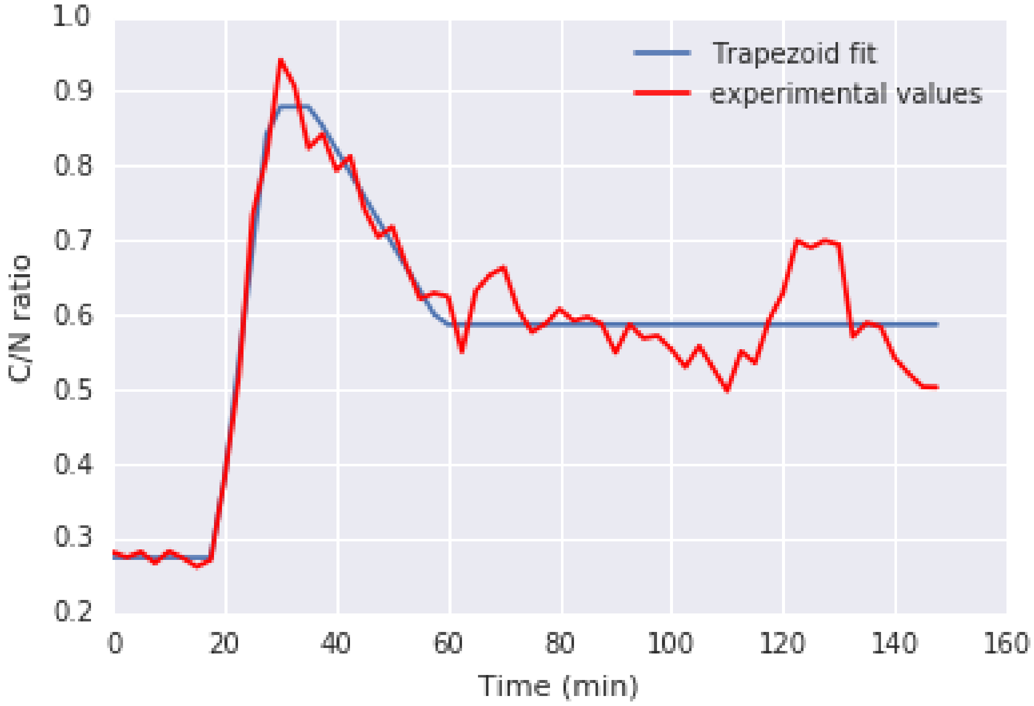
To determine the values for $T_1, \ldots, T_4$, we first fit the time-series of the C/N ratio with the trapezoid function.

```
In [37]: ct1, ct2, ct3, ct4, c1, c2, c3 = fit_trapezoid(time, single_rcn)
         kin_func = lambda t: np.interp(t, [ct1, ct2, ct3, ct4, time[-1]], [c1, c2, c2, c3, c3])
         plt.plot(time, kin_func(time))
         plt.hold(True)
         plt.plot(time, single_rcn, 'r')
         plt.legend(['Trapezoid fit', 'experimental values'])
         plt.ylabel('C/N ratio')
         plt.xlabel('Time (min)')
```

```
/opt/conda/lib/python2.7/site-packages/ipykernel_launcher.py:13: OptimizeWarning: Unknown solver options: max_iter
  del sys.path[0]
```

```
Out[37]: <matplotlib.text.Text at 0x7f154de78410>
```


Nature Protocols: doi:10.1038/nprot.2017.128



The fitting worked well. The *fit_trapezoid* function returns cumulative values (e.g. $C_{T_2} = C_{T_1} + C_{T_2}$), so now we write functions that convert the cumulative values to absolute values (*normt1_to_ct1* and *ct1_to_normt1*). Note that $T_1, \ldots, T_4$ are normalized by the last time point to have a scale from 0 to 1.

```
In [38]: def normt1_to_ct1(rt1, rt2, rt3, rt4, time):
             """convert normalized t1 to cumulative time."""
             ct1 = rt1 * time[-1]
             ct2 = (rt1 + rt2) * time[-1]
             ct3 = (rt1 + rt2 + rt3) * time[-1]
             ct4 = (rt1 + rt2 + rt3 + rt4) * time[-1]
             return ct1, ct2, ct3, ct4

         def ct1_to_normt1(ct1, ct2, ct3, ct4, time):
             """convert cumulative time to normalized t1."""
             t1 = ct1/time[-1]
             t2 = (ct2 - ct1)/time[-1]
             t3 = (ct3 - ct2)/time[-1]
             t4 = (ct4 - ct3)/time[-1]
             return t1, t2, t3, t4

         ini_t1, ini_t2, ini_t3, ini_t4 = ct1_to_normt1(ct1, ct2, ct3, ct4, time)  # initial values
         print 'The initial values for T1, T2, T3 and T4 are {0:0.1f}, {1:0.1f}, {2:0.1f} and {3:0.1f} min.'.format(ini_t1

The initial values for T1, T2, T3 and T4 are 18.2, 9.9, 7.4 and 23.1 min.
```

