

APPENDIX A

Network Signaling Model: Adaptable Script for Simulating and Plotting Fold-Change

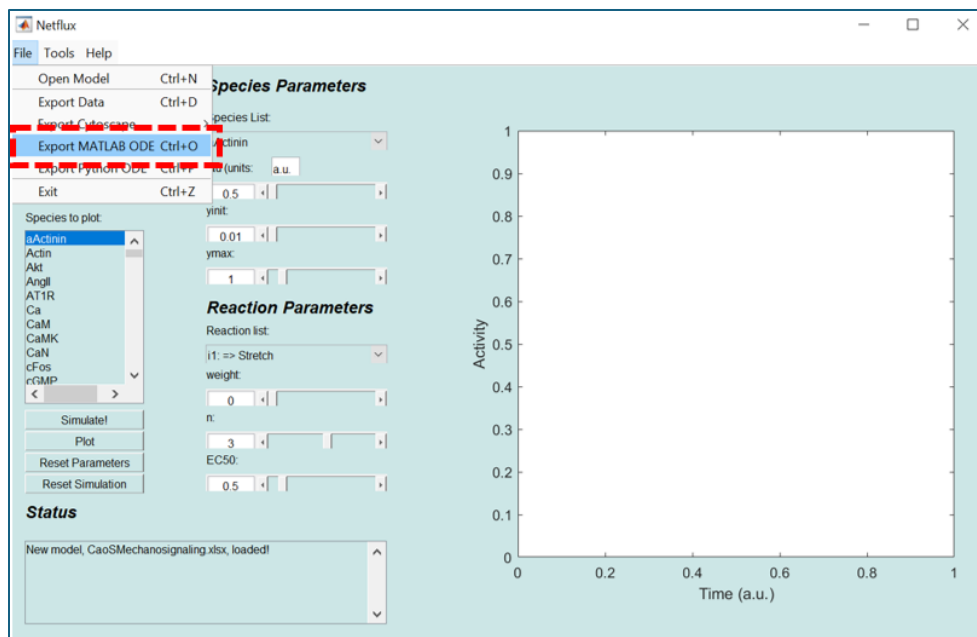
Guidelines for Implementation:

I. NETFLUX

- Go to: <https://github.com/saucermanlab/Netflux>
- Download the repository and follow the instructions/tutorials on implementing Netflux.
- Run** Netflux in command window and open your model in the GUI
 - If not done so already, create your own signaling model that is compatible with Netflux syntax (refer to tutorials for guidance)
 - Ensure that, in the species sheet of your parameter sheet, there is an identifier for gene or FC-type species, such as 'Gene_XXXX':

121	Gene35	gene_Atf3	0.01	1	188.2
122	Gene36	gene_Atic	0.01	1	553
123	Gene37	gene_Atp13a3	0.01	1	649.3
124	Gene38	gene_Atp1a2	0.01	1	817.4
125	Gene39	gene_Atp2a2	0.01	1	380.2
126	Gene40	gene_Atp2b1	0.041	1	453.2
127	Gene41	gene_Atp2b4	0.01	1	337.2
128	Gene42	gene_Atp6v0b	0.82	1	759.5

- Click 'Export MATLAB ODE' and select the destination folder for the functions:



- This will create three files in your destination folder:

- NetfluxODE_Example
- NetfluxODE_Example_loadParams
- NetfluxODE_Example_run

- ## II. NORMALIZATION SCRIPT

- ### Netflux Parameter Import Function (NetfluxODE_loadParams.m)

Copy and paste the contents of the 'NetfluxODE_loadParams.m' file here (make sure to add 'end' at the end of the fu

ii. Importantly, comment-out the fact function in your NetfluxODE function, as it gets replaced by the safe version in the file:

1. What this does is implement the same safeguards that Netflux does to avoid complex numbers at low stimulus values, preventing issues when solving the system.

