

User Guide

Table of Contents

I. System requirements and package information	2
a. Overview	2
b. Software download	2
c. System requirements	2
d. Package contents	2
II. Analysis of thin filament regulation	4
a. Workflow	4
b. Preparing raw titration data for import into MATLAB	4
c. Loading data into MATLAB	5
d. Normalization of technical replicates	5
e. Global fitting and uncertainty estimation	7
f. Verifying that a sufficient number of bootstraps have been performed	9
g. Hypothesis testing	10
h. Classic McKillop and Geeves fitting	13
III. Hypothesis testing for other types of data	17
a. Statistical comparison of the means of two data sets	17
b. Statistical comparison of the medians of two data sets	18
IV. Appendix – Text versions of the code	20
a. Script_normalization_replicate.m	20
b. Script_global_fitting.m	23
c. Script_McKillopGeeves_fitting.m	27
d. Script_calculate_n_bootstraps.m	31
e. Script_hypothesis_testing.m	33
f. Script_mean.m	36
g. Script_median.m	38

I. System requirements and package information

a. Overview

This user guide describes a collection of scripts for analyzing data as described in the following manuscript:

Computational tool to study perturbations in muscle regulation and its application to heart disease. Samantha K. Barrick, Sarah R. Clippinger, Lina Greenberg, Michael J. Greenberg.

b. Software download

The most current version of the software can be downloaded from GitHub at:

https://github.com/GreenbergLab/Thin_Filament_Fitting

c. System requirements

This series of scripts are executable in MATLAB. The software is compatible with at least versions of MATLAB 2017b to 2019a. We have not tested earlier versions.

The scripts require the following MATLAB toolboxes: Global Optimization, Statistics and Machine Learning. The Parallel Computing toolbox is recommended for the purposes of decreasing the time required to perform the fitting, but it is not strictly required to run the scripts.

d. Package contents

The collection of scripts and data include the following:

Script_normalization_replicate.m	This is a script for normalization of technical replicates.
Script_global_fitting.m	This is a script for global fitting of fluorescence titration data and estimation of uncertainties in fitted parameters.
Script_McKillopGeeves_fitting.m	This script performs fitting of the data without global fitting, along with estimation of uncertainties in fitted parameters.
Script_calculate_n_bootstraps.m	This script calculates confidence intervals as a function of the number of bootstrapping simulations. It is used to check whether a sufficient number of bootstraps have been conducted for convergence of the data.
Script_hypothesis_testing.m	This script performs hypothesis testing.
Script_mean.m	This script performs bootstrapping simulations of data to calculate confidence intervals of the mean.
Script_median.m	This script performs bootstrapping simulations of data to calculate confidence intervals of the median.

Sample_data.xlsx

This contains the data used in the paper, as well as sample data for testing the mean and median scripts.

II. Analysis of thin filament regulation

a. Workflow

The basic workflow for the analysis of fluorescence titration data is as follows:

1. Measure the value of the equilibrium constant between the blocked and closed states, K_B , using stopped-flow kinetic techniques (see Supporting Materials of Barrick et al. for details).
2. Perform steady-state fluorescence titrations measuring myosin binding to regulated thin filaments (see Supporting Materials of Barrick et al. for details). The titrations should be carried out at three separate calcium concentrations: low calcium (nocal), saturating calcium (cal), and intermediate calcium (midcal).
3. Normalize the data from each technical replicate (see Section II.d). After normalizing each technical replicate individually, combine these data into a pooled data set.
4. Fit the pooled titration data set and generate synthetic data sets using the bootstrapping algorithm (see Section II.e). The program fits each synthetic data set to determine the optimal values of the fit parameters and calculates confidence intervals. The data are fit by the following equation, giving the fractional change in fluorescence as a function of myosin concentration, $f([m])$:

$$f([m]) = \frac{F_0 - F}{F_0 - F_\infty} = \frac{K_W [m] P^{(nH-1)} (K_T (1 + K_S)^{(nH)} + 1)}{\left(K_T P^{(nH)} + Q^{(nH)} + \frac{1}{K_B} \right) (1 + K_S)^{(nH-1)}} \quad \text{Equation 1}$$

where F is the measured fluorescence; F_0 and F_∞ are the fluorescence in the absence of myosin and at saturating myosin, respectively; $[m]$ is the concentration of myosin; $P = 1 + [m] * K_W (1 + K_S)$; and $Q = 1 + [m] * K_W$.

5. Statistically test for differences between individual parameters obtained from the fitting of two data sets, such as wild-type and mutant proteins (see Section II.g).

b. Preparing raw titration data for import into MATLAB

The steady-state binding of myosin to regulated thin filaments is measured as a function of the concentration of myosin added. When myosin binds to the thin filament, it quenches the fluorescence. Data are collected as raw fluorescence counts, F , as a function of myosin added ($[m]$). These data can be converted to a measurement of the fractional change in fluorescence $f([m])$ using the following:

$$f([m]) = \frac{F_0 - F}{F_0 - F_\infty}$$

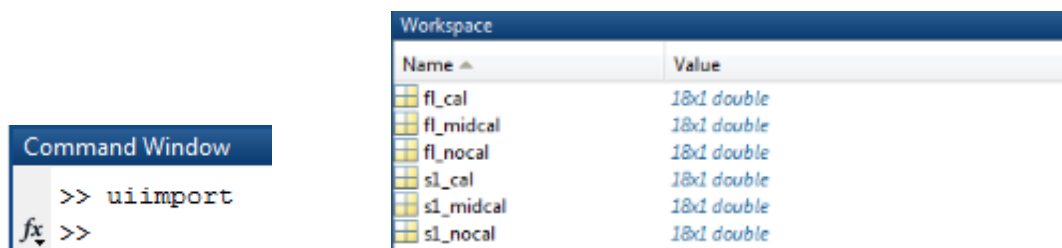
where F_0 is the fluorescence with no myosin added and F_∞ is the fluorescence at the highest concentration of myosin added. This is not properly normalized, so data for each technical replicate should be normalized using the provided MATLAB script (Section II.d).

A table of the fractional change in fluorescence as a function of myosin concentration can be compiled in a spreadsheet application such as Microsoft Excel. We have provided an Excel

spreadsheet containing the data used in the manuscript (sample_data.xls). The raw_WT and raw_dE160 sheets contain the fractional change in fluorescence data for 5 technical replicates of myosin binding to thin filaments regulated by wild-type troponin complex or the Δ E160 troponin T mutant, respectively. The data from a single technical replicate are used as the input for normalization (Section II.d).

c. Loading data into MATLAB

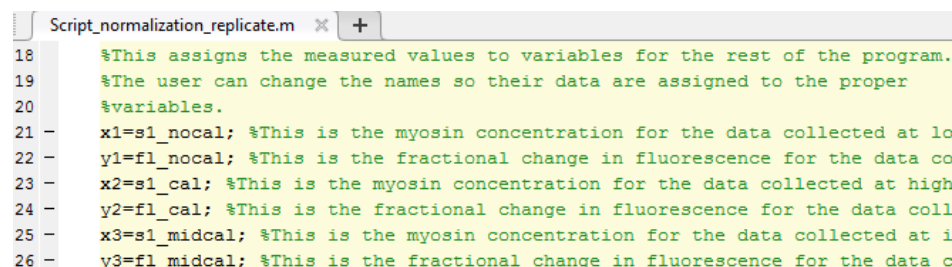
Note this procedure needs to be done for each technical replicate before pooling the data. For each technical replicate, copy the spreadsheet columns containing data. In the MATLAB command window, type “uiimport” and press enter. Select “Clipboard” in the box that pops up. In the Import window, under “Output Type,” select “Column vectors.” If the spreadsheet data contains headers, the column vectors will automatically be assigned names corresponding to the headers; however, if desired, the user can change the names by clicking on the name of each column. Click on the green checkmark above the words “Import Selection” to import your data, which will now be visible in the workspace.



d. Normalization of technical replicates

To normalize the technical replicates, open the script named “Script_normalization_replicate.” This script determines the normalization amplitudes (A, B, C) for the curves acquired at low, high, and intermediate calcium, respectively, and uses these amplitudes to normalize the data. *Note that the code as written assumes that each technical replicate contains a set of three calcium concentrations. If you do not have all three concentrations for each technical replicate, you will need to modify the code.*

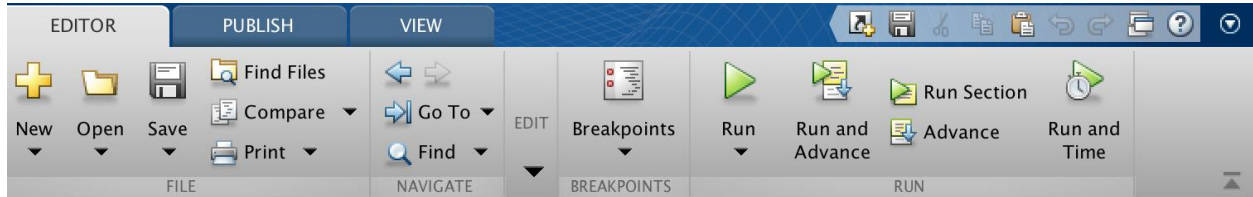
In the code, assign the imported column vectors to variables (x1, y1, x2, ...) to be used in the fitting routine. If necessary, change the right side of each equation to match the names of the imported variables.



Assign appropriate values to the equilibrium constants that will be fixed during the fitting. KBnocal should be changed to the value obtained from the stopped-flow experiments.

```
Script_normalization_replicate.m x +
28 %This assigns the variables appropriate values.
29 - KBnocal=0.290; %User should change this to their value for KB from the stop
30 - KS=18; %This value is fixed as described in McKillop and Geeves (1993).
31 - KBcal=20; %This value is fixed as described in McKillop and Geeves (1993).
```

Run the script by clicking the button labeled “Run and Advance.”