Now that we have our time values estimated, we can try fitting our ODE model (*main_ode*) from before. The function *kinase_dyanmics_ode* fits the reporter profile to the trapezoidal function. The function *rcn_dynamics_ode* uses these values to generate the C/N ratio.

```
In [39]: def kinase_dynamics_ode(kins, time, pset, k1):
             """receives kinase concentration corresponding to C1, C2 and C3 and run a simulation based on those."""
             k2, k3, t1, t2, t3, t4 = kins
             # get model to steady state
             rep0 = calc_rep_profile_at_steady_state(k1, pset)
```


Nature Protocols: doi:10.1038/nprot.2017.128

```
        ct1, ct2, ct3, ct4 = normt1_to_ct1(t1, t2, t3, t4, time)
        t_kins = [ct1, ct2, ct3, ct4, time[-1]]

        pset['kin_c_func'] = lambda t: np.interp(t, t_kins, [k1, k2, k2, k3, k3])
        pset['kin_n_func'] = lambda t: np.interp(t, t_kins, [k1, k2, k2, k3, k3])
        ts = odeint(main_ode, rep0, time, (pset, ))
        return ts

    def rcn_dynamics_ode(kins, time, pset, k1):
        rep = kinase_dynamics_ode(kins, time, pset, k1)
        return (rep[:, 0] + rep[:, 2])/(rep[:, 1] + rep[:, 3])
```

Set the corrected import and export rate constants for this cell to the param_set.

```
In [40]: param_set['k_iu'] = k_iu_sc[cell_id]
         param_set['k_ip'] = k_ip_sc[cell_id]
         param_set['k_eu'] = k_eu_sc[cell_id]
         param_set['k_ep'] = k_ep_sc[cell_id]
```

We assume that the initial and final C/N ratio are pseudo-steady state and we use them to estimate the initial and final kinase concentration.

```
In [41]: kin_ini = estim_steady_state_kinase(single_rcn[0], param_set, bnd_max=KIN_MAX)
         kin_fin = estim_steady_state_kinase(c3, param_set, bnd_max=KIN_MAX)
```

Again, we define a cost function which minimizes the difference between the experimental time-series and the simulated time-series.

```
In [42]: func = lambda x: ((rcn_dynamics_ode(x, time, param_set, kin_ini) - single_rcn)**2).sum()
```

Recall that we derived initial values for $T_1, \ldots, T_4$ by fitting the trapezoid function to the C/N ratio data. However, kinase activation precedes any change in the C/N ratio, so to find the actual $T_1, \ldots, T_4$ values, we use *minimize* to optimize the *rcn_dynamics_ode* function, with $T_1, \ldots, T_4$ bounded by the values from the C/N data.

```
In [43]: bnds = ((0, 1), (0, 1), (0, ini_t1), (0, ini_t2), (0, ini_t3), (0, ini_t4))
         ret = minimize(func, x0=(kin_fin*2, kin_fin, ini_t1, ini_t2, ini_t3, ini_t4), bounds=bnds, method='L-BFGS-B')
```

We can now plot our estimated kinase concentrations and the simulated C/N ratios that would result from those concentrations and see that they closely match the measured C/N ratios.

```
In [44]: fig = plt.figure(figsize=(10, 5))
         ax1 = fig.add_subplot(1,2,1)
         h1 = plt.plot(time, rcn_dynamics_ode(ret.x, time, param_set, kin_ini))
         plt.hold(True)
         h2 = plt.plot(time, single_rcn, 'r')
         ax1.set_ylabel('C/N ratio')
         ax1.legend(h1, ['simulated values'])

         ax2 = fig.add_subplot(1,2,2)
         ct1, ct2, ct3, ct4 = normt1_to_ct1(ret.x[2], ret.x[3], ret.x[4], ret.x[5], time)
         kin_ts = np.interp(time, [ct1, ct2, ct3, ct4, time[-1]], [kin_ini, ret.x[0], ret.x[0], ret.x[1], ret.x[1]])
         h3 = plt.plot(time, kin_ts)
         ax2.set_ylabel('Active kinase concentration (uM)')
         ax2.legend(h3, ['simulated values'])

Out[44]: <matplotlib.legend.Legend at 0x7f154ddba750>
```


