Problem Set 8
EPS 528, Science of Complex Systems

Jonas Katona

November 16, 2022

Problem 1.

(a) *Solution.* Here is my program, `BTWfirst.m`, for this problem:

```
function [ data ] = BTWfirst(L, nmax, plotting, disp)
% Model size is LxL
% nmax is the number of iterations
% plotting = 1 (or true) if plot intermediate results
% plotting = 0 (or false) if do not plot intermediate results
% set disp to the number of plots you want to display
% from intermediate iterations. Samples these at equally-
% spaced intervals, and does NOT include the initial one.
% if disp = 0, then no intermediate steps are saved.

% iterates to save
iter = 1;
if disp > 0
   iterq = [(1 : disp) * nmax / (disp + 1) nmax + 1];
else
   iterq = nmax + 1;
end

% initialize grid
U = randi(4, L) - 1;
% store avalanche count here
avalanche = zeros(L);
avalanchesizes = zeros(1, nmax);
% iterate simulation
if plotting || (disp > 0)
    figure(1)
    imagesc(U)
    colormap(flipud(autumn))
    colorbar
    caxis([0 4])
    title('Iterations: 0')
    xlabel('x-position')
    ylabel('y-position')
end
```

```
for i = 1 : nmax
    ind = randi(L, 1, 2);
    U(ind(1), ind(2)) = U(ind(1), ind(2)) + 1;
    if plotting
        figure(2)
        imagesc(U)
        colormap(flipud(autumn))
        colorbar
        caxis([0 4])
        title(['Iterations: ', num2str(i)])
        xlabel('x-position')
        ylabel('y-position')
    end
    % relax first site if needed because we know where it is
    if U(ind(1), ind(2)) > 3
        if ind(1) ~= L
            U(ind(1) + 1, ind(2)) = U(ind(1) + 1, ind(2)) + 1;
        end
        if ind(1) ~= 1
            U(ind(1) - 1, ind(2)) = U(ind(1) - 1, ind(2)) + 1;
        end
        if ind(2) ~= L
            U(ind(1), ind(2) + 1) = U(ind(1), ind(2) + 1) + 1;
        end
        if ind(2) ~= 1
            U(ind(1), ind(2) - 1) = U(ind(1), ind(2) - 1) + 1;
        end
        U(ind(1), ind(2)) = 0;
        avalanche(ind(1), ind(2)) = 1;
        if plotting
                figure(2)
            imagesc(U)
            colormap(flipud(autumn))
            colorbar
            caxis([0 4])
            title(['Iterations: ', num2str(i)])
            xlabel('x-position')
            ylabel('y-position')
        end
    end
    % find other unstable locations
    [I, J] = find(U > 3);
    len = length(I);
    if len > 0
        randomv = randi(len);
        I = I(randomv);
        J = J(randomv);
    end
    while len > 0
```

```
        if I ~= L
            U(I + 1, J) = U(I + 1, J) + 1;
        end
        if I ~= 1
            U(I - 1, J) = U(I - 1, J) + 1;
        end
        if J ~= L
            U(I, J + 1) = U(I, J + 1) + 1;
        end
        if J ~= 1
            U(I, J - 1) = U(I, J - 1) + 1;
        end
        U(I, J) = 0;
        avalanche(I, J) = 1;
        if plotting
            figure(2)
            imagesc(U)
            colormap(flipud(autumn))
            colorbar
            caxis([0 4])
            title(['Iterations: ', num2str(i)])
            xlabel('x-position')
            ylabel('y-position')
        end
        [I, J] = find(U > 3);
        len = length(I);
        if len > 0
            randomv = randi(len);
            I = I(randomv);
            J = J(randomv);
        end
    end
    if (i >= iterq(iter)) && (disp > 0)
        figure(3 + iter - 1)
        imagesc(U)
        colormap(flipud(autumn))
        colorbar
        caxis([0 4])
        title(['Iterations: ', num2str(i)])
        xlabel('x-position')
        ylabel('y-position')
        iter = iter + 1;
    end
    avalanchesizes(i) = sum(sum(avalanche));
    avalanche = zeros(L);
end
if plotting || (disp > 0)
    figure(3 + disp)
    imagesc(U)
```

```
    colormap(flipud(autumn))
    colorbar
    caxis([0 4])
    title(['Iterations: ', num2str(i)])
    xlabel('x-position')
    ylabel('y-position')
end
[count, domain] = histcounts(avalanchesizes,...
    'BinMethod', 'integers');
domain = domain(1 : end - 1) + 0.5;
data = struct('sizes', avalanchesizes,...
'occurrence', [count; domain]);
```

