Exercises for Chapter 3, The Critical Point

1. **The phase transition point**

   One of the hallmarks of a continuous phase transition is that an order parameter continuously changes value as a function of a control parameter. The region where the order parameter rapidly changes can be used to identify the phase transition point. To illustrate this, Figure 3.3 in the book shows a plot of firing rate (order parameter) against the constructed branching ratio (control parameter) for a simple branching model network. Another signature of a continuous phase transition is that some functions will show sharp peaks near the phase transition point. For example, Figure 4.5 in the book shows that the susceptibility of a branching model has a peak near the phase transition region.

   A. *Phase plot*: In these exercises, we will see if similar plots can be made with a spiking neural network model. The well-known Brunel model (Brunel, 2000, Journal of Computational Neuroscience) consists of 80% excitatory and 20% inhibitory integrate and fire neurons sparsely connected to each other (at 10% connectivity). Jonathan Touboul and Alain Destexhe made a computational model of this network (Destexhe and Touboul, 2021, eNeuro) with open source code. I have modified their code to create the Matlab function BrunelNetworkFunctionTimeConstant. In this version of the model there is no random background activity driving the network. Rather, the network is stimulated once by simultaneously activating a fraction of the neural population.

   Exercise: Using this function, record the average firing rate ("FR," the order parameter) of the Brunel model under different values of the control parameter, "g." To do this, set "ratio" = 1, "Jc" = 1 and vary "g" from 0 to 5 in small steps (say 0.25).

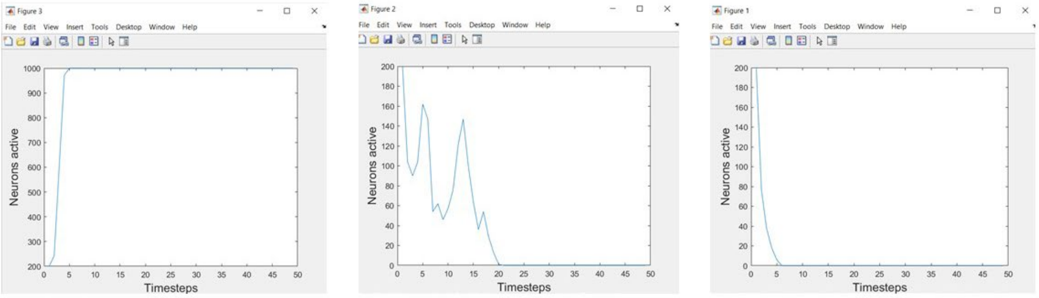
   Use this line of code to get the FR for a single set of input values:

   ```
   [A chi FR] = BrunelNetworkFunctionTimeConstant(gee, ratio, Jc)
   ```

   Based on this plot of FR vs g, where would you predict the phase transition to be (in terms of the value of "g")?

   B. *Peak in time constant*: Continuing with the Brunel model, again use BrunelNetworkFunctionTimeConstant (with "ratio" = 1, "Jc" = 1) to plot the network activity over time in response to a single stimulation. This information is given by plotting the output variable "A" (see the figure below for examples). You can get such plots by using these lines of code:

   ```
   figure; plot(A); xlabel("Timesteps"); ylabel("Neurons active");
   ylim([0 1000]);
   ```

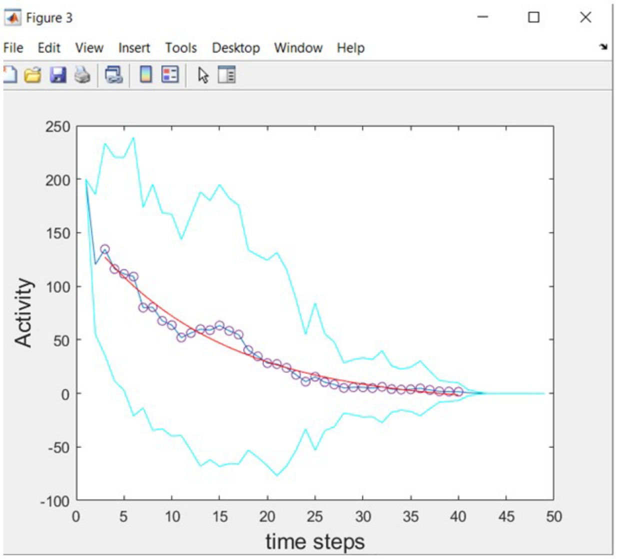


Plotting the activity of the network, variable A, against time. This is done for different values of g, the strength of inhibition

Exercise: How does the response to stimulation change as the control parameter "g" is increased, like in the previous exercise? Note that the network activity can quickly rise, quickly decay, or slowly decay. To quantify the time of this change, run the script BrunelTimeConstantFinal to fit exponential curves to the average of 30 stimulations for each value of "g." Note: this requires the Matlab Curve Fitting Toolbox, so don't do this if you do not have the toolbox. To use the Brunel network simulation, just type this in the Matlab Command Window:

```
BrunelTimeConstantFinal;
```

You should get several plots that look like this:



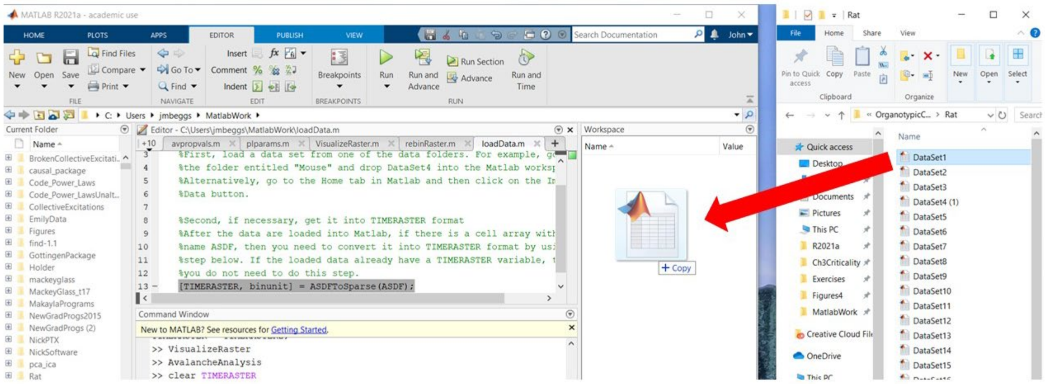
One representative plot produced after the script BrunelTimeConstantFinal was run. This shows an exponential fit to 30 runs of the Brunel model after stimulation. The resulting decay constant, DC, was 13.8 time steps.

You should also get a plot of the time constants found vs “g” (not shown here – for you to find out).

2. Looking at data

The book discusses how avalanches from a network near the critical point should approximately follow power law distributions (page 56, Figure 3.4 for models; page 66, Figure 3.11 for experimental data). For example, Figure 3.11 shows distributions plotted in log-log coordinates for avalanche size, duration and average size vs. duration. These distributions should have regions that can be fit by straight lines if the network is near the critical point. For these exercises, data of several different types are available: (1) Spiking data from cortical slice cultures placed on a 512-electrode array, (2) Spiking data from dissociated cultures grown on 60-electrode arrays, (3) Local field potential (LFP) data from cortical slice cultures placed on 60-electrode arrays. These data are recordings of spontaneous, unstimulated activity. We will examine the data’s structure and then work our way toward plotting power law distributions and examining other signatures of operating near the critical point.

C. *Structure of neural network activity*: We will first look at the structure of the data at long-time scales and at short time scales. Select one of the data sets from the folders provided. For example, go into the folder “OrganotypicData” and then the subfolder “Mouse” and load “DataSet4” into the Matlab workspace. You can do this by dragging and dropping it, as in the screenshot below. Alternatively, go to the Home tab in Matlab and then click on the Import Data button.



Editor - C:\Users\jmbeggs\MatlabWork\loadData.m

```
%First, load a data set from one of the data folders. For example, g
%the folder entitled "Mouse" and drop DataSet4 into the Matlab works
%Alternatively, go to the Home tab in Matlab and then click on the In
%Data button.

%Second, if necessary, get it into TIMERASTER format
%After the data are loaded into Matlab, if there is a cell array wit
%name ASDF, then you need to convert it into TIMERASTER format by us
%step below. If the loaded data already have a TIMERASTER variable, t
%you do not need to do this step.
[TIMERASTER, binunit] = ASDFToSparse(ASDF);
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> VisualizeRaster
>> AvalancheAnalysis
>> clear TIMERASTER
```

Second, if necessary, get the data into TIMERASTER format. (1) If the data are from the “LFP60Data” folder, they already have a TIMERASTER variable present after loading and you do not need to do this step. (2) If you are loading data from the “DissociatedCultures” folder, you should use the function Dissociated2TIMERASTER to produce a TIMERASTER for analysis, like this:

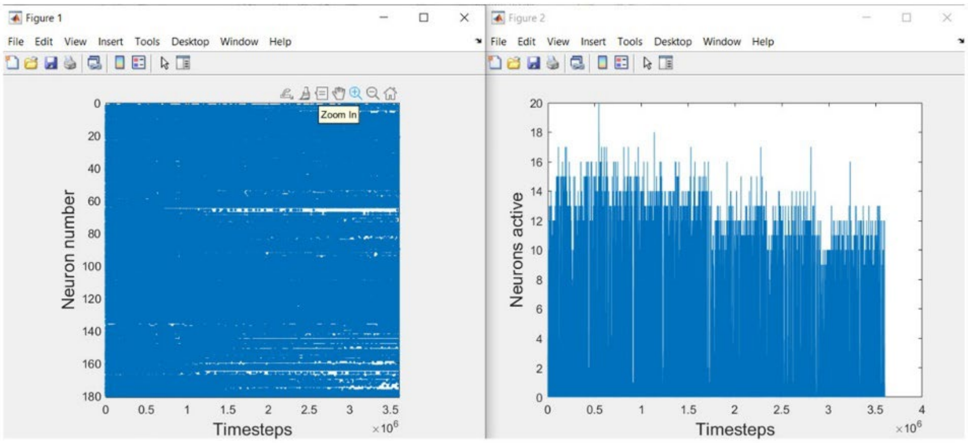
```
[TIMERASTER] = Dissociated2TIMERASTER(data);
```

(3) If you are using data from the “OrganotypicCultures” folder, after the data are loaded into Matlab you should see a cell array with the name ASDF. To convert this cell array into TIMERASTER format, use the function ASDFToSparse like this:

```
[TIMERASTER, binunit] = ASDFToSparse(ASDF);
```