Nature Protocols: doi:10.1038/nprot.2017.128

Now, we compile all of the steps above into a function (*estim_kin_ts*) so that we can easily apply this process to estimate the dynamic kinase concentration for many cells.

```
In [45]: # Put the steps above into this.
         def estim_kin_ts(time, single_rcn, pset, k_iu, k_ip, k_eu, k_ep):
             ct1, ct2, ct3, ct4, c1, c2, c3 = fit_trapezoid(time, single_rcn)
             pset['k_iu'] = k_iu
             pset['k_ip'] = k_ip
             pset['k_eu'] = k_eu
             pset['k_ep'] = k_ep
             kin_ini = estim_steady_state_kinase(c1, pset, bnd_max=KIN_MAX)
             kin_fin = estim_steady_state_kinase(c3, pset, bnd_max=KIN_MAX)

             func = lambda x: ((rcn_dynamics_ode(x, time, param_set, kin_ini) - single_rcn)**2).sum()
             bnds = ((0, 1), (0, 1), (0, ini_t1), (0, ini_t2), (0, ini_t3), (0, ini_t4))
             ret = minimize(func, x0=(kin_fin*3, kin_fin, ini_t1, ini_t2, ini_t3, ini_t4), bounds=bnds, method='L-BFGS-B',

             estim_rcn = rcn_dynamics_ode(ret.x, time, param_set, kin_ini)
             ct1, ct2, ct3, ct4 = normt1_to_ct1(ret.x[2], ret.x[3], ret.x[4], ret.x[5], time)
             estim_kin = np.interp(time, [ct1, ct2, ct3, ct4, time[-1]], [kin_ini, ret.x[0], ret.x[0], ret.x[1], ret.x[1]])
             return estim_rcn, estim_kin
```

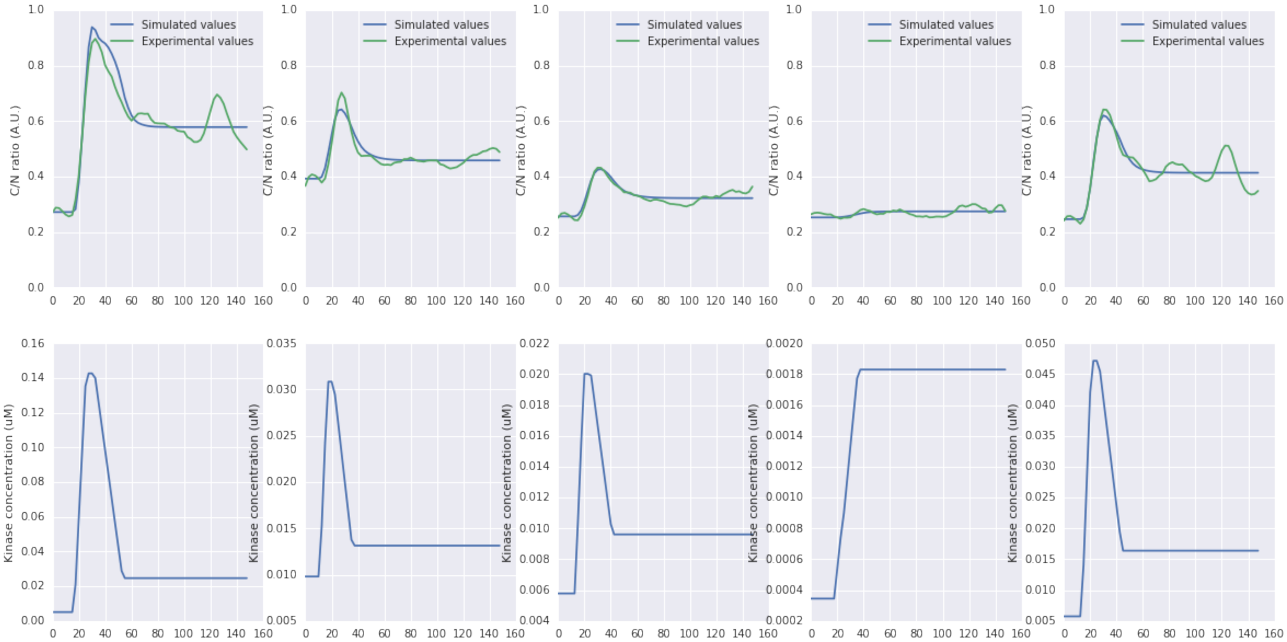
We simulate for five cells to see that our function works well and that the predictions closely match the experimental data.

```
In [46]: plt.figure(figsize=(20, 10))
         median_ratio = np.array(site_il1b['cyto', 'TRITC', 'median_ratio'])
         median_ratio = savgol_filter(median_ratio, window_length=9, polyorder=3, axis=1)  # smoothing
         cell_ids = range(5)
         for num, cell_id in enumerate(cell_ids):
             single_rcn = median_ratio[cell_id]
             print '{0} estimating...'.format(cell_id)
             estim_rcn, estim_kin = estim_kin_ts(time, single_rcn, param_set,  k_iu_sc[cell_id], k_ip_sc[cell_id], k_eu_sc[
             ax1 = plt.subplot(2, 5, num+1)
             h1 = plt.plot(time, estim_rcn, time, single_rcn)
             ax1.set_ylabel('C/N ratio (A.U.)')
             ax1.set_ylim([0, 1])
             ax1.legend(h1, ['Simulated values', 'Experimental values'])
```