The output, which is saved in the workspace, consists of the following:

- 1) a table named “norm_constants” that contains the normalization amplitudes (A for nocal, B for cal, and C for midcal). This table is printed to the command window.
- 2) a variable named “sum_sq_min,” which contains the value of the sum of squares at the minimum of y_error. This value is printed to the command window.
- 3) column vectors cal_norm, midcal_norm, and nocal_norm, which contain the normalized fractional change in fluorescence data for the curves obtained at high, intermediate, and low calcium concentrations, respectively, and the associated myosin concentrations. These variables are output to the workspace. Double click on the name in the workspace to open the values of the vectors. The contents of each column vector should be copied and pasted into a spreadsheet to build the pooled data set, which consists of the individually normalized data sets for each technical replicate (see the “input_WT” sheet of Sample_data.xls for an example of pooled data).

Command Window

```
>> uiimport

norm_constants =

    1×3 table

      A      B      C
    _____
    1.0709  1.076  1.0362

sum_sq_min =

    0.0205

fx >>
```

1

2

3

Workspace	
Name	Value
best_fit_params	[0.0955,0.0458,0.1982,0.0602,5.4178,1.0709,1.0...
cal_norm	18x2 double
fl_cal	18x1 double
fl_midcal	18x1 double
fl_nocal	18x1 double
hybridopts	1x1 PatternsearchOptions
KBcal	20
KBnocal	0.2900
KS	18
midcal_norm	18x2 double
nocal_norm	18x2 double

e. Global fitting and uncertainty estimation

After normalizing each technical replicate and pooling the data, one can proceed with global fitting of the data and uncertainty estimation. Open the script named “Script_global_fitting.” This script performs global fitting of the pooled data set and calculates confidence intervals for each parameter floated in the fitting routine. The input_WT and input_dE160 sheets in the sample_data.xls file contain the pooled sample data sets after normalization of each technical replicate.

Here, the input is the pooled data set containing the individually normalized curves obtained in the absence of calcium (2 mM EGTA; nocal), at intermediate calcium (pCa 6.25; midcal), and at saturating calcium (pCa 3; cal). Copy the data from the spreadsheet program and import into MATLAB using the `uiimport` command, as described in Section II.c. In the script, assign the imported column vectors to variables (`x1`, `y1`, `x2`, ...) to be used in the fitting routine, changing the variable names if necessary, and set `KBnocal` to the value obtained from the stopped-flow experiments.

Command Window

```
>> uiimport
fx >>
```

Workspace

Name	Value
fl_cal	90x1 double
fl_midcal	90x1 double
fl_nocal	90x1 double
sl_cal	90x1 double
sl_midcal	90x1 double
sl_nocal	90x1 double

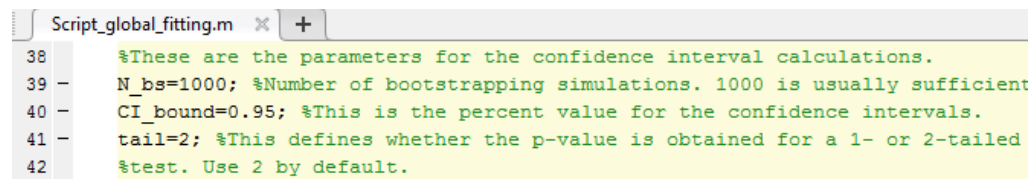
Script_global_fitting.m

```

23 %This assigns the measured values to variables for the rest of the
24 %program. The user can change the name so their data is assigned to the p
25 %variables.
26 x1=sl_nocal; %This is the myosin concentration for the data collected at lo
27 y1=fl_nocal; %This is the fractional change in fluorescence for the data co
28 x2=sl_cal; %This is the myosin concentration for the data collected at high
29 y2=fl_cal; %This is the fractional change in fluorescence for the data coll
30 x3=sl_midcal; %This is the myosin concentration for the data collected at
31 y3=fl_midcal; %This is the fractional change in fluorescence for the data c
32
33 %This assigns the variables appropriate values.
34 KBnocal=0.290; %User should change this to their value for KB from the stop
35 KS=18; %This value is fixed as described in McKillop and Geeves (1993).
36 KBcal=20; %This value is fixed as described in McKillop and Geeves (1993).
```

Set the parameters for the confidence interval calculations. Start with `N_bs = 1000` to perform 1000 bootstrapping simulations. After running the global fitting script, the user should check that a sufficient number of bootstraps have been run for convergence (see Section II.f). `CI_bound` sets

the percent value for the confidence intervals (i.e., $CI_bound = 0.95$ produces 95% confidence intervals). The value of the tail variable should be left as 2 unless a one-tailed test is justified for your experiment.



```
Script_global_fitting.m  x  +
38 %These are the parameters for the confidence interval calculations.
39 - N_bs=1000; %Number of bootstrapping simulations. 1000 is usually sufficient
40 - CI_bound=0.95; %This is the percent value for the confidence intervals.
41 - tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed
42 %test. Use 2 by default.
```

Run the script by clicking “Run and Advance.” This may take 1-2 hours, depending on the number of bootstraps. The output consists of the following:

- 1) a table named “best_fit_values” that contains the best-fit values for each parameter floated during the fitting routine for the actual data. This table is printed to the command window before the bootstrapping simulations are run, so the user can verify that the measured values are reasonable.
- 2) a variable named “sum_sq_min,” which contains the value of the sum of squares at the minimum of y_error. This value is printed to the command window.
- 3) a table named “param_CIs” that contains the best-fit values for each parameter floated during the fitting routine (these will be identical to the values in “best_fit_values”), as well as the difference between best-fit value and the lower (-) and upper (+) bounds of the confidence interval. This table is printed to the command window.
- 4) two matrices: “best_fit_params,” which contains the best-fit values for each parameter floated during the fitting routine for the fitting, and “bootstrap_params,” which contains the best-fit values for each bootstrapping simulation. The order of the parameters is the same as in y_error (KW, KTnocal, KTcal, KTmidcal, nH, A, B, C). These matrices are output to the workspace, which should be saved at this point (Home → Save Workspace). The values can also be copied and pasted into a separate spreadsheet for later use. To access the values, double click on the desired variable in the workspace.

1

2

3

4

Command Window

```

best_fit_values =

1x8 table

    KW    KTnocal    KTcal    KTmidcal    nH    A    B    C
    _____    _____    _____    _____    _____    _____    _____    _____
    0.12954    0.05128    0.16933    0.12495    5.5966    0.99911    1.0025    1.0002

sum_sq_min =

0.8840

param_CIs =

3x8 table

    KW    KTnocal    KTcal    KTmidcal    nH    A    B    C
    _____    _____    _____    _____    _____    _____    _____    _____
value    0.12954    0.05128    0.16933    0.12495    5.5966    0.99911    1.0025    1.0002
-        0.016306    0.04737    0.12115    0.089652    1.6123    0.012609    0.016631    0.013069
+        0.027589    0.089083    0.17908    0.13418    3.8052    0.013343    0.016178    0.011538

fx >>

```

Workspace

Name	Value
best_fit_params	[0.1295,0.0513,0.1693,0.1249,5.5966,0.9991,1.0...
best_fit_values	1x8 table
bootstrap_params	1000x8 double

f. Verifying that a sufficient number of bootstraps have been performed

After running the global fitting and bootstrapping, the user should check whether a sufficient number of bootstrapping simulations were run to give well-defined confidence intervals. If a sufficient number of bootstraps have been performed, the values of the bounds of the confidence interval for each parameter should not change significantly as the number of bootstraps increases. This should be checked for each fitted parameter.

Open the script named “Script_calculate_n_bootstraps.”

The input for this script is the “bootstrap_params” matrix generated during the global fitting and uncertainty estimation (see Section II.e). In the script, set the index (ind) to select the parameter to be examined. The value of this parameter selects the desired column. The columns are in the following order: KW, KTnocal, KTcal, KTmidcal, nH, A, B, and C.

```

Script_calculate_n_bootstraps.m
11 %The input is a matrix of values obtained from bootstrapping (bootstrap_params)
12
13 input=bootstrap_params;
14
15 ind=1; %This is the index of the parameter to be compared. For example,
16 %if the best_fit_params has values (KW,KTnocal,KTcal,KTmidcal,nH,A,B,C),
17 %then ind=1 selects KW, ind=2 selects KTnocal, etc.

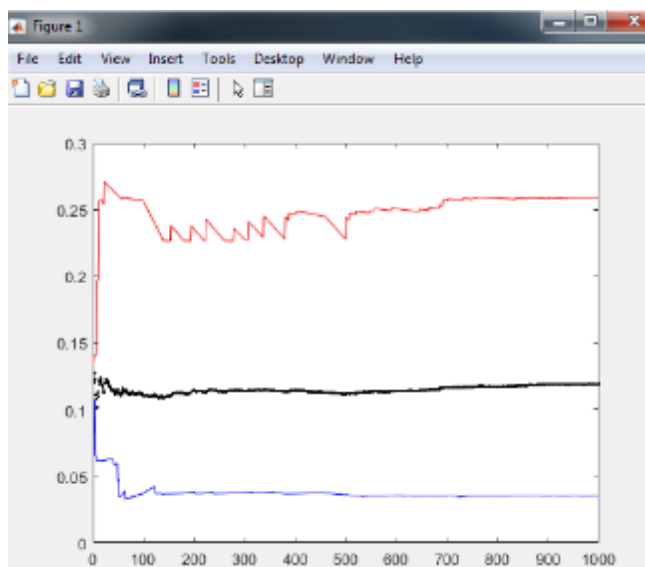
```

Set the parameters for the confidence interval calculations. As before, CI_bound = 0.95 produces

95% confidence intervals and tail = 2 results in a p-value equivalent to that obtained from a two-tailed test.

```
Script_calculate_n_bootstraps.m  X +
19 %These are the parameters for the confidence interval calculations.
20 - CI_bound=0.95; %This is the percent value for the confidence intervals.
21 - tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed
22 %test. Use 2 by default.
```

Run the script by clicking “Run and Advance.” The output consists of a plot (Figure 1) of the mean value of the parameter (black) and the upper (red) and lower (blue) bounds of the confidence interval as a function of the number of bootstrapping simulations.