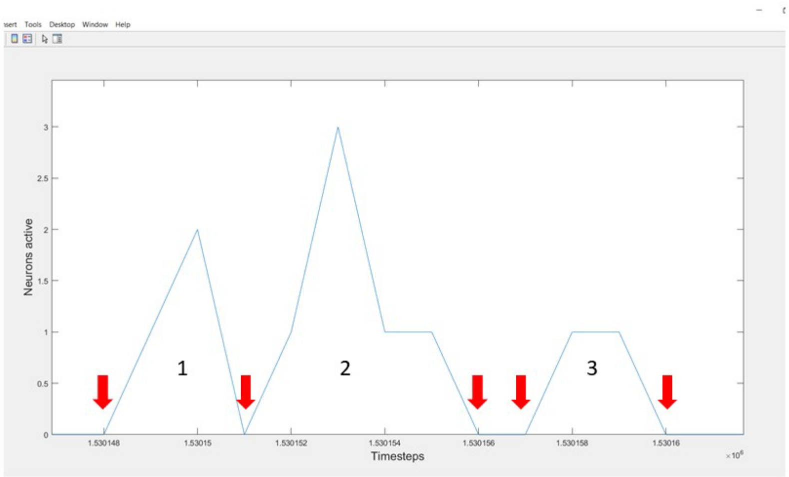
Once you have a TIMERASTER in your Matlab workspace you can use the function VisualizeRaster to plot the data in raster form (neuron number on y-axis, time on x-axis, blue dots represent spikes or suprathreshold LFPs). Most recordings are 1 hr long and so should have 3.6×10^6 time steps when the time bins are 1 ms long. Use the function like this:

```
VisualizeRaster(TIMERASTER);
```



Left, an example raster plot of activity. Each blue dot is a spike or an LFP from an electrode, depending on the data being plotted. To zoom in, click on the positive magnifying glass icon at the top of the figure. Right, the population activity at each time step. This figure may also be enlarged to show individual avalanches.

Use the zoom tool on the figure to look at a network burst at shorter time scale. This script will also plot the population activity at each time step, which is just the number of neurons active in each time bin. If you zoom into this plot, you should be able to see an individual avalanche where there is a series of consecutively active time bins bracketed by inactive time bins.



Viewing the population activity at a much smaller temporal scale reveals brief episodes that are bracketed by no activity (red arrows). Each such period is an avalanche. In this figure, there are three complete avalanches. The number of neurons active in each time step of avalanche 2 is: 1, 3, 1, 1. Thus the size of avalanche 2 is 6 (= 1+3+1+1) and its duration is 4.

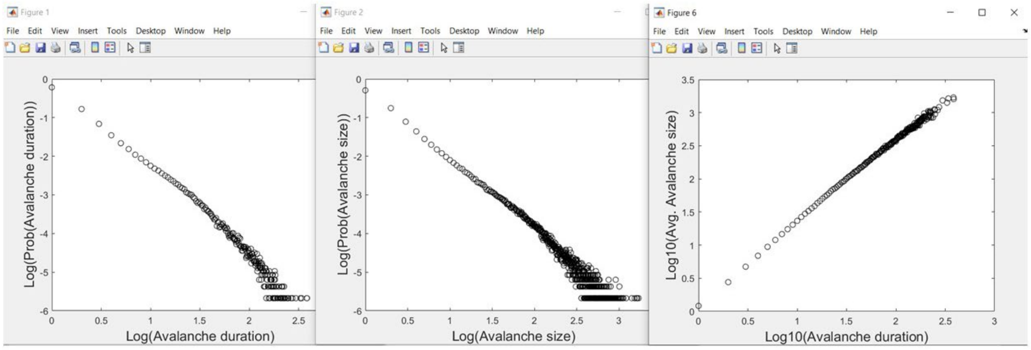
Exercise: Describe how the temporal structure differs between spike data and LFP data. How does the temporal structure differ between spiking data recorded from organotypic cultures (using the 512 electrode array) and spiking data recorded from dissociated cultures (using the 60 electrode array)? Notice that for very large numbers of neurons, it becomes more difficult to find an inactive time bin. Under these circumstances, can you suggest how avalanches be defined?

D. *Avalanche distributions*: Using the function AvalancheAnalysis, plot the distributions for avalanche size, duration and average size for a given duration. Use it like this:

```
[sizeDist, durationDist, SvsT, Events] = AvalancheAnalysis(TIMERASTER);
```

Save the output variables `sizeDist`, `durationDist`, `SvsT`, and `Events` for later use (e.g., see part G below).

Exercise: Do all the distributions show sharp power laws? Can you identify distributions that look slightly subcritical or slightly supercritical (see Figure 3.10)? Take a random sample of 20 data sets. What fraction look like they have straight power law regions? Example plots are shown below.



Plots in log-log coordinates of the avalanche size, duration and size vs duration distributions.

E. *Rebinning the data*: Many of the spike data sets are binned at 1 ms resolution. This resolution may be too small for some data sets, because the time it takes for activity to propagate from one recording site to the next nearest site may be greater than 1 ms. If this is the case, then the data will be artificially fragmented. Under these conditions, the avalanche distributions will curve downward, presenting as subcritical. What is needed is a bin width that approximately matches the average propagation time between adjacent recording sites. To find this proper bin width, you can use the function rebinRaster like this:

```
[TIMERASTER2] = rebinRaster(TIMERASTER, 2);
```

This takes a “TIMEASTER” originally binned at 1 ms resolution and creates “TIMERASTER2” binned at 2 ms resolution. Gradually increase the bin width until you see nearly straight power laws. This works for data that show signatures of criticality but will not work for data that are truly subcritical (as we will see later). For more details, see this paper: (Notarmuzi, Daniele, Claudio Castellano, Alessandro Flammini, Dario Mazzilli, and Filippo Radicchi. "Universality, criticality and complexity of information propagation in social media." *Nature communications* 13, no. 1 (2022): 1-8.)