Nature Protocols: doi:10.1038/nprot.2017.128

```
        ax2 = plt.subplot(2, 5, num+1+5)
        plt.plot(time, estim_kin)
        ax2.set_ylabel('Kinase concentration (uM)')
```

```
0 estimating...


/opt/conda/lib/python2.7/site-packages/ipykernel_launcher.py:13: OptimizeWarning: Unknown solver options: max_iter
  del sys.path[0]


1 estimating...
2 estimating...
3 estimating...
4 estimating...
```



Finally, we can run the simulation for all of the cells. Note that this process may take a while (~30 minutes).

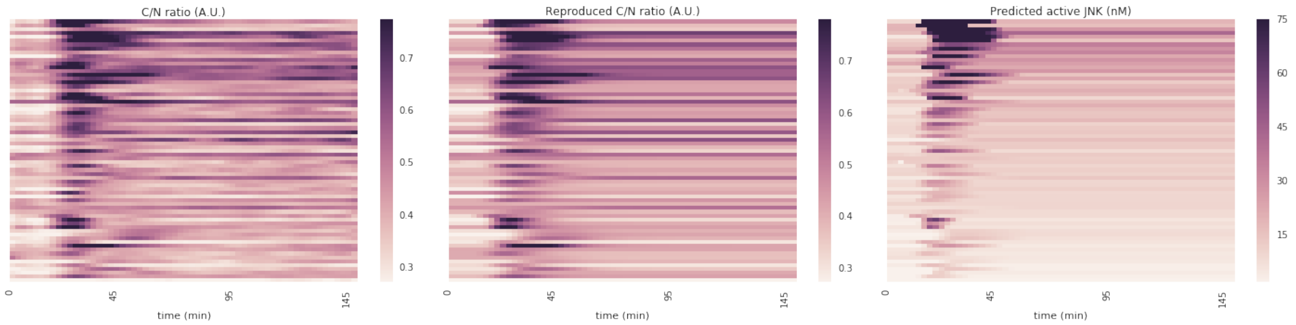
```
In [47]: # Store estimated C/N ratio and estimated active JNK in store
         store = []
         cell_ids = range(median_ratio.shape[0])
         for num, cell_id in enumerate(cell_ids):
             single_rcn = median_ratio[cell_id]
         #     print '{0} estimating...'.format(cell_id)
             estim_rcn, estim_kin = estim_kin_ts(time, single_rcn, param_set,  k_iu_sc[cell_id], k_ip_sc[cell_id], k_eu_sc
             store.append((single_rcn, estim_rcn, estim_kin))
```

```
/opt/conda/lib/python2.7/site-packages/ipykernel_launcher.py:13: OptimizeWarning: Unknown solver options: max_iter
  del sys.path[0]
```

```
In [48]: # get rid of those that did not fit well.
         cleaned_data = [i for i in store if (np.abs(i[1] - i[0])).sum() < 2.5]

In [49]: idx = np.argsort(-np.sum([i[2] for i in cleaned_data], axis=1))
         fig, axes = plt.subplots(1, 3, figsize=(20, 5))
         axes.tolist()
         sns.heatmap([cleaned_data[i][0] for i in idx], robust=True, ax=axes[0])
```


Nature Protocols: doi:10.1038/nprot.2017.128

```
sns.heatmap([cleaned_data[i][1] for i in idx], robust=True, ax=axes[1])
sns.heatmap([cleaned_data[i][2]*1000 for i in idx], robust=True, ax=axes[2])
[ax.set_title(i) for ax, i in zip(axes, ('C/N ratio (A.U.)', 'Reproduced C/N ratio (A.U.)', 'Predicted active JNK
[ax.set_yticklabels("") for ax in axes];
[ax.set_xticks([0, 18, 38, 58]) for ax in axes];
[ax.set_xticklabels([int(time[0]), int(time[18]), int(time[38]), int(time[58])]) for ax in axes];
[ax.set_xlabel("time (min)") for ax in axes];
fig.tight_layout()
fig.savefig('heatmap.svg', format='svg', dpi=1200)
```