This code is somewhat long and demands an explanation. The inputs are as commented out. At first, we set up a vector of iterations, `iterq`, which gives the iterates on which we will plot the model itself as an $L \times L$ grid. The number of iterates we save is given by the input parameter $0 \leq$`disp`$<$`nmax`, which indicates the number of *intermediate* iterates which are plotted. In particular, note that when `disp`=0, no intermediate iterates are plotted, and the simulation will only plot the initial random sandpile configuration and the one at the end of the simulation. However, if we let `plotting`=1, then not only will *every* iterate in the simulation will be plotted, but also, the avalanches within each iteration will be plotted, except this will all be done *in the same plot*. Hence, turning on `plotting` only helps if we want to visualize the simulation as it runs at each iteration; `disp` allows us to actually display the results in separate figures which can later be saved.

After setting this, we initialize our grid by uniformly and independently setting elements in our $L \times L$ grid, `U`, to 0, 1, 2, or 3. We preallocate arrays `avalanche` and `avalanchesizes`. `avalanche` changes upon each iteration, and marks the corresponding elements in `U` at which an avalanche occurs, i.e., sites which have relaxed on that iteration. If a given site relaxes, then that site will be set to 1 in `avalanche`, while all other sites will be marked with a 0. Then, at the end of each iteration `i`, we sum over all elements in `avalanche` to store in `avalanchesizes(i)`, and then reset `avalanche` to all zeros again.

Before we start the loop, we plot the initial configuration at iteration 0, but only if either `plotting` or `disp` has been turned on. Then, we start the for loop. At the start of each iteration, we choose a random element to add 1 to. After this, we plot the new configuration if `plotting` is on; in fact, for sake of brevity, I will just say now that we plot the grid every single time something changes. From here, if the random element we added to now has a value over 3, we give grains to all of the neighboring sites, and of course, if it has no neighboring site in one direction, then the grain going in that direction will just "disappear" or "fall off" because of the open boundaries. Then, if there are other unstable locations in the grid, we choose one at random to relax like we did above. We continue looping through this process until there are no more unstable locations left.

Finally, once the for loop has completed, if `plotting` or `disp` is on, we plot the final configuration, and save the time sequence of avalanche size and the total number of avalanche occurrence per each size in `data`. In particular, `domain` gives the avalanche size, while `count` gives the total number of avalanche occurrences at that size. Aside

from any figures which may have been plotted as the code ran, `data` is the only output for `BTWfirst.m`.

I have plotted some sample results in Figure 1. Note that, as the number of iterations increases, the grid appears to get darker and darker, indicating how, as we might expect, the "begin" to get taller and taller, which increases the frequency of avalanches. In other words, the sand begins to "accumulate" across the domain as the iterations continue, but of course, not completely, because the number of avalanches is bound to increase as well. □