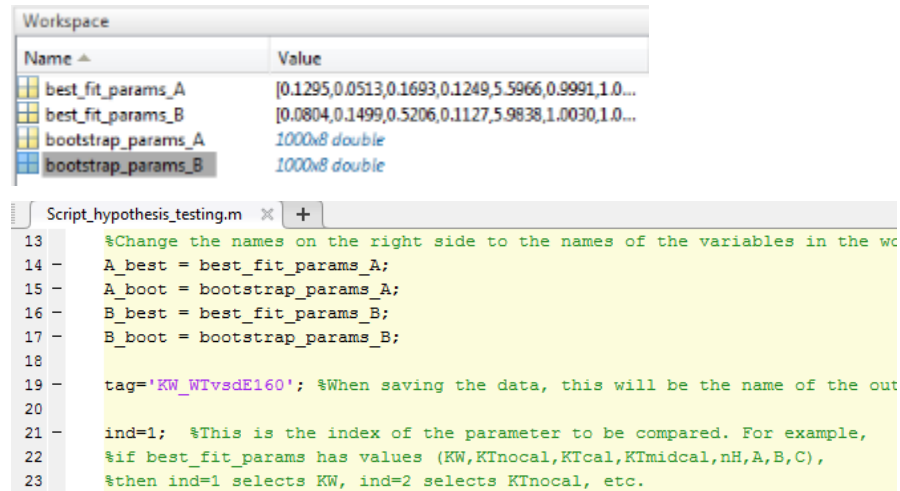
From this plot, the user can visually determine whether a sufficient number of bootstraps have been performed to reach convergence. If an insufficient number of bootstraps have been conducted to get convergence, re-run the fitting section (Section II.e) with a larger number of bootstraps. The plot can be saved by clicking “File → Save As...” in the figure window.

g. Hypothesis testing

After performing global fitting and estimation of parameter uncertainties for two different conditions, one can perform hypothesis testing to see whether a perturbation significantly changes one of the parameters. For the sake of this example, we will refer to the WT data as data set A and the mutant data as data set B.

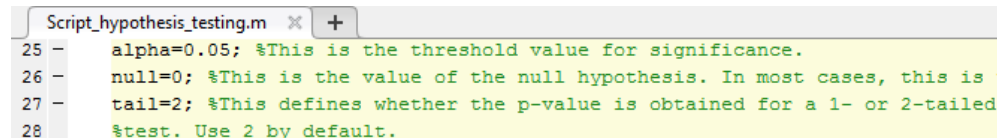
Open the script named “Script_hypothesis_testing.” This script calculates the p-value for the statistical comparison of a parameter determined for two data sets (e.g., K_w for wild-type and mutant). The inputs for this script consist of the “best_fit_params” and “bootstrap_params” matrices generated for data sets A and B during the global fitting and uncertainty estimation (see Section II.e). Import these matrices for both data sets A and B using the `uiimport` command (see Section II.c for details). Rename the variables to distinguish between the two data sets to be compared and update the variable names in the script if necessary. Change the tag variable to the

desired file name for the figures to be saved as output. The figures will be saved in the current folder, so the user should navigate to the desired folder before running the script.



Set the parameters for hypothesis testing in the script. The value of alpha sets the threshold for statistical significance (i.e., $\alpha = 0.05$ results in a test of whether A and B are significantly different at the 95% confidence level), the value of null defines the value of the test statistic for the null hypothesis. Typically, the user will want to set $\text{null} = 0$, meaning there is no statistically significant difference between A and B at the given confidence level. The value of tail defines whether the calculated p-value is equivalent to that obtained from a one- or two-tailed test.

Note for advanced users: The test statistic and the corresponding value of the null hypothesis can be redefined according to the application without loss of generality.



Run the script by clicking, “Run and Advance”. The output consists of the following:

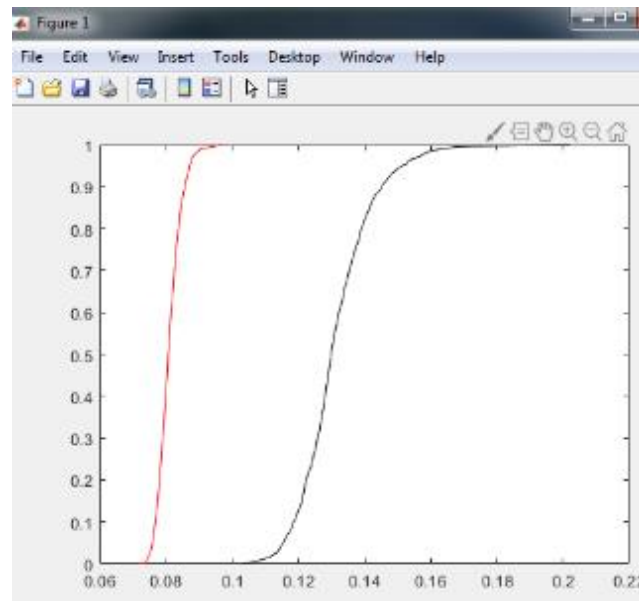
- 1) Figure 1, a plot of the cumulative distributions for bootstrapped values for A (black) and B (red). The value of the cumulative distribution at a given value of x is the fraction of values from the distribution that are less than or equal to that value of x. This plot is saved as <tag>_cumulative.
- 2) Figure 2, a histogram of the values of the test statistic calculated for each bootstrapping simulation. Vertical lines indicate the measured test statistic (yellow), the value of the null hypothesis (blue), and the bounds of the confidence interval determined from the bootstrapping simulations (red). If the blue line falls outside the area bounded by the red lines, then the null hypothesis is rejected at the given confidence level (in this case, 95%). This plot is saved as <tag>_hist.
- 3) Figure 3, a plot of the cumulative distribution of the test statistic for the bootstrapping simulations. The vertical lines are the same as in Figure 2 (yellow--measured value, blue--null hypothesis, red--bounds of the confidence interval). This plot is saved as <tag>_pval.

Note: Each figure is saved to the current directory as a MATLAB figure (.fig) and as an EPS file for downstream use in applications such as Adobe Illustrator. These files are named according to the tag set within the script (see above).

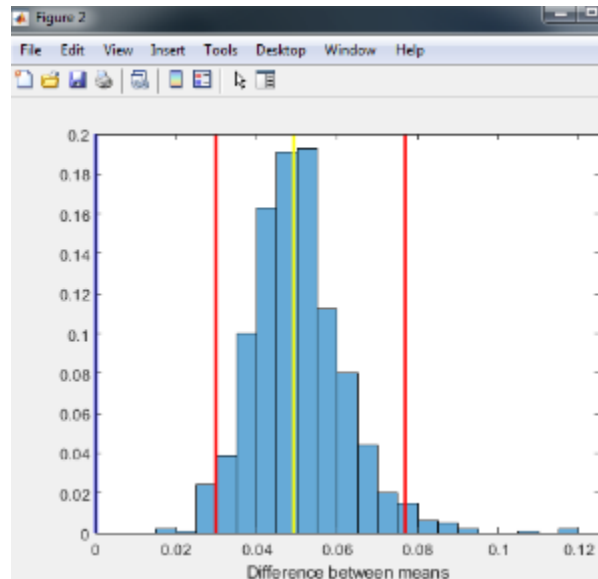
4) a table named “output_params” that contains the best-fit values of the specified parameter, as well as the difference between the best-fit value and the lower (-) and upper (+) bounds of the confidence interval. This table is printed to the command window.

5) a variable named “p_value,” which contains the p-value for the comparison of the specified parameter for data sets A and B. The p-value is equal to twice (for tail = 2) the value of the cumulative distribution at $x = \text{null}$ (in this case, $x = 0$). The value of this variable is printed to the command window. Its value can be copied from the workspace by double clicking on the variable name and copying.

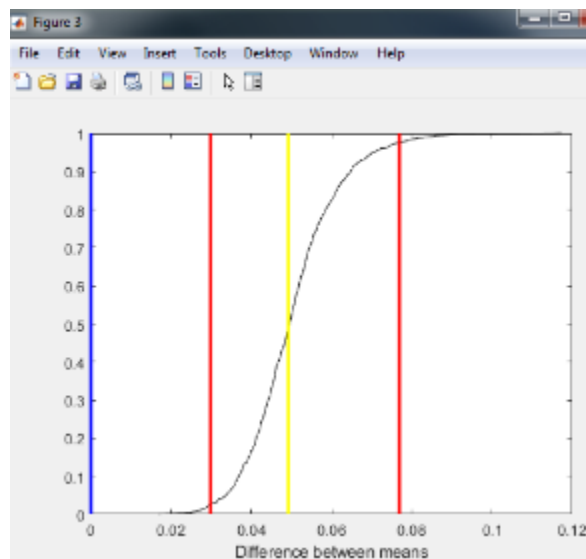
1



2



3



4

```

Command Window

>> uiimport
>> uiimport

output_params =

    3x2 table

           A           B
value    0.12954    0.080355
-        0.016306    0.0052089
+        0.027589    0.0079641

p_value =

    0.0020

fx >>

```

5

h. Classic McKillop and Geeves fitting

This script fits the pooled, normalized data sets (nocal and cal only) independently and calculates confidence intervals for each parameter floated in the fitting routine. Global fitting is not performed, so floated parameters are not shared between the curves obtained at 2 mM EGTA (nocal) and pCa 3 (cal). The nocal and cal data (first four columns of the input_WT and input_dE160 sheets in the sample_data.xls file) are the inputs for classic McKillop and Geeves fitting.

Open the script named “Script_McKillopGeeves_fitting.” Here, the input consists of the pooled data sets containing the individually normalized curves obtained at 2 mM EGTA (nocal) and pCa 3 (cal). Import the data using the “uiimport” command in the command window. As before, assign the imported column vectors to variables (x1, y1, x2, y2) to be used in the fitting routine, changing the variable names if necessary, and set KBnocal to the value obtained from the stopped-flow experiments.

```

Script_McKillopGeeves_fitting.m  x +
15 %This assigns the measured values to variables for the rest of the
16 %program. The user can change the names so their data are assigned to the p
17 %variables.
18 - x1=s1_nocal; %This is the myosin concentration for the data collected at lo
19 - y1=f1_nocal; %This is the fractional change in fluorescence for the data co
20 - x2=s1_cal; %This is the myosin concentration for the data collected at high
21 - y2=f1_cal; %This is the fractional change in fluorescence for the data coll
22
23 %This assigns the variables appropriate values.
24 - KBnocal=0.290; %User should change this to their value for KB from the stop
25 - KS=18; %This value is fixed as described in McKillop and Geeves (1993).
26 - KBcal=20; %This value is fixed as described in McKillop and Geeves (1993).