Exercise: Now, if you go back to your 20 randomly sampled data sets and re-bin them at larger values, what fraction of them appear to have straight power law segments when passed through the AvalancheAnalysis function? If you re-bin by a factor of 20 or 30, can you make the avalanche distributions look supercritical, like what is shown in Figure 3.10 C of the book?

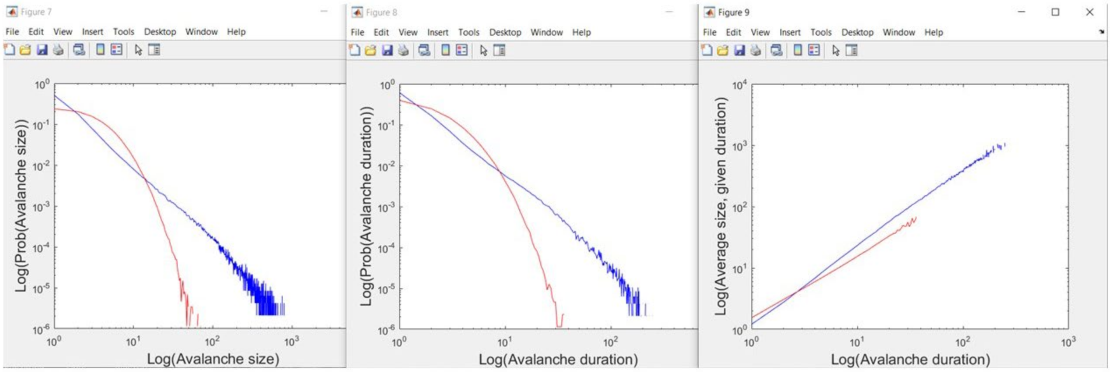
F. *Temporal structure (shuffling)*: The avalanche size distributions reflect the propagation of activity through neural networks at relatively short time scales. It would stand to reason then that disrupting the temporal structure of these avalanches should also disrupt their distributions. Use the timeShuffle function to randomly shuffle the data by maintaining the same number of events in each channel but randomizing when they occur in time. You can use it like this:

```
[TIMERASTERshuf] = timeShuffle(TIMERASTER);
```

Temporal structure (jittering): Use the <u>JitterRaster</u> function to jitter the event times by different amounts (e.g., sigma = 2000 bins, which would be 2 s if the data are binned at 1 ms resolution). “sigma” is the standard deviation of a normal distribution from which the jitter times are drawn. This is a milder form of temporal shuffling when the choice of “sigma” is smaller than 2 s. You can use the function like this:

```
[JRASTER] = JitterRaster(TIMERASTER, sigma);
```

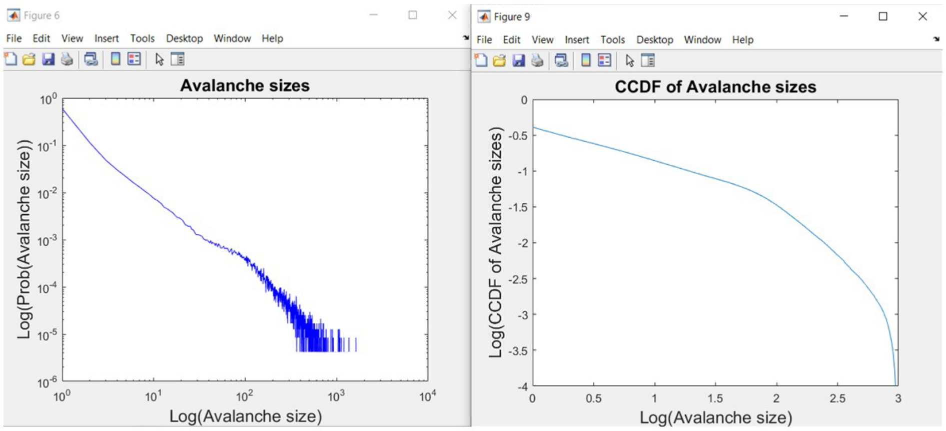
Exercise: After using shuffling or jittering, now plot the disrupted avalanche distributions (again using <u>AvalancheAnalysis</u>) and compare them to those from the original data. What differences do you notice? Example plots are shown below. Can you explain why these differences occur? What is the largest value of “sigma” for which these plots still look about the same?



Plots of avalanche distributions from unshuffled data (in blue) and shuffled data (red).