(b) *Solution.* To run the desired results for this part, I wrote the following script, `BTWsecond.m`, which relies off of `BTWfirst.m` to run

```
tic
Lq = 2 .^ (3 : 6);
nmax = 1e5;

data1 = BTWfirst(Lq(1), nmax, 0, 0);
stuff = data1.occurrence;
P1 = stuff(1, :) / nmax;
domain1 = stuff(2, :);
Pshorter = P1(2 : ceil(end / 5));
domainshorter = domain1(2 : ceil(end / 5));
Pnonzero = Pshorter(intersect(...
    intersect(find(domainshorter < 21),...
    find(Pshorter)), find(domainshorter)));
domainnonzero = domainshorter(intersect(intersect(find(...
    domainshorter < 21), find(...
    Pshorter)), find(domainshorter)));
m1 = sum((log(Pnonzero) - mean(log(Pnonzero)))...
    .* (log(domainnonzero) - mean(log(domainnonzero))))...
    / sum((log(domainnonzero) -...
    mean(log(domainnonzero))) .^ 2);

data2 = BTWfirst(Lq(2), nmax, 0, 0);
stuff = data2.occurrence;
P2 = stuff(1, :) / nmax;
domain2 = stuff(2, :);
Pshorter = P2(2 : ceil(end / 5));
domainshorter = domain2(2 : ceil(end / 5));
Pnonzero = Pshorter(intersect(...
    intersect(find(domainshorter < 21),...
    find(Pshorter)), find(domainshorter)));
domainnonzero = domainshorter(intersect(intersect(find(...
    domainshorter < 21), find(...
    Pshorter)), find(domainshorter)));
m2 = sum((log(Pnonzero) - mean(log(Pnonzero)))...
    .* (log(domainnonzero) - mean(log(domainnonzero))))...
    / sum((log(domainnonzero) -...
    mean(log(domainnonzero))) .^ 2);
```

```
data3 = BTWfirst(Lq(3), nmax, 0, 0);
stuff = data3.occurrence;
P3 = stuff(1, :) / nmax;
domain3 = stuff(2, :);
Pshorter = P3(2 : ceil(end / 5));
domainshorter = domain3(2 : ceil(end / 5));
Pnonzero = Pshorter(intersect(...
    intersect(find(domainshorter < 21),...
    find(Pshorter)), find(domainshorter)));
domainnonzero = domainshorter(intersect(intersect(find(...
    domainshorter < 21), find(...
    Pshorter)), find(domainshorter)));
m3 = sum((log(Pnonzero) - mean(log(Pnonzero)))...
    .* (log(domainnonzero) - mean(log(domainnonzero))))...
    / sum((log(domainnonzero) -...
    mean(log(domainnonzero))) .^ 2);

data4 = BTWfirst(Lq(4), nmax, 0, 0);
stuff = data4.occurrence;
P4 = stuff(1, :) / nmax;
domain4 = stuff(2, :);
Pshorter = P4(2 : ceil(end / 5));
domainshorter = domain4(2 : ceil(end / 5));
Pnonzero = Pshorter(intersect(...
    intersect(find(domainshorter < 21),...
    find(Pshorter)), find(domainshorter)));
domainnonzero = domainshorter(intersect(intersect(find(...
    domainshorter < 21), find(...
    Pshorter)), find(domainshorter)));
m4 = sum((log(Pnonzero) - mean(log(Pnonzero)))...
    .* (log(domainnonzero) - mean(log(domainnonzero))))...
    / sum((log(domainnonzero) -...
    mean(log(domainnonzero))) .^ 2);

figure(1)
loglog(domain1, P1, domain2, P2, domain3, P3, domain4, P4)
title('Avalanche probability vs. size', 'interpreter',...
    'latex', 'fontsize', 18)
xlabel('Avalanche size ($s$)', 'interpreter',...
    'latex', 'fontsize', 14)
ylabel('Avalanche probability ($P(s; L)$)',...
    'interpreter', 'latex', 'fontsize', 14)
legend(['L=8 (slope: ', num2str(m1), ')'],...
    ['L=16 (slope: ', num2str(m2), ')'],...
    ['L=32 (slope: ', num2str(m3), ')'],...
    ['L=64 (slope: ', num2str(m4), ')'], 'interpreter', 'latex')
toc
```

Note that to get higher-quality results, we run our model for $L = 8$, 16, 32, and 64

still, but with $n_{\max} = 10^5$ instead. In the code, we run through each of these cases for L and extract both `domaini`, the vector of avalanche sizes recorded for case `i`, of which there are four such cases, as well as `Pi`, which is $P(s; L)$ for case `i`. Furthermore, we also extract the slope of the log-log plot of $P(s; L)$ vs. s for each case, but only for avalanche sizes $s = 1, \ldots, 20$,[1] and also excluding any cases where $P(s; L)$ might be zero, which usually only occurs because the probability is so small that we were not able to capture it, even when we iterated the simulation 10^5 times. The reason why we exclude results for avalanche sizes which are too large is because, due to the effects of a finite domain (mean-field theory assumes an infinite domain), the simulation results exhibit an increasing deviation from the expected power law results. Note that the only reason why we want to find the slope of the log-log plot is because we want to check the results mentioned in the notes: This slope should be roughly -1 for the *conserved* BTW model, indicating that $P(s; L) \approx Cs^{-1}$ for some constant $C \in \mathbb{R}$ and only when $s \ll L^2$.