```

Set the parameters for the confidence interval calculations. Start with $N_{bs} = 1000$ to perform 1000 bootstrapping simulations. The required number of bootstraps can be determined empirically (see Section II.f). As before, CI_bound sets the percent value for the confidence intervals and the value of the tail variable should be left at 2 unless a one-tailed test is justified for your experiment.

```

Script_McKillopGeeves_fitting.m  x +
28 %These are the parameters for the confidence interval calculations.
29 - N_bs=10; %Number of bootstrapping simulations. 1000 is usually sufficient.
30 - CI_bound=0.95; %This is the percent value for the confidence intervals.
31 - tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed
32 %test. Use 2 by default.

```

Note: For this demonstration, we use $N_{bs} = 10$. For such small numbers of bootstraps, the output can vary significantly from run to run.

Run the script by clicking “Run and Advance.” This may take 1-2 hours, depending on the number of bootstraps. The resulting output consists of the following:

- 1) a table named “best_fit_values_nocal” that contains the best-fit values for each parameter floated during fitting of the data acquired at 2 mM EGTA (nocal). This table is printed to the command window.
- 2) a variable named “sum_sq_min_nocal,” which contains the value of the sum of squares at the minimum of y_error_nocal . This value is printed to the command window.
- 3) a table named “best_fit_values_cal” that contains the best-fit values for each parameter floated during fitting of the data acquired at pCa 3 (cal). This table is printed to the command window.
- 4) a variable named “sum_sq_min_cal,” which contains the value of the sum of squares at the minimum of y_error_cal . This value is printed to the command window.
- 5) a table named “param_CI_nocal” that contains the best-fit values for each parameter floated during fitting of the data acquired at 2 mM EGTA (nocal), as well as the difference between best-fit value and the lower (-) and upper (+) bounds of the confidence interval. This table is printed to the command window.
- 6) a table named “param_CI_cal” that contains the best-fit values for each parameter floated during fitting of the data acquired at pCa 3 (cal), as well as the difference between best-fit value and the lower (-) and upper (+) bounds of the confidence interval. This table is printed to the command window.
- 7) matrices named “best_fit_params_nocal” and “best_fit_params_cal” that contain the best-fit values for each parameter floated during fitting of the data acquired at 2 mM EGTA (nocal) and

pCa3 (cal), respectively. This matrix can be copied by double clicking its name in the workspace and then copying the data.

8) matrices named “bootstrap_params_nocal” and “bootstrap_params_cal” that contain the best-fit values for each bootstrapping simulation for data acquired at 2 mM EGTA (nocal) and pCa3 (cal), respectively. This matrix can be copied by double clicking its name in the workspace and then copying the data.

```

Command Window

best_fit_values_nocal =

    1×4 table
      KW      KTnocal      nH      A
      ---      ---      ---      ---
    0.1313    0.032688    6.0395    0.99567

sum_sq_min_nocal =

    0.4202

best_fit_values_cal =

    1×4 table
      KW      KTcal      nH      B
      ---      ---      ---      ---
    0.16196    0.21823    3.6748    0.98573

sum_sq_min_cal =

    0.1697
  
```

1

2

3

4

5

```

param_CI_nocal =

3x4 table

      KW      KTnocal      nH      A
      ----      -
value  0.1313    0.032688    6.0395    0.99567
-      0.030185    0.011885    3.3815    0.02063
+      0.28418    0.022115    1.831     0.020047

```

6

```

param_CI_cal =

3x4 table

      KW      KTcal      nH      B
      ----      -
value  0.16196    0.21823    3.6748    0.98573
-      0.03456    0.21774    1.1534    0.0076154
+      2.3614     0.12004    4.6699    0.018754

```

7

8

Workspace

Name ▲	Value
best_fit_params_cal	[0.1620,0.2182,3.6748,0.9857]
best_fit_params_nocal	[0.1313,0.0327,6.0395,0.9957]
best_fit_values_cal	1x4 table
best_fit_values_nocal	1x4 table
bootstrap_params_cal	10x4 double
bootstrap_params_nocal	10x4 double
bootstrap_sum_sq_min	11x1 double

If the user wants to test hypotheses (see Section II.g) based on the data produced in this section, the workspace should be saved (Home → Save Workspace). The values can also be copied and pasted into a spreadsheet for later use, as described earlier.

III. Hypothesis testing for other types of data

a. Statistical comparison of the means of two data sets

The script named “Script_hypothesis_testing” (see Section II.g) enables hypothesis testing for any data set, given the measured parameter value and the set of parameter values calculated from bootstrapping simulations. Here we calculate the mean and confidence intervals for a generic data set, in this case a distribution of cell widths for two distinct populations of cells. These data are provided in the sheet named “cell_width” in the sample_data.xls file. The data here is from **Disrupted Mechanobiology Links the Molecular and Cellular Phenotypes in Familial Dilated Cardiomyopathy**. Sarah R. Clippinger, Paige E. Cloonan, Lina Greenberg, Melanie Ernst, W. Tom Stump, Michael J. Greenberg.

Open the script named “Script_mean.” Import your data set using “uiimport” (see Section II.c). The code assigns the imported column vector to the variable x1. If necessary, change the right side of the equation to match the name of the imported variable.

```
Script_mean.m  x +
11      %Here, we input a range of measured values into x1.
12 -    x1=data;
```

Set the parameters for the confidence interval calculations. Start with N_bs = 1000 to perform 1000 bootstrapping simulations. The required number of bootstraps can be determined empirically (see Section II.f). CI_bound sets the percent value for the confidence intervals (i.e., CI_bound = 0.95 produces 95% confidence intervals). The value of the tail variable should be left at 2 unless a one-tailed test is justified for your experiment.

```
Script_mean.m  x +
14      %These are the parameters for the confidence interval calculations.
15 -    N_bs=1000; %Number of bootstrapping simulations. 1000 is usually sufficient
16 -    CI_bound=0.95; %This is the percent value for the confidence intervals.
17 -    tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed
18      %test. Use 2 by default.
```

Run the script by clicking “Run and Advance.” The output consists of the following:

- 1) a table named “param_CI” that contains the mean value of the real data set, as well as the difference between mean and the lower (-) and upper (+) bounds of the confidence interval. This table is printed to the command window.
- 2) matrices named “best_fit_param” and “bootstrap_param” that contain the mean values of the real data set and of each bootstrapping simulation, respectively. These matrices are output to the workspace, which should be saved at this point (Home → Save Workspace). The values can also be copied and pasted into a spreadsheet for later use. To do this, double click on the variable names in the workspace and copy the values. Hypothesis testing can be performed using the script named “Script_hypothesis_testing” (see Section II.g) with the “best_fit_param” and “bootstrap_param” matrices as input.

1

Command Window

```
>> uiimport

param_CI =

3x1 table

    paramCI
    _____
    mean      13.854
    -         0.39997
    +         0.40976

fx >>
```

2

Workspace

Name	Value
best_fit_param	13.8544
bootstrap_param	1000x1 double

Note: Although this script simply calculates the mean and does not perform fitting, we have chosen to retain the “best_fit_param” nomenclature to facilitate downstream use of the hypothesis testing script (see Section II.g).

b. Statistical comparison of the medians of two data sets

Here we are interested in comparing the medians of two populations. The procedure is identical to that described in the previous section (III.a), with the exception that the script calculates the median rather than the mean of the data set.

Open the script named “Script_median.” Import your data, renaming the variable in the code if necessary. Set the parameters for the confidence interval calculations. As before, start with $N_{bs} = 1000$ to perform 1000 bootstrapping simulations. CI_bound sets the percent value for the confidence intervals, and the value of the tail variable should be left at 2 unless a one-tailed test is justified for your experiment.

```
Script_median.m  x +
11 %Here, we input a range of measured values into x1.
12 x1=data;
13
14 %These are the parameters for the confidence interval calculations.
15 N_bs=1000; %Number of bootstrapping simulations. 1000 is usually sufficient
16 CI_bound=0.95; %This is the percent value for the confidence intervals.
17 tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed
18 %test. Use 2 by default.
```

Run the script by clicking “Run and Advance.” The output consists of the following:

- 1) a table named “param_CI” that contains the median value of the real data set, as well as the difference between median and the lower (-) and upper (+) bounds of the confidence interval. This table is printed to the command window.
- 2) matrices named “best_fit_param” and “bootstrap_param” that contain the median value of the real data set and of each bootstrapping simulation, respectively. These matrices are output to the workspace, which should be saved at this point (Home → Save Workspace). The values can also

be copied and pasted into a spreadsheet for later use. Hypothesis testing can be performed using the script named “Script_hypothesis_testing” (see Section II.g) with the “best_fit_param” and “bootstrap_param” matrices as input.

1

```
Command Window
>> uiimport

param_CI =

3x1 table

      paramCI
      -----
median    13.691
-         0.4485
+         0.521

fx >>
```

2

```
Workspace

Name      Value
-----
best_fit_param  13.6905
bootstrap_param 1000x1 double
```

Note: Although this script simply calculates the median and does not perform fitting, we have chosen to retain the “best_fit_param” nomenclature in order to facilitate downstream use of the hypothesis testing script (see Section II.g).