Adding Epsilon floor to ODEsolver to prevent complex values:

Comment out the fact utility function at the bottom of the NetfluxODE function and use this instead.

This will incorporate the safeguards that Netflux has coded in to prevent complex values from occurring at low input stimuli:

```
function fact = act(x, rpar)
    w = rpar(1);
    n = rpar(2);
    EC50 = rpar(3);
    beta = (EC50.^n - 1) ./ (2 * EC50.^n - 1);
    K = (beta - 1).^(1 ./ n);

    epsilon = 1e-6; % Small constant floor
    x_safe = max(x, epsilon);

    fact = w .* (beta .* x_safe.^n) ./ (K.^n + x_safe.^n);
    if fact > w
        fact = w;
    end
end
```

- iii. Adjust NetfluxODE command to accept your S_in stimulus by locating the old activation function for the stimulus and changing the rpar(1, X) to S_in:
 1. Note: If your stimulus has feedback, then adjust only the reaction weight for the true stimulus, not the feedback.

```
dydt(Stretch) = (S_in*ymax(Stretch) - y(Stretch))/tau(Stretch);
% dydt(Stretch) = (rpar(1, 1)*ymax(Stretch) - y(Stretch))/tau(Stretch);
```

Original Stimulus ODE: $\text{dydt}(\text{Stretch}) = (w \cdot \text{ymax}(\text{Stretch}) - y(\text{Stretch})) / \text{tau}(\text{Stretch});$

Adjustable Stimulus ODE: $\text{dydt}(\text{Stretch}) = (S_in \cdot \text{ymax}(\text{Stretch}) - y(\text{Stretch})) / \text{tau}(\text{Stretch});$

Function Handle:

```
function dydt=NetfluxODE(t,y,params)
```

New Function Handle:



```
function dydt=NetfluxODE(t,y,params, S_in)
```

- iv. Adjust target genes as desired. This is simply for plotting purposes to avoid cluttering the plots with excessive profiles:

```
% Targets chosen arbitrarily since Matlab cannot plot 50+ timecourses
% without breaking
targets = {'gene_Adamts9', 'gene_Magi1', 'gene_Nr4a3', 'gene_Fgfr2', 'gene_Per1', 'gene_Csrp3', 'gene_Lmcd1'};
targetsLegend = erase(targets, 'gene_'); % removes qualifiers for legend clarity, replace if necessary
% change to your own
```

- v. Input Stimuli: Adjust according to baseline/activated protocol specifications:
1. The values in the brackets [a, b] are default for reference, but slider values are the ones actually run in the model, so make sure those are the ones that are properly populated.

```
% Input Stimuli
S_in = [0.400, 0.7]; % Stretch Inputs (Baseline, Stimulated), change as necessary

% Stimuli Sliders: Use these if you want to quickly observe impacts of
% changing baseline/activated stimuli (changing either value automatically
% runs the model)
S_in(1) = 0.4  ; % Baseline Stimulus: Slider for quick adjustment
S_in(2) = 0.7  ; % Activated Stimulus: Slider for quick adjustment
```

- vi. Stimulus index parsing: Replace the 'Stretch' in the code with the name of your stimulus to parse for the species index for stimulus

```
% Identify Stimulus and Gene indices
stimIdx = find(strcmp(speciesNames, 'Stretch')); % change 'Stretch' to the name of your stimulus species
```

- vii. Gene index parsing: This will depend on how you have placed your identifier:

```
% If NETFLUX exports only speciesNames and your species IDs column contains
% the identifier use this:

% geneIdx = find(contains(speciesNames, 'Gene')); % ensure 'ID' column has 'Gene' or some sort of identifier for Fold-Change terms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% If your identifier is in speciesTypes but NETFLUX does not export this,
% use this:

% filename = 'RxnParsed_CaoSMechanosignaling.xlsx'; % Replace with your Netflux paramsheet title
% types = readcell(filename);
% speciesTypes = types(3:end, 1);
% geneIdx = find(contains(speciesTypes, 'Gene')); % ensure 'module' column has 'Gene' or some sort of identifier for Fold-Change terms
```

1. If within species name (ID column of the species sheet), just uncomment the first command and replace 'Gene' with your identifier.