Two sample results produced by running `BTWsecond.m` can be found in Figure 2. The first simulation took 40.691451 seconds to run and the second one 46.145808 seconds, but in other cases, it took anywhere from around 30 seconds to just under a minute. Note that for all cases of L, at least for smaller values of s (until the finite domain deviations kick in), the log-log plots exhibited linear trends that were all very close to -1, confirming the power law behavior we were told should occur for the original BTW model in the notes. Here are two other relevant observations for our results:

(a) Regardless of L, the probability functions $P(s; L)$ appear roughly the same, but as L increased, these go down ever-so-slightly. This can be best explained by how, with a larger domain size, it is possible to have larger avalanches, albeit, these will be increasingly rare with size. However, since we are referring to a probability here, for a fixed value of n_{iter}, there will have to be relatively less smaller avalanches if we are allowing for larger avalanches as well. This brings the whole curve down, since as with any appropriately-normalized probability distribution, the area under the curve should be 1.

(b) The larger the value of L, the later the deviation from the power law occurs. This can be best explained because of the finite-domain effect: naturally, as L increases, greater avalanches will be better permitted, and in the absence of finite-domain restrictions, the probability of these occurring should follow a power law with an exponent of -1.

I would in fact predict that, if we chose some arbitrarily massive values for n_{iter} and L, then we would indeed exhibit a power law behavior up to arbitrarily large values of s, since increases the first two parameters would put the finite-domain effect at larger and larger values of s on our plot. □

Problem 2.

Solution. I wrote the following code, `BTWnonconsfirst.m`, to implement the non-conservative/random-neighbor BTW model in 2-D, which is largely based off of `BTWfirst.m` from Problem 1(a):

```
function [ data ] = BTWnonconsfirst(L, nmax, plotting, disp, p)
% Model size is LxL
```