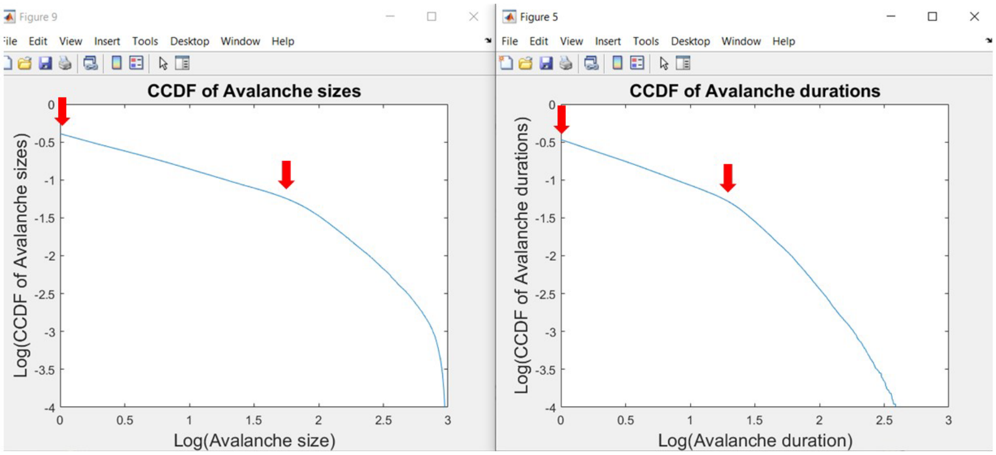
Another question: Can shuffled or jittered data that appears to be subcritical somehow be “rescued” and made critical again? To try this, take a disrupted data set (shuffled or jittered sufficiently) that has a downwardly curving avalanche size distribution. Now re-bin it by some factor (e.g., 2, 4, 10, 20) to see if this curve can be straightened out and made to look like a power law. Is this successful? Why or why not? What are the implications of your results for data interpretation?

G. *Exponent relation*: Another signature of being near the critical point is the exponent relation (page 66). Here, we will attempt a very quick way to assess if this is satisfied. We will explain a more rigorous and time-consuming way later. For this quick check, we will transform the original avalanche distributions into a format where they show less variability; this will allow us to discern regions for power law fitting more clearly. To make this transformation, we will construct complementary cumulative distribution functions (CCDFs) for each original distribution. Briefly, the CCDF is based on the cumulative sum of the distribution up to each value of x. This cumulative sum smooths out the bumpiness that is apparent in the original distributions. In this way, it reduces the variability. Here is an example of what it looks like:



At left is an avalanche size distribution. Notice that it is bumpy in parts. At right is its complementary cumulative distribution function (CCDF). Notice that it is much smoother, making straight line regions easier to identify

If we obtain the CCDFs for both avalanche size and duration distributions, it allows us to select domains where they appear to have approximately straight-line regions.



Estimating the range of the power law fit. Both plots show complementary cumulative distribution functions (CCDFs). These are used to increase the accuracy of power law fitting. The range over which the CCDF appears linear can be used to estimate the range over which a power law fit may be attempted. The red arrows denote a range for the avalanche size CCDF on the left and for the avalanche duration CCDF on the right.

Use the function GetCCDFs to obtain CCDFs for both avalanche size and duration distributions:

```
[CCS, CCD] = GetCCDFs(sizeDist, durationDist);
```

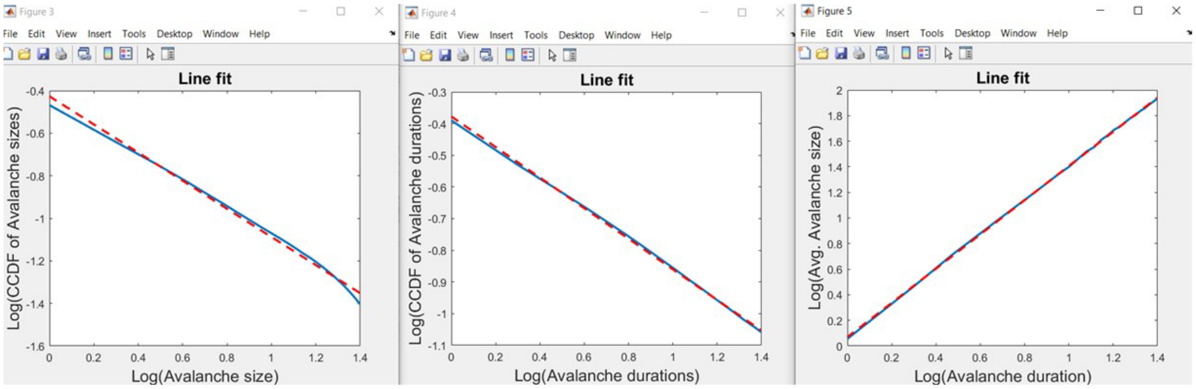
Once you have these, discern a straight-line region from the CCDF plots, if there is one. This will allow you to pick a lower limit and an upper limit over which power law fittings will be attempted. For the examples shown in the figure above, it looks like the limits for the size distribution would be [10^0, $10^{1.6}$] = [1, 40]; for the duration distribution they would be [10^0, $10^{1.4}$] = [1, 25]. Therefore, the domain over which we could expect scaling to apply would to both distributions would be the intersection of these two, from 1 to 25.

To see if the exponent relation can now be fit, use the function <u>ExponentRelation</u> like this (providing as inputs the variables you obtained earlier: `CCS, CCD, SvsT`):