IV. Appendix

This section includes the full text of the script files.

a. Script_normalization_replicate.m

```
%This MATLAB code is associated with the following manuscript: Barrick,  
%S.K., S.R. Clippinger, L. Greenberg, M.J. Greenberg. 2019. Computational  
%tool to study perturbations in muscle regulation and its application to  
%heart disease.  
  
%This is a script for normalization of a single technical replicate.  
  
%%  
  
%Normalization of technical replicates  
  
%This script performs fitting of fluorescence titration data to obtain the  
%normalization constants A, B, and C for data collected at 2 mM EGTA (nocal),  
%pCa 3 (cal), and pCa 6.25 (midcal), respectively. For each titration, the  
%myosin concentration and fractional change in fluorescence at that  
%concentration are recorded.  
  
%This assigns the measured values to variables for the rest of the program.  
%The user can change the names so their data are assigned to the proper  
%variables.  
x1=s1_nocal; %This is the myosin concentration for the data collected at low  
calcium (2 mM EGTA)  
y1=f1_nocal; %This is the fractional change in fluorescence for the data  
collected at low calcium (2 mM EGTA)  
x2=s1_cal; %This is the myosin concentration for the data collected at high  
calcium (pCa 3)  
y2=f1_cal; %This is the fractional change in fluorescence for the data  
collected at high calcium (pCa 3)  
x3=s1_midcal; %This is the myosin concentration for the data collected at  
intermediate calcium (pCa 6.25)  
y3=f1_midcal; %This is the fractional change in fluorescence for the data  
collected at intermediate calcium (pCa 6.25)  
  
%This assigns the variables appropriate values.  
KBnocal=0.290; %User should change this to their value for KB from the  
stopped-flow experiments  
KS=18; %This value is fixed as described in McKillop and Geeves (1993).  
KBcal=20; %This value is fixed as described in McKillop and Geeves (1993).  
  
%This removes any non-numerical values in the imported data set (for  
%example, if you have blank spaces due to different numbers of data points  
%for each calcium concentration).  
q=find(isnan(y1));  
x1(q)=[];  
y1(q)=[];  
q=find(isnan(y2));  
x2(q)=[];  
y2(q)=[];  
q=find(isnan(y3));  
x3(q)=[];
```

```

y3(q)=[];

%This clears the variables used during fitting to make it easier to
%consecutively perform multiple iterations of the fitting.
clear y_error* best_fit_params sum_sq_min *_norm

%The following code performs the fit via simulated annealing and minimization
of
%y_error. Here, we are interested in obtaining the normalization amplitudes
%(A, B, C) to enable us to normalize each technical replicate before
combining
%the data into a pooled data set for global fitting and error estimation.

%This defines y_error for the nocal, cal, and midcal conditions as the sum
%of squared differences between Equation 1 (see user guide) and the
fractional
%change in fluorescence data for that condition (y1, y2, y3).
y_error_nocal = @(X1,KW,KTnocal,nH,A) sum(((A.*KW.*x1.*(1 + KW.*x1.*(1 +
KS)).^(nH - 1)).*(KTnocal.*(1 + KS).^nH + 1))./((KTnocal.*(1 + KW.*x1.*(1 +
KS)).^nH + (1 + KW.*x1).^nH + 1./KBnocal).*(1 + KS).^(nH - 1)) - y1).^2);
y_error_cal = @(X2,KW,KTcal,nH,B) sum(((B.*KW.*x2.*(1 + KW.*x2.*(1 +
KS)).^(nH - 1)).*(KTcal.*(1 + KS).^nH + 1))./((KTcal.*(1 + KW.*x2.*(1 +
KS)).^nH + (1 + KW.*x2).^nH + 1./KBcal).*(1 + KS).^(nH - 1)) - y2).^2);
y_error_midcal = @(X3,KW,KTmidcal,nH,C) sum(((C.*KW.*x3.*(1 + KW.*x3.*(1 +
KS)).^(nH - 1)).*(KTmidcal.*(1 + KS).^nH + 1))./((KTmidcal.*(1 + KW.*x3.*(1 +
KS)).^nH + (1 + KW.*x3).^nH + 1./KBcal).*(1 + KS).^(nH - 1)) - y3).^2);

%The error from all three curves is summed to create y_error, which is later
%minimized in the global fit.
y_error = @(X1,X2,X3,KW,KTnocal,KTcal,KTmidcal,nH,A,B,C)
y_error_nocal(X1,KW,KTnocal,nH,A)+y_error_cal(X2,KW,KTcal,nH,B)+y_error_midca
l(X3,KW,KTmidcal,nH,C);

%This sets the options for the fitting.
hybridopts = optimoptions('patternsearch','TolFun',1e-8,'TolX',1e-
7,'TolMesh',1e-
7,'display','none','MaxIter',50000,'MaxFunEvals',60000,'TimeLimit',120);
options = optimoptions('simulannealbnd','TolFun',1e-
8,'TimeLimit',10,'display','none','InitialTemperature',0.5,'MaxIter',100000);
options =
optimoptions('simulannealbnd',options,'HybridFcn',{@patternsearch,hybridopts}
);

%Note that the 'optimoptions' command is not compatible with some earlier
%versions of MATLAB. If you get an error related to this command, use the
%following code to set the options instead:
%hybridopts = psoptimset('TolFun',1e-8,'TolX',1e-7,'TolMesh',1e-
7,'display','none','MaxIter',50000,'MaxFunEvals',60000,'TimeLimit',120);
%options = saoptimset('TolFun',1e-
8,'TimeLimit',10,'display','none','InitialTemperature',0.5,'MaxIter',100000);
%options = saoptimset(options,'HybridFcn',{@patternsearch,hybridopts});

%This performs the global fit by minimizing y_error.

```

```

[best_fit_params,sum_sq_min] = simulannealbnd(@(z)
y_error(x1,x2,x3,z(1),z(2),z(3),z(4),z(5),z(6),z(7),z(8)),[0.03,0.12,0.24,0.1
8,7,1,1,1],[0,0,0,0,0,0,0,0],[10,10,10,10,20,10,10,10],options);

%This returns a table containing the normalization amplitudes for nocal (A),
%cal (B), and midcal (C), as well as the value of the sum of squares at the
%minimum (sum_sq_min).
normIDs = {'A','B','C'};
norm_constants = array2table(best_fit_params(6:8),'VariableNames',normIDs)
sum_sq_min

%The normalized fluorescence values and corresponding myosin concentrations
%are then output to the workspace. The normalized values for each technical
%replicate should be pooled to form a pooled data set, used for the global
fitting.
nocal_norm = [x1 y1./best_fit_params(6)]; %Normalized data for nocal
cal_norm = [x2 y2./best_fit_params(7)]; %Normalized data for cal
midcal_norm = [x3 y3./best_fit_params(8)]; %Normalized data for midcal

```

b. Script_global_fitting.m

```
%This MATLAB code is associated with the following manuscript: Barrick,  
%S.K., S.R. Clippinger, L. Greenberg, M.J. Greenberg. 2019. Computational  
%tool to study perturbations in muscle regulation and its application to  
%heart disease.  
  
%This is a script for global fitting of the data and estimation of  
%uncertainties by bootstrapping.  
  
%%  
  
%Global fitting and uncertainty estimation  
  
%This script performs global fitting and bootstrapping simulations of the  
%pooled fluorescence titration data collected at three calcium  
concentrations:  
%2 mM EGTA (nocal), pCa 3 (cal), and pCa 6.25 (midcal).  
  
%NOTE: Each technical replicate should be normalized separately, using the  
%script named "Script_normalization_replicate." These normalized curves  
%should be combined into a pooled data set consisting of the individually  
%normalized data for each technical replicate. This pooled data set is the  
%input for this script.  
  
%This assigns the measured values to variables for the rest of the  
%program. The user can change the name so their data is assigned to the  
proper  
%variables.  
x1=s1_nocal; %This is the myosin concentration for the data collected at low  
calcium (2 mM EGTA)  
y1=f1_nocal; %This is the fractional change in fluorescence for the data  
collected at low calcium (2 mM EGTA)  
x2=s1_cal; %This is the myosin concentration for the data collected at high  
calcium (pCa 3)  
y2=f1_cal; %This is the fractional change in fluorescence for the data  
collected at high calcium (pCa 3)  
x3=s1_midcal; %This is the myosin concentration for the data collected at  
intermediate calcium (pCa 6.25)  
y3=f1_midcal; %This is the fractional change in fluorescence for the data  
collected at intermediate calcium (pCa 6.25)  
  
%This assigns the variables appropriate values.  
KBnocal=0.290; %User should change this to their value for KB from the  
stopped flow experiments  
KS=18; %This value is fixed as described in McKillop and Geeves (1993).  
KBcal=20; %This value is fixed as described in McKillop and Geeves (1993).  
  
%These are the parameters for the confidence interval calculations.  
N_bs=1000; %Number of bootstrapping simulations. 1000 is usually sufficient.  
CI_bound=0.95; %This is the percent value for the confidence intervals.  
tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed  
%test. Use 2 by default.  
  
%This removes any non-numerical values in the imported data set (for
```

```

%example, if you have blank spaces due to different numbers of data points
%for each calcium concentration).
q=find(isnan(y1));
x1(q)=[];
y1(q)=[];
q=find(isnan(y2));
x2(q)=[];
y2(q)=[];
q=find(isnan(y3));
x3(q)=[];
y3(q)=[];

%This clears the variables used during fitting to make it easier to
%consecutively perform multiple iterations of the fitting.
clear y_error* best_fit_params *_sq_min bs* bootstrap_params*

%This sets the options for the fitting.
hybridopts = optimoptions('patternsearch','TolFun',1e-8,'TolX',1e-
7,'TolMesh',1e-
7,'display','none','MaxIter',50000,'MaxFunEvals',60000,'TimeLimit',120);
options = optimoptions('simulannealbnd','TolFun',1e-
8,'TimeLimit',10,'display','none','InitialTemperature',0.5,'MaxIter',100000);
options =
optimoptions('simulannealbnd',options,'HybridFcn',{@patternsearch,hybridopts}
);

%Note that the 'optimoptions' command is not compatible with some earlier
%versions of MATLAB. If you get an error related to this command, use the
%following code to set the options instead:
%hybridopts = psoptimset('TolFun',1e-8,'TolX',1e-7,'TolMesh',1e-
7,'display','none','MaxIter',50000,'MaxFunEvals',60000,'TimeLimit',120);
%options = saoptimset('TolFun',1e-
8,'TimeLimit',10,'display','none','InitialTemperature',0.5,'MaxIter',100000);
%options = saoptimset(options,'HybridFcn',{@patternsearch,hybridopts});

%The following code performs the global fit via simulated annealing and
%least-squares minimization of y_error. It returns the set of best-fit
parameters
%(best_fit_params) as well as the value of the sum of squares at the
%minimum (sum_sq_min). The order of parameters in best_fit_params is the
%same as in y_error. Fixed parameters are the values of KBnocal, KBcal, and
%KS. Fitted parameters are nH (the Hill coefficient), KW, KT at low
%(KTnocal), high (KTcal), and intermediate calcium (KTmidcal), and the
%normalization amplitudes (A,B,C) for the pooled data sets.

%This defines y_error for the nocal, cal, and midcal conditions as the sum
%of squared differences between Equation 1 (see user guide) and pooled
%fractional change in fluorescence data for that condition (y1, y2, y3).
y_error_nocal = @(X1,KW,KTnocal,nH,A) sum(((A.*KW.*x1.*(1 + KW.*x1.*(1 +
KS)).^(nH - 1)).*(KTnocal.*(1 + KS).^nH + 1))./((KTnocal.*(1 + KW.*x1.*(1 +
KS)).^nH + (1 + KW.*x1).^nH + 1./KBnocal).*(1 + KS).^(nH - 1)) - y1).^2);
y_error_cal = @(X2,KW,KTcal,nH,B) sum(((B.*KW.*x2.*(1 + KW.*x2.*(1 +
KS)).^(nH - 1)).*(KTcal.*(1 + KS).^nH + 1))./((KTcal.*(1 + KW.*x2.*(1 +
KS)).^nH + (1 + KW.*x2).^nH + 1./KBcal).*(1 + KS).^(nH - 1)) - y2).^2);