[1] Using s only up to $s = 20$ was done for no particular reasons, other than we needed to choose an $s \ll L^2$. I encourage you to rerun my code, except with 21 replaced by some other number.

```
% nmax is the number of iterations
% plotting = 1 (or true) if plot intermediate results
% plotting = 0 (or false) if do not plot intermediate results
% set disp to the number of plots you want to display
% from intermediate iterations. Samples these at equally-
% spaced intervals, and does NOT include the initial one.
% if disp = 0, then no intermediate steps are saved.
% p is the probability that a grain is successfully distributed

% iterates to save
iter = 1;
if disp > 0
   iterq = [(1 : disp) * nmax / (disp + 1) nmax + 1];
else
   iterq = nmax + 1;
end

% initialize grid
U = randi(4, L) - 1;
% store avalanche count here
avalanche = zeros(L);
avalanchesizes = zeros(1, nmax);
% iterate simulation
if plotting || (disp > 0)
    figure(1)
    imagesc(U)
    colormap(flipud(autumn))
    colorbar
    caxis([0 4])
    title('Iterations: 0')
    xlabel('x-position')
    ylabel('y-position')
end
for i = 1 : nmax
    ind = randi(L, 1, 2);
    U(ind(1), ind(2)) = U(ind(1), ind(2)) + 1;
    if plotting
        figure(2)
        imagesc(U)
        colormap(flipud(autumn))
        colorbar
        caxis([0 4])
        title(['Iterations: ', num2str(i)])
        xlabel('x-position')
        ylabel('y-position')
    end
    if (i >= iterq(iter)) && (disp > 0)
            figure(3 + iter - 1)
            imagesc(U)
```

```
        colormap(flipud(autumn))
        colorbar
        caxis([0 4])
        title(['Iterations: ', num2str(i)])
        xlabel('x-position')
        ylabel('y-position')
        iter = iter + 1;
end
% relax first site if needed because we know where it is
if U(ind(1), ind(2)) > 3
    more = rand < mod(4 * p, 1);
    sites = randperm(4, min(floor(4 * p) + more, 4));
    if (ind(1) ~= L) && (ismember(1, sites))
        U(ind(1) + 1, ind(2)) = U(ind(1) + 1, ind(2)) + 1;
    end
    if (ind(1) ~= 1) && (ismember(2, sites))
        U(ind(1) - 1, ind(2)) = U(ind(1) - 1, ind(2)) + 1;
    end
    if (ind(2) ~= L) && (ismember(3, sites))
        U(ind(1), ind(2) + 1) = U(ind(1), ind(2) + 1) + 1;
    end
    if (ind(2) ~= 1) && (ismember(4, sites))
        U(ind(1), ind(2) - 1) = U(ind(1), ind(2) - 1) + 1;
    end
    U(ind(1), ind(2)) = 0;
    avalanche(ind(1), ind(2)) = 1;
    if plotting
            figure(2)
        imagesc(U)
        colormap(flipud(autumn))
        colorbar
        caxis([0 4])
        title(['Iterations: ', num2str(i)])
        xlabel('x-position')
        ylabel('y-position')
    end
end
% find other unstable locations
[I, J] = find(U > 3);
len = length(I);
if len > 0
    randomv = randi(len);
    I = I(randomv);
    J = J(randomv);
end
while len > 0
    more = rand < mod(4 * p, 1);
    sites = randperm(4, min(floor(4 * p) + more, 4));
    if (I ~= L) && (ismember(1, sites))
```

```
            U(I + 1, J) = U(I + 1, J) + 1;
        end
        if (I ~= 1) && (ismember(2, sites))
            U(I - 1, J) = U(I - 1, J) + 1;
        end
        if (J ~= L) && (ismember(3, sites))
            U(I, J + 1) = U(I, J + 1) + 1;
        end
        if (J ~= 1) && (ismember(4, sites))
            U(I, J - 1) = U(I, J - 1) + 1;
        end
        U(I, J) = 0;
        avalanche(I, J) = 1;
        if plotting
            figure(2)
            imagesc(U)
            colormap(flipud(autumn))
            colorbar
            caxis([0 4])
            title(['Iterations: ', num2str(i)])
            xlabel('x-position')
            ylabel('y-position')
        end
        [I, J] = find(U > 3);
        len = length(I);
        if len > 0
            randomv = randi(len);
            I = I(randomv);
            J = J(randomv);
        end
    end
    if (i >= iterq(iter)) && (disp > 0)
        figure(3 + iter - 1)
        imagesc(U)
        colormap(flipud(autumn))
        colorbar
        caxis([0 4])
        title(['Iterations: ', num2str(i)])
        xlabel('x-position')
        ylabel('y-position')
        iter = iter + 1;
    end
    avalanchesizes(i) = sum(sum(avalanche));
    avalanche = zeros(L);
end
if plotting || (disp > 0)
    figure(3 + disp)
    imagesc(U)
    colormap(flipud(autumn))
```

```
    colorbar
    caxis([0 4])
    title(['Iterations: ', num2str(i)])
    xlabel('x-position')
    ylabel('y-position')
end
[count, domain] = histcounts(avalanchesizes,...
    'BinMethod', 'integers');
domain = domain(1 : end - 1) + 0.5;
data = struct('sizes', avalanchesizes,...
    'occurrence', [count; domain]);
```

The only differences between `BTWnonconsfirst.m` and `BTWfirst.m` is that whenever a site relaxes, it only distributes grains to a fraction of its neighbors, and loses the rest of these that would have been distributed in the conservative model. This loss is determined by some probability parameter $p \in [0, 1]$, such that the expected number of grains distributed at each iteration will be $4p$, while the amount lost from the system entirely will be $4 - 4p = 4(1 - p)$. Clearly, if p is a multiple of $1/4$, then we can just have the system distribute $4p$ grains at each iteration, as the problem states we should do. If p is not a multiple of $1/4$, then we could have been lazy and let the code round p to the nearest multiple of $1/4$ or just stopped the code and forced one to put in p which is a multiple of $1/4$. However, instead, in this case we will draw from a uniform distribution on $[0, 1]$. If the number we draw is below $4p \bmod 1$, then we distribute $4\lceil p \rceil$ grains. Else, we only add $4\lfloor p \rfloor$. Hence, we can sort of weight the probability that we distribute $4\lceil p \rceil$ or $4\lfloor p \rfloor$ grains by how close the given p is to some integer multiple of $1/4$.