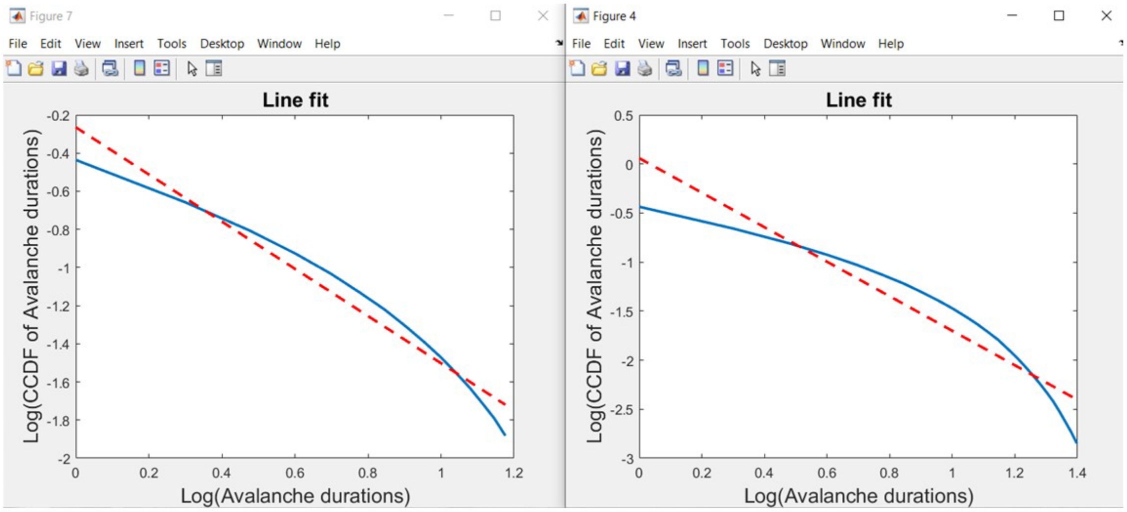
```
[alpha, tau, gamma_est, gamma_act, error] = ExponentRelation(CCS, CCD, SvsT, LimL, LimU);
```

Where “LimL” is the lower limit and “LimU” is the upper limit. The function will plot linear fits to the CCDFs over this domain and will return the estimated exponents “alpha,” “tau,” “gamma_est” (gamma estimated from the exponent relation), and “gamma_act” (gamma estimated from the size vs duration plot).

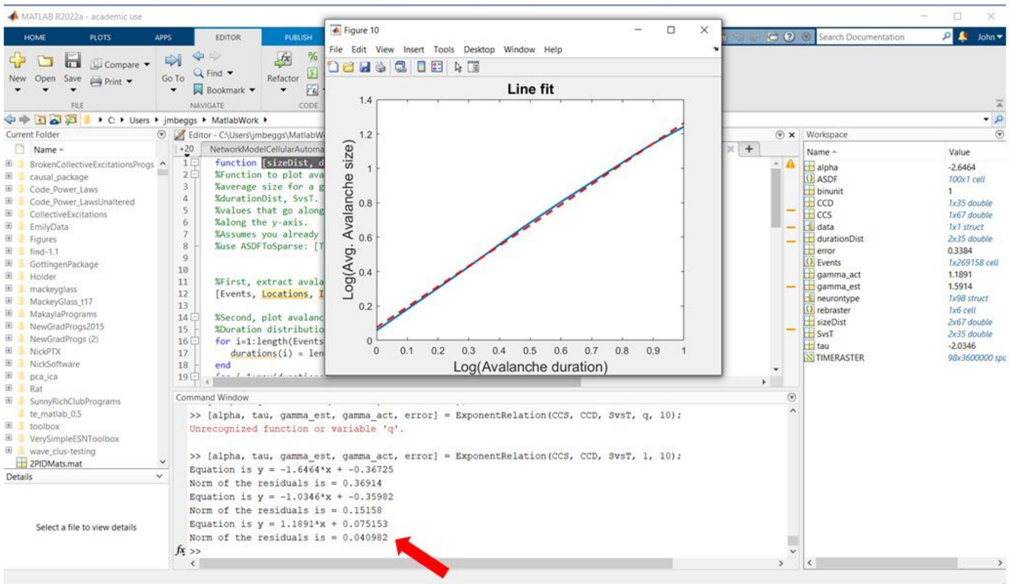
You can tell by looking at the plots if the linear fits are reasonable or not. For example, in the first figure below they are reasonable, while in the second they are not. The fit quality can be quantified by reading the norm of the residuals given by the least squares fit, which the function puts out for each fit. For the fits in the first figure below, these values are 0.106, 0.026, 0.031, all of which are acceptably low. When they are larger, the fits are poor, as shown in the second figure below. This should also be evident by looking at them. In that case, you should consider another region of the CCDF curve for fitting. Alternatively, the data may need to be re-binned. If this does not improve the fit, the data are perhaps not nearly critical; not all data sets are.



Line fits to the CCDFs of avalanche size (left), avalanche duration (middle) and the average size for a given duration vs the duration (right). All fits were conducted over the range specified by the red arrows chosen in the CCDF plots. Data set used was Mouse18. The norm of the residuals from the least squares fits were 0.106, 0.026 and 0.031, respectively (left to right).



Examples of poor line fits to the CCDFs of avalanche duration. First, they do not look like good fits. Second, we can quantify this poor quality with the norm of residuals, an output of the function ExponentRelation. The left plot had a norm of residuals of 0.345, while the right plot had 1.106.



```
function [sizeDist, d
%Function to plot ava
%average size for a g
%durationDist, SvsT.
%values that go along
%along the y-axis.
%Assumes you already
%use ASDFToSparse: [T


%First, extract avala
[Events, Locations, I

%Second, plot avalanc
%Duration distributio
for i=1:length(Events
   durations(i) = len
end
```