```



```

y_error_midcal = @(X3,KW,KTmidcal,nH,C) sum(((C.*KW.*x3.*(1 + KW.*x3.*(1 +
KS)).^(nH - 1)).*(KTmidcal.*(1 + KS).^nH + 1))./((KTmidcal.*(1 + KW.*x3.*(1 +
KS)).^nH + (1 + KW.*x3).^nH + 1./KBcal)).*(1 + KS).^(nH - 1)) - y3).^2);

%The error from all three curves is summed to create y_error, which is later
%minimized in the global fit.
y_error = @(X1,X2,X3,KW,KTnocal,KTcal,KTmidcal,nH,A,B,C)
y_error_nocal(X1,KW,KTnocal,nH,A)+y_error_cal(X2,KW,KTcal,nH,B)+y_error_midca
l(X3,KW,KTmidcal,nH,C);

%This performs the global fit by minimizing y_error. Here, we are interested
%in the best-fit parameter values.
[best_fit_params,sum_sq_min] = simulannealbnd(@(z)
y_error(x1,x2,x3,z(1),z(2),z(3),z(4),z(5),z(6),z(7),z(8)), [0.03,0.12,0.24,0.1
8,7,1,1,1], [0,0,0,0,0,0,0,0], [10,10,10,10,20,10,10,10],options);

%This prints to the command window a table containing the best-fit values
%of the parameters floated during the fitting (best_fit_values), as well as
%the value of the sum of squares at the minimum (sum_sq_min).
paramIDs = {'KW','KTnocal','KTcal','KTmidcal','nH','A','B','C'};
best_fit_values = array2table(best_fit_params,'VariableNames',paramIDs)
sum_sq_min

%The following code performs the bootstrapping simulations for error
estimation.
%It outputs a matrix, bootstrap_params, that contains the fit values
%for each of the simulated data sets. The columns give the values of the
%parameters and each row contains values obtained from a separate simulation.

%This randomly selects indices of the original, pooled data set to
%generate the simulated (i.e., bootstrapped) data sets.
bs_ind_nocal = ceil(length(x1)*rand(length(x1),N_bs));
bs_ind_cal = ceil(length(x2)*rand(length(x2),N_bs));
bs_ind_midcal = ceil(length(x3)*rand(length(x3),N_bs));

%This generates fits to the N_bs simulated data sets. Each row of
%bootstrap_fit_params contains the best-fit parameters for one round of
%resampling. The order of parameters in the columns of bootstrap_params is
%the same as in y_error.
parfor i=1:N_bs
    x_ind_nocal = x1(bs_ind_nocal(:,i));
    y_ind_nocal = y1(bs_ind_nocal(:,i));
    x_ind_cal = x2(bs_ind_cal(:,i));
    y_ind_cal = y2(bs_ind_cal(:,i));
    x_ind_midcal = x3(bs_ind_midcal(:,i));
    y_ind_midcal = y3(bs_ind_midcal(:,i));

y_error_nocal = @(X1,KW,KTnocal,nH,A) sum(((A.*KW.*X1.*(1 + KW.*X1.*(1 +
KS)).^(nH - 1)).*(KTnocal.*(1 + KS).^nH + 1))./((KTnocal.*(1 + KW.*X1.*(1 +
KS)).^nH + (1 + KW.*X1).^nH + 1./KBnocal)).*(1 + KS).^(nH - 1)) -
y_ind_nocal).^2);
y_error_cal = @(X2,KW,KTcal,nH,B) sum(((B.*KW.*X2.*(1 + KW.*X2.*(1 +
KS)).^(nH - 1)).*(KTcal.*(1 + KS).^nH + 1))./((KTcal.*(1 + KW.*X2.*(1 +
KS)).^nH + (1 + KW.*X2).^nH + 1./KBcal)).*(1 + KS).^(nH - 1)) -
y_ind_cal).^2);

```

```

y_error_midcal = @(X3,KW,KTmidcal,nH,C) sum(((C.*KW.*X3.*(1 + KW.*X3.*(1 +
KS)).^(nH - 1)).*(KTmidcal.*(1 + KS).^nH + 1))./((KTmidcal.*(1 + KW.*X3.*(1 +
KS)).^nH + (1 + KW.*X3).^nH + 1./KBcal)).*(1 + KS).^(nH - 1)) -
y_ind_midcal).^2);
y_error = @(X1,X2,X3,KW,KTnocal,KTcal,KTmidcal,nH,A,B,C)
y_error_nocal(X1,KW,KTnocal,nH,A)+y_error_cal(X2,KW,KTcal,nH,B)+y_error_midca
l(X3,KW,KTmidcal,nH,C)
[bootstrap_params(i,:),bootstrap_sum_sq_min(i,:)] = simulannealbnd(@(z)
y_error(x_ind_nocal,x_ind_cal,x_ind_midcal,z(1),z(2),z(3),z(4),z(5),z(6),z(7)
,z(8)), [0.1,0.12,0.24,0.18,7,1,1,1],[0,0,0,0,0,0,0,0],[10,10,10,10,20,10,10,1
0],options);
end
%delete(gcf) %This optional step closes the parallel pool when the loop is
done running.

%This calculates the confidence intervals (CIs) based on the bootstrapped
data.
%It prints to the command window a table containing the confidence intervals
%for the best-fit values of the parameters floated during the fitting
(param_CIs).
%The first row of param_CIs contains the measured values for each parameter.
%The second row contains the distance from the lower bound of the CI to the
%measured value, and the third row contains the distance from the upper bound
%of the CI to the measured value.
clear CI paramCIs param_CIs

rowNames = {'value','-','+'};
paramIDs = {'KW','KTnocal','KTcal','KTmidcal','nH','A','B','C'};
CI = prctile(bootstrap_params,[100*(1-CI_bound)/tail,100*(1-(1-
CI_bound)/tail)]);
paramCIs = [best_fit_params;best_fit_params-CI(1,:);CI(2,:)-best_fit_params];
param_CIs =
array2table(paramCIs,'RowNames',rowNames,'VariableNames',paramIDs)