A sample result for $L = 100$, $n_{\max} = 10^4$, and $p = 0.5$ is given in Figure 3, where we have saved two intermediate results. Note that in this case, unlike with the conservative model, the amount of shading in the plot does not appear to increase as the simulation progresses. This is because, due to the dissipation, we get a much lower accumulation of "sandpiles" which are close to being unstable, i.e., with some value such as 2 or 3.

Furthermore, note that in class, Professor Korenaga only derives $P(s, p)$ for the 1D random-neighbor sandpile model; the one we have here is in 2D. To derive a new theoretical prediction for $P(s, p)$, the probability of having an avalanche of size s with a relaxation probability of p, we now have to consider the number of *quaternary* trees of size s, since each relaxed site can distribute its grains to *four* neighbors, rather than *two* neighbors as we find in the 1D case. Without proof, I turn you to two following sources: Example 5 in Section 7.8 of Graham, Knuth, and Patashnik's *Concrete Mathematics*, and an abridged version of this found on Stack Exchange: https://math.stackexchange.com/questions/1596453/number-of-ternary-trees-finding-a-recurrent-relationship. From here, the number of m-trees of size s is $\frac{1}{(m-1)s+1}\binom{ms}{s}$. When we are dealing with binary trees, $m = 2$, such that we have $\frac{1}{(2-1)s+1}\binom{2s}{s} = \frac{1}{s+1}\binom{2s}{s}$, the same result derived in class and in the notes. Hence, for quaternary trees, $m = 4$, which gives us that there are $\frac{1}{(4-1)s+1}\binom{4s}{s} = \frac{1}{3s+1}\binom{4s}{s}$ quaternary trees of size s. Finally, we also need to consider the probability that an avalanche of that size occurs, as represented with such a quaternary tree. Let $\tilde{p}$ be the probability that a node induces a relaxation, which we will relate to p in the problem later. Then, if we inductively count the number of nodes where grains have spread to (with double,

triple, etc. counting if necessary), it is not hard to see that

$$\boxed{P(s,p) = \frac{1}{3s+1}\binom{4s}{s}\tilde{p}^{s-1}(1-\tilde{p})^{3s+1}.} \tag{1}$$

We can further approximate (1) using Stirling's approximation, $n! \approx \sqrt{2\pi n}n^n \exp(-n)$, and how $\frac{1}{3s+1} \approx \frac{1}{3s}$ for $n, s \gg 1$ to derive a more informative form of the size-frequency distribution.

$$\begin{aligned}
P(s,p) &= \frac{1}{3s+1}\binom{4s}{s}\tilde{p}^{s-1}(1-\tilde{p})^{3s+1} = \frac{1}{3s+1}\frac{(4s)!}{(3s)!s!}\tilde{p}^{s-1}(1-\tilde{p})^{3s+1} \\
&\approx \frac{1}{3s}\frac{\sqrt{8\pi s}(4s)^{4s}\exp(-4s)}{\sqrt{6\pi s}(3s)^{3s}\exp(-3s)\sqrt{2\pi s}s^s\exp(-s)}\tilde{p}^{s-1}(1-\tilde{p})^{3s+1} \\
&= \frac{1}{3s}\frac{2^{1/2+8s}3^{-1/2-3s}}{\sqrt{\pi s}}\tilde{p}^{s-1}(1-\tilde{p})^{3s+1} = \frac{2^{1/2+8s}3^{-3/2-3s}}{\sqrt{\pi}}s^{-3/2}\tilde{p}^{s-1}(1-\tilde{p})^{3s+1} \\
&= \frac{1}{3s}\frac{2^{1/2}2^{8s}3^{-1/2}3^{-3s}}{\sqrt{\pi s}}\tilde{p}^{s-1}(1-\tilde{p})^{3s+1} = \frac{2^{1/2}2^{8s}3^{-3/2}3^{-3s}}{\sqrt{\pi}}s^{-3/2}\frac{1-p}{p}\tilde{p}^{s}(1-\tilde{p})^{3s} \\
&= \frac{\sqrt{6}}{9\sqrt{\pi}}s^{-3/2}\frac{1-p}{p}\left[\frac{256}{27}\tilde{p}(1-\tilde{p})^3\right]^s
\end{aligned}$$