Nature Protocols: doi:10.1038/nprot.2017.128

Supplementary Data
pLenti PGK Promoter JNK KTR–mClover2
The same sequences are available as Addgene #59151.

```
LOCUS       pLenti_PGK_JNKKTRClover 8848 bp ds-DNA     circular     10-MAR-
2017
DEFINITION  .
FEATURES             Location/Qualifiers
     misc_feature    4776..4958
                     /label="JNK KTR"
     rep_origin      1087..1769
                     /label="ColE1 origin"
     primer_bind     2137..2157
                     /label="M13-rev"
     promoter        6323..6835
                     /label="mouse Pgk promoter"
     misc_feature    4965..5648
                     /label="Clover"
     misc_binding    2109..2131
                     /label="LacO"
     misc_feature    4023..4038
                     /label="cPPT"
     misc_recomb     complement(5685..5699)
                     /label="AttR2"
     LTR             7642..7822
                     /label="HIV LTR 2"
     CDS             8318..8386
                     /label="LacZ alpha"
     CDS             330..989
                     /label="AmpR"
     primer_bind     complement(8230..8247)
                     /label="M13-fwd"
     LTR             2448..2628
                     /label="HIV LTR 1"
     misc_recomb     4754..4770
                     /label="AttL1"
     rep_origin      complement(8396..8836)
                     /label="F1 ori"
     misc_recomb     complement(5675..5684)
                     /label="AttL2"
     misc_feature    3290..3523
                     /label="RRE"
     misc_feature    7572..7587
                     /label="cPPT"
     rep_origin      8391..8846
                     /label="M13 origin"
     primer_bind     complement(7896..7913)
                     /label="SG53"
     3'UTR           5732..6307
                     /label="WPRE"
     primer_bind     2175..2194
                     /label="T3"
     misc_recomb     4746..4753
                     /label="AttR1"
     promoter        4179..4694
                     /label="human PGK promoter"
```

```
     CDS             6852..7451
                     /label="Puro cds"
ORIGIN
        1 caggtggcac ttttcgggga aatgtgcgcg gaacccctat ttgtttattt ttctaaatac
       61 attcaaatat gtatccgctc atgagacaat aaccctgata aatgcttcaa taatattgaa
      121 aaaggaagag tatgagtatt caacatttcc gtgtcgccct tattcccttt tttgcggcat
      181 tttgccttcc tgtttttgct cacccagaaa cgctggtgaa agtaaaagat gctgaagatc
      241 agttgggtgc acgagtgggt tacatcgaac tggatctcaa cagcggtaag atccttgaga
      301 gttttcgccc cgaagaacgt tttccaatga tgagcacttt taaagttctg ctatgtggcg
      361 cggtattatc ccgtattgac gccgggcaag agcaactcgg tcgccgcata cactattctc
      421 agaatgactt ggttgagtac tcaccagtca cagaaaagca tcttacggat ggcatgacag
      481 taagagaatt atgcagtgct gccataacca tgagtgataa cactgcggcc aacttacttc
      541 tgacaacgat cggaggaccg aaggagctaa ccgctttttt gcacaacatg ggggatcatg
      601 taactcgcct tgatcgttgg gaaccggagc tgaatgaagc cataccaaac gacgagcgtg
      661 acaccacgat gcctgtagca atggcaacaa cgttgcgcaa actattaact ggcgaactac
      721 ttactctagc ttcccggcaa caattaatag actggatgga ggcggataaa gttgcaggac
      781 cacttctgcg ctcggccctt ccggctggct ggtttattgc tgataaatct ggagccggtg