2. If within species types (module column of the species sheet), un-comment the second corresponding set of commands which reads in your parameter-sheet to extract gene indices.
 3. If your Netflux has the capability to extract speciesTypes (which is unlikely), un-comment the third set of the commands.
- viii. Time-scale adjustment: The baseline timescale should be sufficiently long to reach a quasi-steady-state, but adjust the activated stimulus timescale to reflect experimental conditions:

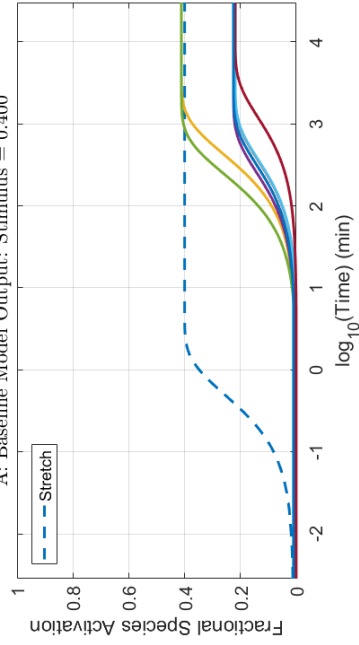
```
% Defining Time-Scale
tspan = 0:0.1:(24*60); % used for experimental/activated simulation timecourse
tspan1 = [0 30000]; % used for baseline simulation, should be sufficiently long to allow for QSS baseline
```

- ix. ODENorm function: Scroll down to the functions part of the script and locate the ODENorm function. This function normalizes your activated simulation by the baseline output values. Adjust the outputs to the desired time checkpoints (they default to 30 mins and 4 hours):

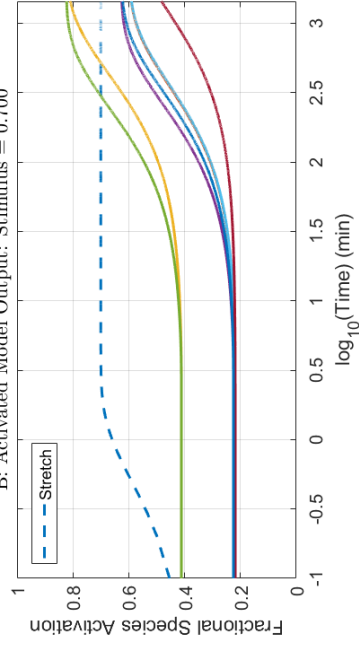
```
% Normalization of Activated gene expression by Baseline value (FC)
function [yout_4h, yout_30m, ynorm] = ODENorm(t_in, y_in, ss, geneIdx)
ynorm = y_in;
ynorm(:, geneIdx) = y_in(:, geneIdx) ./ ss(geneIdx)';
yout_4h = ynorm(find(t_in >= 240, 1), :); % adjust this and the following line according to experimental timescale
yout_30m = ynorm(find(t_in >= 30, 1), :);
end
```

- x. For importing data, ensure the lengths of the imported data are the same as model outputs to avoid dimensioning errors. One tip is to simply add in an import function to the bottom of the script to readmatrix('filename.xlsx') for your datasheet.
- xi. Error Metrics: Add in any additional error matrices that are needed, especially if comparing with data:
 1. Defaults: Avg(FC), Avg(log2(FC)), Frequency Distribution (Model)
- xii. Adjust any figure titles as necessary
- xiii. Run the model and enjoy:

A: Baseline Model Output: Stimulus = 0.400



B: Activated Model Output: Stimulus = 0.700



C: Normalized Gene Expression (Fold-Change)