```

c. Script_McKillopGeeves_fitting.m

```
%This MATLAB code is associated with the following manuscript: Barrick,  
%S.K., S.R. Clippinger, L. Greenberg, M.J. Greenberg. 2019. Computational  
%tool to study perturbations in muscle regulation and its application to  
%heart disease.  
  
%This script performs the fitting of the data to the McKillop and Geeves  
%model without global fitting. The code extends the original method to  
%include error estimation of the fit parameters.  
  
%%  
  
%Here, we have data collected at two calcium concentrations: 2 mM EGTA  
(nocal)  
%and pCa 3 (cal).  
  
%This assigns the measured values to variables for the rest of the  
%program. The user can change the names so their data are assigned to the  
%proper  
%variables.  
x1=s1_nocal; %This is the myosin concentration for the data collected at low  
calcium (2 mM EGTA)  
y1=f1_nocal; %This is the fractional change in fluorescence for the data  
collected at low calcium (2 mM EGTA)  
x2=s1_cal; %This is the myosin concentration for the data collected at high  
calcium (pCa 3)  
y2=f1_cal; %This is the fractional change in fluorescence for the data  
collected at high calcium (pCa 3)  
  
%This assigns the variables appropriate values.  
KBnocal=0.290; %User should change this to their value for KB from the  
stopped-flow experiments.  
KS=18; %This value is fixed as described in McKillop and Geeves (1993).  
KBcal=20; %This value is fixed as described in McKillop and Geeves (1993).  
  
%These are the parameters for the confidence interval calculations.  
N_bs=1000; %Number of bootstrapping simulations. 1000 is usually sufficient.  
CI_bound=0.95; %This is the percent value for the confidence intervals.  
tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed  
%test. Use 2 by default.  
  
%This removes any non-numerical values in the imported data set (for  
%example, if you have blank spaces due to different numbers of data points  
%for each calcium concentration).  
q=find(isnan(y1));  
x1(q)=[];  
y1(q)=[];  
q=find(isnan(y2));  
x2(q)=[];  
y2(q)=[];  
  
%This clears the variables used during fitting to make it easier to  
%consecutively perform multiple iterations of the fitting.  
clear y_error* best_fit_params *_sq_min bs* bootstrap_params*
```

```
%This sets the options for the fitting.
hybridopts = optimoptions('patternsearch','TolFun',1e-8,'TolX',1e-
7,'TolMesh',1e-
7,'display','none','MaxIter',50000,'MaxFunEvals',60000,'TimeLimit',120);
options = optimoptions('simulannealbnd','TolFun',1e-
8,'TimeLimit',10,'display','none','InitialTemperature',0.5,'MaxIter',100000);
options =
optimoptions('simulannealbnd',options,'HybridFcn',{@patternsearch,hybridopts}
);
```

```
%Note that the 'optimoptions' command is not compatible with some earlier
%versions of MATLAB. If you get an error related to this command, use the
%following code to set the options instead:
%hybridopts = psoptimset('TolFun',1e-8,'TolX',1e-7,'TolMesh',1e-
7,'display','none','MaxIter',50000,'MaxFunEvals',60000,'TimeLimit',120);
%options = saoptimset('TolFun',1e-
8,'TimeLimit',10,'display','none','InitialTemperature',0.5,'MaxIter',100000);
%options = saoptimset(options,'HybridFcn',{@patternsearch,hybridopts});
```

```
%The following code performs the fitting of the two curves via simulated
%annealing and least-squares minimization. Note that this is not global
fitting,
%as each curve is fit individually. It returns the set of best-fit
parameters
%(best_fit_params) as well as the value of the sum of squares at the
%minimum (sum_sq_min). The order of parameters in best_fit_params is the
%same as in y_error. Fixed parameters are the values of KBnocal, KBcal, and
%KS. Fitted parameters are nH (the Hill coefficient), KW, KT at low
%(KTnocal) and high (KTcal) calcium (KTmidcal), and the
%normalization amplitudes (A,B).
```

```
%This defines y_error for the nocal and cal conditions as the sum
%of squared differences between Equation 1 (see user guide) and pooled
%fractional change in fluorescence data for that condition (y1, y2).
y_error_nocal = @(X1,KW,KTnocal,nH,A) sum(((A.*KW.*x1.*(1 + KW.*x1.*(1 +
KS)).^(nH - 1)).*(KTnocal.*(1 + KS).^(nH + 1))./((KTnocal.*(1 + KW.*x1.*(1 +
KS)).^nH + (1 + KW.*x1).^nH + 1./KBnocal).*(1 + KS).^(nH - 1)) - y1).^2);
y_error_cal = @(X2,KW,KTcal,nH,B) sum(((B.*KW.*x2.*(1 + KW.*x2.*(1 +
KS)).^(nH - 1)).*(KTcal.*(1 + KS).^(nH + 1))./((KTcal.*(1 + KW.*x2.*(1 +
KS)).^nH + (1 + KW.*x2).^nH + 1./KBcal).*(1 + KS).^(nH - 1)) - y2).^2);
```

```
%This line performs the fitting by minimizing y_error_nocal.
[best_fit_params_nocal,sum_sq_min_nocal] = simulannealbnd(@(z)
y_error_nocal(x1,z(1),z(2),z(3),z(4)), [0.03,0.12,7,1], [0,0,0,0], [10,10,20,10]
,options);
```

```
%This line performs the fitting by minimizing y_error_cal.
[best_fit_params_cal,sum_sq_min_cal] = simulannealbnd(@(z)
y_error_cal(x2,z(1),z(2),z(3),z(4)), [0.03,0.24,7,1], [0,0,0,0], [10,10,20,10],o
ptions);
```

```
%This prints to the command window a table containing the best-fit values
%of the parameters floated during each fit (best_fit_values_nocal and
%best_fit_values_cal), as well as the value of the sum of squares at the
```

```

%minimum for each fit (sum_sq_min_nocal and sum_sq_min_cal).
paramIDs_nocal = {'KW','KTnocal','nH','A'};
paramIDs_cal = {'KW','KTcal','nH','B'};
best_fit_values_nocal =
array2table(best_fit_params_nocal,'VariableNames',paramIDs_nocal)
sum_sq_min_nocal
best_fit_values_cal =
array2table(best_fit_params_cal,'VariableNames',paramIDs_cal)
sum_sq_min_cal

%The following code performs the bootstrapping simulations for error
estimation.
%It outputs a matrix bootstrap_params that contains the fit values
%for each of the simulated data sets. The columns give the values of the
%parameters and each row contains values obtained from a separate simulation.

%This randomly selects indices of the original, pooled data set to
%generate the simulated (i.e., bootstrapped) data sets.
bs_ind_nocal = ceil(length(x1)*rand(length(x1),N_bs));
bs_ind_cal = ceil(length(x2)*rand(length(x2),N_bs));

%This generates fits to the N_bs simulated data sets. Each row of
%bootstrap_fit_params contains the best-fit parameters for one round of
%resampling. The order of parameters in the columns of bootstrap_params is
%the same as in y_error.
parfor i=1:N_bs
    x_ind_nocal = x1(bs_ind_nocal(:,i));
    y_ind_nocal = y1(bs_ind_nocal(:,i));
    x_ind_cal = x2(bs_ind_cal(:,i));
    y_ind_cal = y2(bs_ind_cal(:,i));

    y_error_nocal = @(X1,KW,KTnocal,nH,A) sum(((A.*KW.*X1.*(1 + KW.*X1.*(1 +
KS)).^(nH - 1)).*(KTnocal.*(1 + KS).^nH + 1))./(KTnocal.*(1 + KW.*X1.*(1 +
KS)).^nH + (1 + KW.*X1).^nH + 1./KBnocal).*(1 + KS).^(nH - 1)) -
y_ind_nocal).^2);
    [bootstrap_params_nocal(i,:),bootstrap_sum_sq_min(i+1,:)] =
simulannealbnd(@(z)
y_error_nocal(x_ind_nocal,z(1),z(2),z(3),z(4)), [0.03,0.12,7,1], [0,0,0,0], [10,
10,20,10],options)

    y_error_cal = @(X2,KW,KTcal,nH,B) sum(((B.*KW.*X2.*(1 + KW.*X2.*(1 +
KS)).^(nH - 1)).*(KTcal.*(1 + KS).^nH + 1))./(KTcal.*(1 + KW.*X2.*(1 +
KS)).^nH + (1 + KW.*X2).^nH + 1./KBcal).*(1 + KS).^(nH - 1)) -
y_ind_cal).^2);
    [bootstrap_params_cal(i,:),bootstrap_sum_sq_min(i+1,:)] = simulannealbnd(@(z)
y_error_cal(x_ind_cal,z(1),z(2),z(3),z(4)), [0.03,0.24,7,1], [0,0,0,0], [10,10,2
0,10],options)

end
%delete(gcf) %This optional step closes the parallel pool when the loop is
done running.

%This calculates the confidence intervals based on the bootstrapped data.
%It returns param_CI_nocal and param_CI_cal. The first row contains the
%measured values for each parameter. The second row contains the distance

```

```

%from the lower bound of the CI to the measured value, and the third row
%contains the distance from the upper bound of the CI to the measured value.
clear CI_nocal paramCIs_nocal CI_cal paramCIs_cal param_CI_nocal param_CI_cal

rowNames = {'value', '-', '+'};
paramIDs_nocal = {'KW', 'KTnocal', 'nH', 'A'};
paramIDs_cal = {'KW', 'KTcal', 'nH', 'B'};
CI_nocal = prctile(bootstrap_params_nocal, [100*(1-CI_bound)/tail, 100*(1-(1-
CI_bound)/tail)]);
paramCIs_nocal = [best_fit_params_nocal; best_fit_params_nocal-
CI_nocal(1,:); CI_nocal(2,:)-best_fit_params_nocal];
CI_cal = prctile(bootstrap_params_cal, [100*(1-CI_bound)/tail, 100*(1-(1-
CI_bound)/tail)]);
paramCIs_cal = [best_fit_params_cal; best_fit_params_cal-
CI_cal(1,:); CI_cal(2,:)-best_fit_params_cal];
param_CI_nocal =
array2table(paramCIs_nocal, 'RowNames', rowNames, 'VariableNames', paramIDs_nocal
)
param_CI_cal =
array2table(paramCIs_cal, 'RowNames', rowNames, 'VariableNames', paramIDs_cal)

```

d. Script_calculate_n_bootstraps.m

```
%This MATLAB code is associated with the following manuscript: Barrick,  
%S.K., S.R. Clippinger, L. Greenberg, M.J. Greenberg. 2019. Computational  
%tool to study perturbations in muscle regulation and its application to  
%heart disease.  
  
%This script tests whether a sufficient number of bootstraps have been  
%performed.  
  
%%  
%The input is a matrix of values obtained from bootstrapping  
(bootstrap_params). A large number of bootstraps should be used.  
  
input=bootstrap_params;  
  
ind=1; %This is the index of the parameter to be compared. For example,  
%if best_fit_params is ordered as (KW,KTnocal,KTcal,KTmidcal,nH,A,B,C),  
%then ind=1 selects KW, ind=2 selects KTnocal, etc.  
  
%These are the parameters for the confidence interval calculations.  
CI_bound=0.95; %This is the percent value for the confidence intervals.  
tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed  
%test. Use 2 by default.  
  
%This clears the variables used to make it easier to run the script  
consecutively  
%for multiple indices.  
clear data_subset data_mean CI_data_subset  
  
data=input(:,ind); %This selects the desired index of the output matrix  
  
%This loop calculates the mean value obtained if only i bootstraps were  
%run, along with the confidence intervals for this number of bootstraps.  
data_mean = zeros(length(data)-1, 0);  
CI_data_subset = zeros(2, length(data)-1);  
for i=2:length(data)  
    clear data_subset  
    data_subset = data(1:i);  
    data_mean(i-1) = mean(data_subset);  
    CI_data_subset(:,i-1) = prctile(data_subset,[100*(1-  
CI_bound)/tail,100*(1-(1-CI_bound)/tail)]);  
end  
  
x_axis=2:length(data); %This generates the x-axis.  
  
%This generates a plot. The x-axis is the number of bootstraps. The  
%black line is the mean parameter value from that number of bootstraps.  
%The red and blue lines show the upper and lower bounds, respectively, of the  
%95% confidence intervals. If a sufficient number of bootstraps have been  
%performed, the values of the mean and bounds of the confidence intervals  
%should change only minimally at large values of x (number of bootstraps).  
  
figure(1)  
hold off
```

```
plot(x_axis,data_mean,'k.')  
hold on  
plot(CI_data_subset(1,:), 'b')  
plot(CI_data_subset(2,:), 'r')
```