      841 agcgtgggtc tcgcggtatc attgcagcac tggggccaga tggtaagccc tcccgtatcg
      901 tagttatcta cacgacgggg agtcaggcaa ctatggatga acgaaataga cagatcgctg
      961 agataggtgc ctcactgatt aagcattggt aactgtcaga ccaagtttac tcatatatac
     1021 tttagattga tttaaaactt catttttaat ttaaaaggat ctaggtgaag atcctttttg
     1081 ataatctcat gaccaaaatc ccttaacgtg agttttcgtt ccactgagcg tcagaccccg
     1141 tagaaaagat caaaggatct tcttgagatc ctttttttct gcgcgtaatc tgctgcttgc
     1201 aaacaaaaaa accaccgcta ccagcggtgg tttgtttgcc ggatcaagag ctaccaactc
     1261 tttttccgaa ggtaactggc ttcagcagag cgcagatacc aaatactgtc cttctagtgt
     1321 agccgtagtt aggccaccac ttcaagaact ctgtagcacc gcctacatac ctcgctctgc
     1381 taatcctgtt accagtggct gctgccagtg gcgataagtc gtgtcttacc gggttggact
     1441 caagacgata gttaccggat aaggcgcagc ggtcgggctg aacggggggt tcgtgcacac
     1501 agcccagctt ggagcgaacg acctacaccg aactgagata cctacagcgt gagctatgag
     1561 aaagcgccac gcttcccgaa gggagaaagg cggacaggta tccggtaagc ggcagggtcg
     1621 gaacaggaga gcgcacgagg gagcttccag ggggaaacgc ctggtatctt tatagtcctg
     1681 tcgggtttcg ccacctctga cttgagcgtc gatttttgtg atgctcgtca ggggggcgga
     1741 gcctatggaa aaacgccagc aacgcggcct ttttacggtt cctggccttt tgctggcctt
     1801 ttgctcacat gttctttcct gcgttatccc ctgattctgt ggataaccgt attaccgcct
     1861 ttgagtgagc tgataccgct cgccgcagcc gaacgaccga gcgcagcgag tcagtgagcg
     1921 aggaagcgga agagcgccca atacgcaaac cgcctctccc cgcgcgttgg ccgattcatt
     1981 aatgcagctg gcacgacagg tttcccgact ggaaagcggg cagtgagcgc aacgcaatta
     2041 atgtgagtta gctcactcat taggcacccc aggctttaca ctttatgctt ccggctcgta
     2101 tgttgtgtgg aattgtgagc ggataacaat ttcacacagg aaacagctat gaccatgatt
     2161 acgccaagcg cgcaattaac cctcactaaa gggaacaaaa gctggagctg caagcttaat
     2221 gtagtcttat gcaatactct tgtagtcttg caacatggta acgatgagtt agcaacatgc
     2281 cttacaagga gagaaaaagc accgtgcatg ccgattggtg gaagtaaggt ggtacgatcg
     2341 tgccttatta ggaaggcaac agacgggtct gacatggatt ggacgaacca ctgaattgcc
     2401 gcattgcaga gatattgtat ttaagtgcct agctcgatac aataaacggg tctctctggt
     2461 tagaccagat ctgagcctgg gagctctctg gctaactagg gaacccactg cttaagcctc
     2521 aataaagctt gccttgagtg cttcaagtag tgtgtgcccg tctgttgtgt gactctggta
     2581 actagagatc cctcagaccc ttttagtcag tgtggaaaat ctctagcagt ggcgcccgaa
     2641 cagggacctg aaagcgaaag ggaaaccaga gctctctcga cgcaggactc ggcttgctga
     2701 agcgcgcacg gcaagaggcg aggggcggcg actggtgagt acgccaaaaa ttttgactag
     2761 cggaggctag aaggagagag atgggtgcga gagcgtcagt attaagcggg ggagaattag
     2821 atcgcgatgg gaaaaaattc ggttaaggcc agggggaaag aaaaaatata aattaaaaca
     2881 tatagtatgg gcaagcaggg agctagaacg attcgcagtt aatcctggcc tgttagaaac
     2941 atcagaaggc tgtagacaaa tactgggaca gctacaacca tcccttcaga caggatcaga
     3001 agaacttaga tcattatata atacagtagc aaccctctat tgtgtgcatc aaaggataga
     3061 gataaaagac accaaggaag ctttagacaa gatagaggaa gagcaaaaca aaagtaagac
     3121 caccgcacag caagcggccg ctgatcttca gacctggagg aggagatatg agggacaatt
     3181 ggagaagtga attatataaa tataaagtag taaaaattga accattagga gtagcaccca
```

```
3241 ccaaggcaaa gagaagagtg gtgcagagag aaaaaagagc agtgggaata ggagctttgt
3301 tccttgggtt cttgggagca gcaggaagca ctatgggcgc agcctcaatg acgctgacgg
3361 tacaggccag acaattattg tctggtatag tgcagcagca gaacaatttg ctgagggcta