Name	Value
alpha	-2.6464
ASDF	100x1 cell
binunit	1
CCD	1x35 double
CCS	1x67 double
data	1x1 struct
durationDist	2x35 double
error	0.3384
Events	1x269158 cell
gamma_act	1.1891
gamma_est	1.5914
neurontype	1x98 struct
rebraster	1x6 cell
sizeDist	2x67 double
SvsT	2x35 double
tau	-2.0346
TIMERASTER	98x3600000 spc

```
>> [alpha, tau, gamma_est, gamma_act, error] = ExponentRelation(CCS, CCD, SvsT, q, 10);
Unrecognized function or variable 'q'.

>> [alpha, tau, gamma_est, gamma_act, error] = ExponentRelation(CCS, CCD, SvsT, 1, 10);
Equation is y = -1.6464*x + -0.36725
Norm of the residuals is = 0.36914
Equation is y = -1.0346*x + -0.35982
Norm of the residuals is = 0.15158
Equation is y = 1.1891*x + 0.075153
Norm of the residuals is = 0.040982
>>
```

The best fit line to the data will also have something called the "Norm of the residuals" that is printed out in the Matlab command window, as shown by the red arrow. Good fits, like the one shown here, generally have low values.

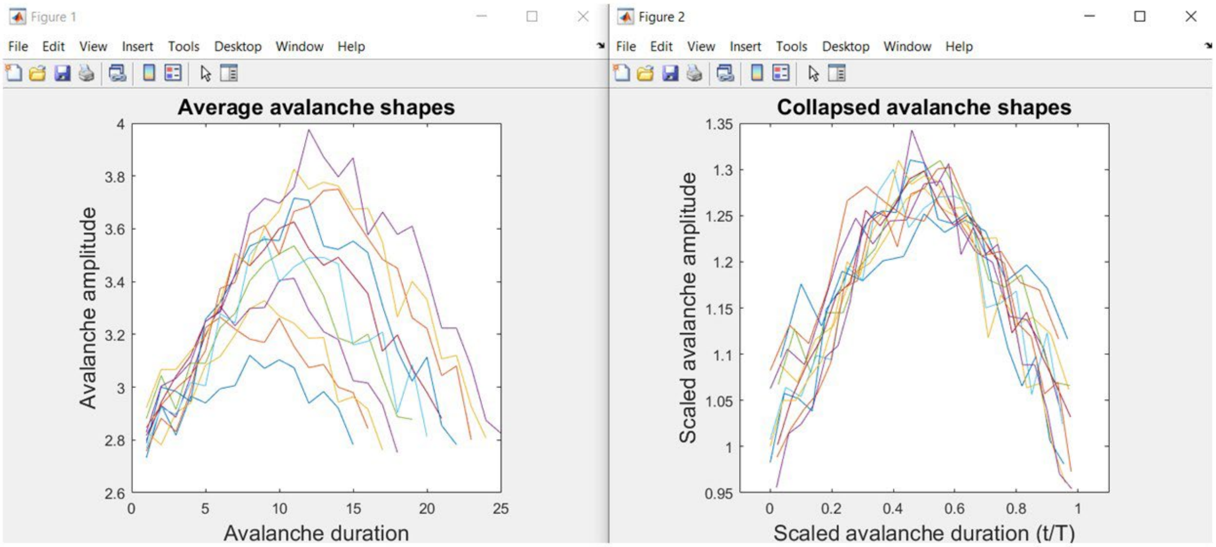
Finally, <u>ExponentRelation</u> will return another value, called "error" as a measure of how close "gamma_act" and "gamma_est" are to each other. An accepted value in the literature is for the "error" ≤ 0.20 (Ma et al., 2019). This is a measure of how well the exponent relation is satisfied.

Exercise: Using the exponents "alpha" and "tau," plot 10 data sets on the alpha, tau plane like what is shown in the book (Figures 3.15, 5.3). Do they lie along a line whose slope matches that given by "gamma_act?" What happens to these exponents if the data that produced them is shuffled or jittered? Plot "alpha" and "tau" for the same data set after increasing amounts of jitter (say, "sigma" = 10, 100, 1000, 10000).

H. Avalanche shape collapse is another indication that a system is operating near the critical point. Using the function ShapeCollapse, perform avalanche shape collapse on the data you examined previously.

```
[Shapes] = ShapeCollapse(Events, LimL, LimU, interval, gamma_act);
```

Here, the data structure Events was previously produced by AvalancheAnalysis, and `interval` is just a variable that allows you to plot every other avalanche if it equals 2, or every avalanche if it equals 1. Sometimes the plots get too dense if all avalanche shapes are included.



Avalanche shape collapse. Left plot shows avalanches of durations 15 to 25, interval = 1, without rescaling. Right plot shows rescaled avalanches.

Exercise: Find several Organotypic data sets that show good collapse. Next, jitter these data sets by increasing amounts (try "sigma" = 2, 5, 10, 20, 50, 100 ms). At what value of jitter does the collapse typically fall apart? For comparison, note also how the avalanche distributions look after jittering by these amounts. Which signature of criticality is the most fragile to jittering: Power laws? Exponent relation? Shape collapse? Explain why you think this is the case.