$$\Rightarrow \boxed{P(s,p) \approx \frac{\sqrt{6}}{9\sqrt{\pi}}s^{-3/2}\frac{1-p}{p}\left[\frac{256}{27}\tilde{p}(1-\tilde{p})^3\right]^s \Rightarrow P(s,p) \sim s^{-3/2}\exp\left(-\frac{s}{s_c(p)}\right),} \tag{2}$$

where in this case, $\boxed{s_c(p) = -\frac{1}{\log\left[\frac{256}{27}\tilde{p}(1-\tilde{p})^3\right]}}$. Finally, in relation to the problem, since we can distribute up to 4 sand grains at each relaxation, $\boxed{\tilde{p} = \frac{p}{4}}$, i.e., effectively, p in the problem is equivalent to α_n in the notes, where $\alpha_n z_c = 4$ in this case.

We can use (2) to see what we should expect from our simulations before even running any simulations. Namely, we want to check that what we derived here, (1)-(2), actually reproduces similar results to what we saw in lecture on the 1D random-neighbor sandpile model. First off, note that $p \to 1 \Rightarrow \widetilde{p} \to \frac{1}{4} \Rightarrow \frac{256}{27}\frac{1}{4}\left(1-\frac{1}{4}\right)^3 = 1 \Rightarrow s_c(p=1) = \infty \Rightarrow P(s, p=1) \sim s^{-3/2}$, which showcases a power law with the *same* exponent as the one we got via mean-field approximation in lecture. Similarly, (2) appears to be a function of the same form as the one derived in lecture, except with a *different* function $s_c(p)$ that appears to vary more sharply with p, indicating the increase in the degrees of freedom in the system as we go from 1D to 2D, and hence, the number of possible directions for avalanches to occur. Finally, $P(s,p)$ exhibits the same trend as the one in lecture, where for $\widetilde{p} < \frac{1}{4}$, the log-log plot of $P(s,p)$ has a concave down, decreasing shape, while at the critical point $\widetilde{p} = \frac{1}{4}$, the log-log plot is linear.