3421 ttgaggcgca acagcatctg ttgcaactca cagtctgggg catcaagcag ctccaggcaa
3481 gaatcctggc tgtggaaaga tacctaaagg atcaacagct cctggggatt tggggttgct
3541 ctggaaaact catttgcacc actgctgtgc cttggaatgc tagttggagt aataaatctc
3601 tggaacagat ttggaatcac acgacctgga tggagtggga cagagaaatt aacaattaca
3661 caagcttaat acactcctta attgaagaat cgcaaaacca gcaagaaaag aatgaacaag
3721 aattattgga attagataaa tgggcaagtt tgtggaattg gtttaacata acaaattggc
3781 tgtggtatat aaaattattc ataatgatag taggaggctt ggtaggttta agaatagttt
3841 ttgctgtact ttctatagtg aatagagtta ggcagggata ttcaccatta tcgtttcaga
3901 cccacctccc aaccccgagg ggacccgaca ggcccgaagg aatagaagaa gaaggtggag
3961 agagagacag agacagatcc attcgattag tgaacggatc tcgacggtat cggttaactt
4021 ttaaaagaaa aggggggatt ggggggtaca gtgcagggga aagaatagta gacataatag
4081 caacagacat acaaactaaa gaattacaaa aacaaattac aaaaattcaa aattttatcg
4141 atcacgagac tagcctcgag aagcttgata tcgaattccc acggggttgg ggttgcgcct
4201 tttccaaggc agccctgggt ttgcgcaggg acgcggctgc tctgggcgtg gttccgggaa
4261 acgcagcggc gccgaccctg ggtctcgcac attcttcacg tccgttcgca gcgtcacccg
4321 gatcttcgcc gctacccttg tgggcccccc ggcgacgctt cctgctccgc ccctaagtcg
4381 ggaaggttcc ttgcggttcg cggcgtgccg gacgtgacaa acggaagccg cacgtctcac
4441 tagtaccctc gcagacggac agcgccaggg agcaatggca gcgcgccgac cgcgatgggc
4501 tgtggccaat agcggctgct cagcggggcg cgccgagagc agcggccggg aaggggcggt
4561 gcgggaggcg gggtgtgggg cggtagtgtg ggccctgttc ctgcccgcgc ggtgttccgc
4621 attctgcaag cctccggagc gcacgtcggc agtcggctcc ctcgttgacc gaatcaccga
4681 cctctctccc caggggggatc accggTCGTC GACTAGTCCA GTGTGGTGGA ATTCTGCAGA
4741 TATCAACAAG TTTGTACAAA AAAGCAGGCT CCACCATGAG TAACCCTAAG ATCCTAAAAC
4801 AGAGCATGAC CTTGAACCTG GCCGACCCGG TGGGCAGTCT GAAGCCGCAC CTCCGCGCCA
4861 AGCGCTCGGC CGGTCTCAAC GAAGACGAGT CGCCATCTAA GAAGCTGGCG TCGCCGGAGG
4921 TGTCGTCAAG GCTGGAGCGC CTGACCCTCC AGTCCAGCac tagtatggtg agcaagggcg
4981 aggagctgtt caccggggtg gtgcccatcc tggtcgagct ggacggcgac gtaaacggcc
5041 acaagttcag cgtCCGCggc gagggcgagg gcgatgccac cAACggcaag ctgaccctga
5101 agttcatctg caccaccggc aagctgcccg tgccctggcc caccctcgtg accaccttcg
5161 gctacggcgt ggcctgcttc agccgctacc ccgaccacat gaagcagcac gacttcttca
5221 agtccgccat gcccgaaggc tacgtccagg agcgcaccat ctctttcaag gacgacggTA
5281 CCtacaagac ccgcgccgag gtgaagttcg agggcgacac cctggtgaac cgcatcgagc
5341 tgaagggcat cgacttcaag gaggacggca acatcctggg gcacaagctg gagtacaact
5401 tcaacagcca caacgtctat atcacggccg acaagcagaa gaacggcatc aaggctaact
5461 tcaagatccg ccacaacGTT gaggacggca gcgtgcagct cgccgaccac taccagcaga
5521 acacccccat cggcgacggc cccgtgctgc tgcccgacaa ccactacctg agcCATcagt
5581 ccgccctgag caaagacccc aacgagaagc gcgatcacat ggtcctgctg gagttcgtga
5641 ccgccgccta aGCGGCCGCA CTCGAGATAT CTAGACCCAG CTTTCTTGTA CAAAGTGGTT
5701 GATATCCAGC ACAGTGGCGG CCGCTCGAca atcaacctct ggattacaaa atttgtgaaa
5761 gattgactgg tattcttaac tatgttgctc cttttacgct atgtggatac gctgctttaa
5821 tgcctttgta tcatgctatt gcttcccgta tggctttcat tttctcctcc ttgtataaat
5881 cctggttgct gtctctttat gaggagttgt ggcccgttgt caggcaacgt ggcgtggtgt
5941 gcactgtgtt tgctgacgca acccccactg gttggggcat tgccaccacc tgtcagctcc
6001 tttccgggac tttcgctttc cccctcccta ttgccacggc ggaactcatc gccgcctgcc
6061 ttgcccgctg ctggacaggg gctcggctgt tgggcactga caattccgtg gtgttgtcgg