I. *A tunable branching model*: The book uses the simple branching model to illustrate many of the concepts surrounding criticality in neural networks. In fact, the first half of Chapter 3 is devoted to showing how this tunable model can show multiple signatures of criticality. The second half of the chapter is devoted to seeing if these signatures are present in the experimental data. In these exercises, things are run in reverse. Now that you have explored signatures of criticality in the data, you will see if you can produce these signatures when a simple model is appropriately tuned. The function for simulating the branching process is called BranchingModel. It takes as inputs the number of timesteps the simulation is to be run (“timesteps”), the rows (“r”) and columns (“c”) of the sheet of neurons to be simulated (r × c = number of neurons), the probability of each neuron spontaneously firing (“p”), the refractory period of each neuron after firing (“refperiod”), the number of connections each neuron is to have (“connects”), the constructed branching ratio for the network (“BR”), as well as the exponent by which the network transmission probabilities will decay when they are listed in descending order (“beta”). Sample values are shown below.

```
[TIMERASTER] = BranchingModel(timesteps, r, c, p, refperiod, connects, BR, beta);
```

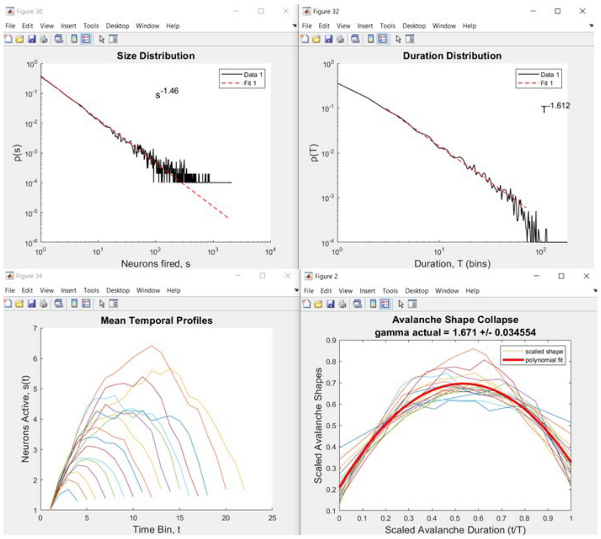
```
[TIMERASTER] = BranchingModel(1000000, 12, 12, 0.0001, 10, 10, 1.0, 0.5);
```

Running the model for 1 million time steps, as suggested here, took about 20 seconds on my laptop (Intel Core i7-8750H CPU @ 2.20 GHz, 16 GB Ram, 64 bit operating system, x64-based processor). While more time steps and more neurons produce better statistics, you should explore what run times are reasonable for you, given your equipment.

Exercise: Plot avalanche distributions for the model as you sweep “BR” from 0.5 to 1.5 by increments of 0.1. If you save the data, you can perform multiple analyses later (see next exercise). Report on the following: The quality of linear fits to the avalanche distributions (if possible), the error in the exponent relation (if possible), and the quality of avalanche shape collapse. Note that we will examine finite size effects in the exercises for the next chapter.

For each value of “BR,” calculate the total number of active neurons from the TIMERASTER output – you can get this by just taking the double sum of timeraster: TotalNum = sum(sum(TIMERASTER)). Now plot the total number of active neurons against “BR.” Does it look like the phase plot that was produced by Exercise A in this chapter? Could it be similar, only reversed?

J. *More rigorously examining criticality*: The previous methods for fitting power laws are rapid estimates, but there is a more exact and time-consuming way. As mentioned in the book (page 180), there are several software packages that do automated fitting to data to see if a power law distribution is more likely than another distribution, like a lognormal. The script AutomatedFitting will do this if the data is first put into a TIMERASTER format. This script is only slightly modified from a script named demoempdata that was first introduced with the analysis toolbox by (Marshall, Timme, et al., 2016). This toolbox, NCCToolboxV1, should be loaded into your Matlab path so it can provide the functions needed by AutomatedFitting. Note that this analysis may take significantly longer and, in some cases, does not always converge. However, it converged within a few minutes on my computer when I gave it the output of BranchingModel described in the previous section (see the figure below). Another useful toolbox for fitting power laws, though it does not perform avalanche shape collapse, can be found in (Alstott, Bullmore and Plenz, 2014).



Example output from a tunable branching simulation, like BranchingModel, when fed to the script AutomatedFitting. Although this analysis can take longer to perform, it returns maximum likelihood fits to the distributions as well as probability values so you can assess if power laws are more likely than other distributions. A polynomial fit is also provided for the avalanche shape collapse.

Exercise: Use the AutomatedFitting script to process the “TIMERASTER” output from BranchingModel as you sweep “BR” from 0.5 to 1.5 by increments of 0.1. For what values of “BR” does there appear to be reasonable avalanche shape collapse?

K. The empirically measured branching ratio, σ, is an indicator of proximity to the critical point. In an ideal branching model, when σ=1, the network is exactly critical.

   Exercise: Use the function branchingEstimate (adapted from the methods discovered by (Wilting and Priesemann, 2018)) to estimate the branching ratio in a sample of the data sets (say, 20). Measure the branching ratio also in corresponding shuffled data sets. Plot the distribution of branching ratios obtained from actual data and shuffled data, similar to what is shown in Figure 3.9 of the book. What differences do you notice? Is there any relationship between the shape of the avalanche size distributions and the estimated branching ratios? Is there any relationship between the estimated branching ratios and the quality of the avalanche shape collapse?

Matlab code used for exercises in this chapter, listed in order of use:

BrunelNetworkFunctionTimeConstant
BrunelTimeConstantFinal
DemoScriptChapter3
ASDFToSparse
Dissociated2TIMERASTER
VisualizeRaster
spy2
AvalancheAnalysis
runStats
rebinRaster
branchingEstimate
timeShuffle
JitterRaster
GetCCDFs
ExponentRelation
ShapeCollapse
BranchingModel
AutomatedFitting
TIMERASTER_to_asdf2
NCCToolboxV1 (a toolbox containing many functions needed for AutomatedFitting)