e. Script_hypothesis_testing.m

```
%This MATLAB code is associated with the following manuscript: Barrick,  
%S.K., S.R. Clippinger, L. Greenberg, M.J. Greenberg. 2019. Computational  
%tool to study perturbations in muscle regulation and its application to  
%heart disease.  
  
%This is a script for hypothesis testing. The input is two matrices, each  
%of which contain the best-fit parameters from the real data  
(best_fit_params)  
%and from the accompanying bootstrapping simulations (bootstrap_params) for  
%one of the two data sets to be compared (e.g., WT and mutant).  
  
%%  
  
%Change the names on the right side to the names of the variables in the  
workspace.  
A_best = best_fit_params_A;  
A_boot = bootstrap_params_A;  
B_best = best_fit_params_B;  
B_boot = bootstrap_params_B;  
  
tag='KW_WTvsdE160'; %When saving the data, this will be the name of the  
output.  
  
ind=1; %This is the index of the parameter to be compared. For example,  
%if best_fit_params is ordered as (KW,KTnocal,KTcal,KTmidcal,nH,A,B,C),  
%then ind=1 selects KW, ind=2 selects KTnocal, etc.  
  
alpha=0.05; %This is the threshold value for significance.  
null=0; %This is the value of the null hypothesis. In most cases, this is the  
hypothesis that A=B, implying A-B=0.  
tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed  
%test. Use 2 by default.  
  
clearvars -except best_fit_params* bootstrap_params* *_best *_boot ind tag  
alpha null tail  
  
close all  
  
A = A_boot(:, ind); %This selects the parameters to be examined.  
B = B_boot(:, ind);  
  
A(isnan(A))=[]; %This gets rid of blanks.  
B(isnan(B))=[];  
  
%This extracts the true values of A and B and calculates the test statistic  
%(i.e., the difference between A and B).  
out_data = [A_best(ind) B_best(ind)]; %Values obtained from raw data  
out_stat_data = A_best(ind)-B_best(ind); %This is the test statistic value.  
  
%This calculates the cumulative distributions and plots them in Figure 1.  
A_index = (1:length(A))./length(A);  
B_index = (1:length(B))./length(B);  
cumulative_A = sort(A);
```

```

cumulative_B = sort(B);

figure(1)
hold off
plot(cumulative_A,A_index,'k')
hold on
plot(cumulative_B,B_index,'r')

%This saves the plots. It can be commented out if desired.
saveas(gcf, strcat(tag, '_cumulative'), 'epsc'); %Option for saving cumulative
distributions
saveas(gcf, strcat(tag, '_cumulative'), 'fig')

%The following code calculates the test statistic for each bootstrapping
%simulation to evaluate whether there is a statistically significant
difference
%in the means of A and B.

%This creates a matrix of the parameter values obtained from the
%bootstrapping simulations.
clear out_sim

out_sim=[A B];

%This calculates the test statistic for the simulated data.
out_stat_sim = out_sim(:,1)-out_sim(:,2);
if median(out_stat_sim) <= 0 %This flips the curve for consistency.
    out_stat_sim=-out_stat_sim;
    out_stat_data=-out_stat_data;
end

%This calculates the confidence intervals for both the data and the test
%statistic.
CI_stat = prctile(out_stat_sim,[100*alpha/tail,100*(1-alpha/tail)]);
CI_data = prctile(out_sim,[100*alpha/tail,100*(1-alpha/tail)]);

%This makes the figure (Figure 2) showing the bootstrapped test statistic.
%Histogram shows the test statistic for all simulations. Red lines indicate
%the bounds of the 95% CI. The blue line indicates the null hypothesis. The
%yellow line marks the measured test statistic. If the blue line is outside
%of the boundaries defined by the red lines, then the null hypothesis is
rejected.
figure(2)
hold off
histogram(out_stat_sim(:,1), 'Normalization', 'probability')
hold on
plot(out_stat_data(1)*[1,1],ylim,'y-','LineWidth',2);
plot(CI_stat(1)*[1,1],ylim,'r-','LineWidth',2);
plot(CI_stat(2)*[1,1],ylim,'r-','LineWidth',2);
plot(null*[1,1],ylim,'b-','LineWidth',2);
xlabel('Difference between means');

set(gcf, 'renderer', 'Painters')
print(strcat(tag, '_hist'), '-depsc') %Option for saving the histograms
saveas(gcf, strcat(tag, '_hist'), 'fig')

```

```

%Next we get the p-value from the cumulative distribution of the test
statistic (Figure 3).

out_cumulative_mean_ind =
(1:length(out_stat_sim(:,1)))./length(out_stat_sim(:,1));
cumulative_mean = sort(out_stat_sim(:,1));

figure(3)
hold off
plot(cumulative_mean,out_cumulative_mean_ind,'k')
hold on
h1=plot(out_stat_data(1)*[1,1],ylim,'y-','LineWidth',2);
h2=plot(CI_stat(1)*[1,1],ylim,'r-','LineWidth',2);
plot(CI_stat(2)*[1,1],ylim,'r-','LineWidth',2);
h3=plot(null*[1,1],ylim,'b-','LineWidth',2);
xlabel('Difference between means');

saveas(gcf,strcat(tag,'_pval'),'epsc'); %Optional for saving cumulative
distributions
saveas(gcf,strcat(tag,'_pval'),'fig')

%This displays a table containing the true fit value and the -/+ values for
%the confidence interval (CI) for A and B, as well as the p-value for the
%comparison of A and B.
p_ind = find(cumulative_mean>=0,1);
p_value = out_cumulative_mean_ind(p_ind)*tail;

rowNames = {'value','-','+'};
colNames = {'A','B'};
output = [out_data(1),out_data(2);out_data(1)-CI_data(1),out_data(2)-
CI_data(3);
CI_data(2)-out_data(1),CI_data(4)-out_data(2)];
output_params =
array2table(output,'RowNames',rowNames,'VariableNames',colNames)
p_value

```

f. Script_mean.m

```
%This MATLAB code is associated with the following manuscript: Barrick,  
%S.K., S.R. Clippinger, L. Greenberg, M.J. Greenberg. 2019. Computational  
%tool to study perturbations in muscle regulation and its application to  
%heart disease.  
  
%This is a script to calculate the mean value for a population of objects  
%and the associated uncertainties.  
  
%%  
  
%Here, we input a range of measured values into x1.  
x1=data;  
  
%These are the parameters for the confidence interval calculations.  
N_bs=1000; %Number of bootstrapping simulations. 1000 is usually sufficient.  
CI_bound=0.95; %This is the percent value for the confidence intervals.  
tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed  
%test. Use 2 by default.  
  
%This removes any non-numerical values in the imported data set.  
q=find(isnan(x1));  
x1(q)=[];  
  
%This calculates the mean of the real data.  
best_fit_param=mean(x1);  
  
%The following code performs the bootstrapping simulations for error  
%estimation.  
%It outputs a matrix, bootstrap_param, that contains the mean value  
%for each of the simulated data sets. Each row contains the value obtained  
%from a separate simulation.  
  
%This randomly selects indices of the original data set used to  
%generate the simulated (i.e., bootstrapped) data sets.  
bs_ind=ceil(length(x1)*rand(length(x1),N_bs));  
  
%This generates fits to the N_bs simulated data sets. Each row of  
%bootstrap_param contains the mean for one round of resampling.  
bootstrap_param = zeros(length(data)-1, 1);  
  
parfor i=1:N_bs  
    x_ind=x1(bs_ind(:,i));  
  
    bootstrap_param(i,:)=mean(x_ind)  
end  
  
%This calculates the confidence intervals based on the bootstrapped data.  
%It returns param_CI. The first row contains the measured mean value. The  
%second row contains the distance from the lower bound of the CI to the  
%measured value, and the third row contains the distance from the upper  
%bound of the CI to the measured value.  
clear CI paramCI param_CI
```

```

rowNames = {'mean', '-', '+'};
CI = prctile(bootstrap_param, [100*(1-CI_bound)/tail, 100*(1-(1-
CI_bound)/tail)]);
paramCI = [best_fit_param; best_fit_param-CI(1); CI(2)-best_fit_param];
param_CI = array2table(paramCI, 'RowNames', rowNames)

```

g. Script_median.m

```
%This MATLAB code is associated with the following manuscript: Barrick,  
%S.K., S.R. Clippinger, L. Greenberg, M.J. Greenberg. 2019. Computational  
%tool to study perturbations in muscle regulation and its application to  
%heart disease.  
  
%This is a script to calculate the median value for a population of objects  
%and the associated uncertainties.  
  
%%  
  
%Here, we input a range of measured values into x1.  
x1=data;  
  
%These are the parameters for the confidence interval calculations.  
N_bs=1000; %Number of bootstrapping simulations. 1000 is usually sufficient.  
CI_bound=0.95; %This is the percent value for the confidence intervals.  
tail=2; %This defines whether the p-value is obtained for a 1- or 2-tailed  
%test. Use 2 by default.  
  
%This removes any non-numerical values in the imported data set.  
q=find(isnan(x1));  
x1(q)=[];  
  
%This calculates the median of the real data.  
best_fit_param = median(x1);  
  
%The following code performs the bootstrapping simulations for error  
%estimation.  
%It outputs a matrix, bootstrap_param, that contains the median value  
%for each of the simulated data sets. Each row contains the value obtained  
%from a separate simulation.  
  
%This randomly selects indices of the original data set used to  
%generate the simulated (i.e., bootstrapped) data sets.  
bs_ind=ceil(length(x1)*rand(length(x1),N_bs));  
  
%This generates fits to the N_bs simulated data sets. Each row of  
%bootstrap_param contains the median for one round of resampling.  
bootstrap_param = zeros(length(data)-1, 1);  
  
parfor i=1:N_bs  
    x_ind = x1(bs_ind(:,i));  
  
    bootstrap_param(i,:)=median(x_ind)  
end  
  
%This calculates the confidence intervals based on the bootstrapped data.  
%It returns param_CI. The first row contains the measured median value. The  
%second row contains the distance from the lower bound of the CI to the  
%measured value, and the third row contains the distance from the upper  
%bound of the CI to the measured value.  
clear CI paramCI param_CI
```

```

rowNames = {'median', '-', '+'};
CI = prctile(bootstrap_param, [100*(1-CI_bound)/tail, 100*(1-(1-
CI_bound)/tail)]);
paramCI = [best_fit_param; best_fit_param-CI(1); CI(2)-best_fit_param];
param_CI = array2table(paramCI, 'RowNames', rowNames)

```