6121 ggaagctgac gtcctttcca tggctgctcg cctgtgttgc cacctggatt ctgcgcggga
6181 cgtccttctg ctacgtccct tcggccctca atccagcgga ccttccttcc cgcggcctgc
6241 tgccggctct gcggcctctt ccgcgtcttc gccttcgccc tcagacgagt cggatctccc
6301 tttgggccgc ctccccgcct gGAATTCTAC CGGGTAGGGG AGGCGCTTTT CCCAAGGCAG
6361 TCTGGAGCAT GCGCTTTAGC AGCCCCGCTG GGCACTTGGC GCTACACAAG TGGCCTCTGG
6421 CCTCGCACAC ATTCCACATC CACCGGTAGG CGCCAACCGG CTCCGTTCTT TGGTGGCCCC
6481 TTCGCGCCAC CTTCTACTCC TCCCCTAGTC AGGAAGTTCC CCCCCGCCCC GCAGCTCGCG
6541 TCGTGCAGGA CGTGACAAAT GGAAGTAGCA CGTCTCACTA GTCTCGTGCA GATGGACAGC
6601 ACCGCTGAGC AATGGAAGCG GGTAGGCCTT TGGGGCAGCG GCCAATAGCA GCTTTGCTCC
```

```
6661 TTCGCTTTCT GGGCTCAGAG GCTGGGAAGG GGTGGGTCCG GGGGCGGGCT CAGGGGCGGG
6721 CTCAGGGGCG GGGCGGGCGC CCGAAGGTCC TCCGGAGGCC CGGCATTCTG CACGCTTCAA
6781 AAGCGCACGT CTGCCGCGCT GTTCTCCTCT TCCTCATCTC CGGGCCTTTC GACCTGCAGC
6841 CCAAGCTTAC CATGACCGAG TACAAGCCCA CGGTGCGCCT CGCCACCCGC GACGACGTCC
6901 CCAGGGCCGT ACGCACCCTC GCCGCCGCGT TCGCCGACTA CCCCGCCACG CGCCACACCG
6961 TCGATCCGGA CCGCCACATC GAGCGGGTCA CCGAGCTGCA AGAACTCTTC CTCACGCGCG
7021 TCGGGCTCGA CATCGGCAAG GTGTGGGTCG CGGACGACGG CGCCGCGGTG GCGGTCTGGA
7081 CCACGCCGGA GAGCGTCGAA GCGGGGGCGG TGTTCGCCGA GATCGGCCCG CGCATGGCCG
7141 AGTTGAGCGG TTCCCGGCTG GCCGCGCAGC AACAGATGGA AGGCCTCCTG GCGCCGCACC
7201 GGCCCAAGGA GCCCGCGTGG TTCCTGGCCA CCGTCGGCGT CTCGCCCGAC CACCAGGGCA
7261 AGGGTCTGGG CAGCGCCGTC GTGCTCCCCG GAGTGGAGGC GGCCGAGCGC GCCGGGGTGC
7321 CCGCCTTCCT GGAGACCTCC GCGCCCCGCA ACCTCCCCTT CTACGAGCGG CTCGGCTTCA
7381 CCGTCACCGC CGACGTCGAG GTGCCCGAAG GACCGCGCAC CTGGTGCATG ACCCGCAAGC
7441 CCGGTGCCTG ACGCCCGCCC CACGACCCGC AGCGCCCGAC CGAAAGGAGC GCACGACCCC
7501 ATGCATCTCG AGGGCCCGGT ACctttaaga ccaatgactt acaaggcagc tgtagatctt
7561 agccactttt taaaagaaaa ggggggactg gaagggctag ctcactccca acgaagacaa
7621 gatctgcttt ttgcttgtac tgggtctctc tggttagacc agatctgagc ctgggagctc
7681 tctggctgcc tagggaaccc actgcttaag cctcaataaa gcttgccttg agtgcttcaa
7741 gtagtgtgtg cccgtctgtt gtgtgactct ggtaactaga gatccctcag acccttttag
7801 tcagtgtgga aaatctctag cagtagtagt tcatgtcatc ttattattca gtatttataa
7861 cttgcaaaga aatgaatatc agagagtgag aggaacttgt ttattgcagc ttataatggt
7921 tacaaataaa gcaatagcat cacaaatttc acaaataaag catttttttc actgcattct
7981 agttgtggtt tgtccaaact catcaatgta tcttatcatg tctggctcta gctatcccgc
8041 ccctaactcc gcccagttcc gcccattctc cgccccatgg ctgactaatt ttttttattt
8101 atgcagaggc cgaggccgcc tcggcctctg agctattcca gaagtagtga ggaggctttt
8161 ttggaggcct aggcttttgc gtcgagacgt acccaattcg ccctatagtg agtcgtatta
8221 cgcgcgctca ctggccgtcg ttttacaacg tcgtgactgg gaaaaccctg gcgttaccca
8281 acttaatcgc cttgcagcac atcccccttt cgccagctgg cgtaatagcg aagaggcccg
8341 caccgatcgc ccttcccaac agttgcgcag cctgaatggc gaatggcgcg acgcgccctg
8401 tagcggcgca ttaagcgcgg cgggtgtggt ggttacgcgc agcgtgaccg ctacacttgc
8461 cagcgcccta gcgcccgctc ctttcgcttt cttcccttcc tttctcgcca cgttcgccgg
8521 ctttccccgt caagctctaa atcgggggct ccctttaggg ttccgattta gtgctttacg
8581 gcacctcgac cccaaaaaac ttgattaggg tgatggttca cgtagtgggc catcgccctg
8641 atagacggtt tttcgccctt tgacgttgga gtccacgttc tttaatagtg gactcttgtt
8701 ccaaactgga acaacactca accctatctc ggtctattct tttgatttat aagggatttt
8761 gccgatttcg gcctattggt taaaaaatga gctgatttaa caaaaattta acgcgaattt
8821 taacaaaata ttaacgttta caatttcc
//
```