Now, the question is, do we see these same trends exhibited in the mean-field approximations (1)-(2) from our numerical simulations? We must put these predictions to the test using the code `BTWnonconssecond.m`, which is a modified version of `BTWsecond.m` which was used in Problem 1(b):

```
L = 128;
nmax = 1e4;
pq = 0.25 : 0.25 : 1;
```

```
data1 = BTWnonconsfirst(L, nmax, 0, 0, pq(1));
stuff = data1.occurrence;
P1 = stuff(1, :) / nmax;
domain1 = stuff(2, :);
disp('Completed p=0.25.')

data2 = BTWnonconsfirst(L, nmax, 0, 0, pq(2));
stuff = data2.occurrence;
P2 = stuff(1, :) / nmax;
domain2 = stuff(2, :);
disp('Completed p=0.5.')

data3 = BTWnonconsfirst(L, nmax, 0, 0, pq(3));
stuff = data3.occurrence;
P3 = stuff(1, :) / nmax;
domain3 = stuff(2, :);
disp('Completed p=0.75.')

data4 = BTWnonconsfirst(L, nmax, 0, 0, pq(4));
stuff = data4.occurrence;
P4 = stuff(1, :) / nmax;
domain4 = stuff(2, :);
disp('Completed p=1.')

% we already know that it will roughly lie in this range
Pshorter = P4(2 : ceil(end / 2));
domainshorter = domain4(2 : ceil(end / 2));
% only do lsq when avalanche size is under 30
Pnonzero = Pshorter(intersect(...
    intersect(find(domainshorter < 30),...
    find(Pshorter)), find(domainshorter)));
domainnonzero = domainshorter(intersect(intersect(find(...
    domainshorter < 30), find(Pshorter)), find(domainshorter)));

P = @(s, p) (factorial(4 * s) ./ (factorial(s)...
    .* factorial(3 * s))) .* (p .^ (s - 1)) .*...
    ((1 - p) .^ (3 * s + 1)) ./ (3 * s + 1);
P1theory = (dot(P1(2 : end), P(domain1(2 : end), pq(1) / 4))...
    / dot(P(domain1(2 : end), pq(1) / 4), P(domain1(2 : end),...
    pq(1) / 4))) * P(domain1, pq(1) / 4);
P2theory = (dot(P2(2 : end), P(domain2(2 : end), pq(2) / 4))...
    / dot(P(domain2(2 : end), pq(2) / 4), P(domain2(2 : end),...
    pq(2) / 4))) * P(domain2, pq(2) / 4);
P3theory = (dot(P3(2 : end), P(domain3(2 : end), pq(3) / 4))...
    / dot(P(domain3(2 : end), pq(3) / 4), P(domain3(2 : end),...
    pq(3) / 4))) * P(domain3, pq(3) / 4);
P4theory = (dot(Pnonzero, P(domainnonzero, pq(4) / 4))...
    / dot(P(domainnonzero, pq(4) / 4), P(domainnonzero,...
    pq(4) / 4))) * P(domain4, pq(4) / 4);
```

```
figure(1)
loglog(domain1, [P1; P1theory])
title('Avalanche probability vs. size ($p=0.25$)',...
    'interpreter', 'latex', 'fontsize', 18)
xlabel('Avalanche size ($s$)', 'interpreter',...
    'latex', 'fontsize', 14)
ylabel('Avalanche probability ($P(s; p)$)',...
    'interpreter', 'latex', 'fontsize', 14)
legend('Numerical', 'Theory', 'interpreter', 'latex')

figure(2)
loglog(domain2, [P2; P2theory])
title('Avalanche probability vs. size ($p=0.5$)',...
    'interpreter', 'latex', 'fontsize', 18)
xlabel('Avalanche size ($s$)', 'interpreter',...
    'latex', 'fontsize', 14)
ylabel('Avalanche probability ($P(s; p)$)',...
    'interpreter', 'latex', 'fontsize', 14)
legend('Numerical', 'Theory', 'interpreter', 'latex')

figure(3)
loglog(domain3, [P3; P3theory])
title('Avalanche probability vs. size ($p=0.75$)',...
    'interpreter', 'latex', 'fontsize', 18)
xlabel('Avalanche size ($s$)', 'interpreter',...
    'latex', 'fontsize', 14)
ylabel('Avalanche probability ($P(s; p)$)',...
    'interpreter', 'latex', 'fontsize', 14)
legend('Numerical', 'Theory', 'interpreter', 'latex')

figure(4)
loglog(domain4, [P4; P4theory])
title('Avalanche probability vs. size ($p=1$)',...
    'interpreter', 'latex', 'fontsize', 18)
xlabel('Avalanche size ($s$)', 'interpreter',...
    'latex', 'fontsize', 14)
ylabel('Avalanche probability ($P(s; p)$)',...
    'interpreter', 'latex', 'fontsize', 14)
legend('Numerical', 'Theory', 'interpreter', 'latex')

figure(5)
loglog(domain1, P1, domain2, P2,...
    domain3, P3, domain4, P4)
title('Avalanche probability vs. size (all cases)',...
    'interpreter', 'latex', 'fontsize', 18)
xlabel('Avalanche size ($s$)', 'interpreter',...
    'latex', 'fontsize', 14)
ylabel('Avalanche probability ($P(s; p)$)',...
```

```
    'interpreter', 'latex', 'fontsize', 14)
legend('p=0.25', 'p=0.5', 'p=0.75',...
    'p=1', 'interpreter', 'latex')
```

The code above runs the cases $p = 0.25$, $p = 0.5$, $p = 0.75$, and $p = 1$ with $L = 128$ (increased from $L = 64$ in the problem so that we have less finite-domain effects) for the dissipative model we introduced via the script `BTWnonconsfirst.m`, compares these with the theoretical model (1),[2] and then plots the results in five plots. The first four plots compare each of the cases separately with their associated mean-field models given by (1), while the fifth one plots all of the simulation results together on the same plot but *without* the mean-field models.

One result of these simulations is given in Figures 4 and 5. All cases reproduce the trends described in lecture, as well as from our derived equations (1)-(2) pretty well, in the sense that we had a falling curve for the $p < 1$ cases and a straight line when $p = 1$ in these log-log plots, which corresponds to the conservative BTW model. The main place where the numerical results diverged from the mean-field model was when s got too large, indicating the influence of the finite grid size as we saw in the conservative model as well.

However, the $p = 1$ case appeared to diverge much earlier. This is actually due to the failures of our mean-field model: As discussed in Problem 1, we know empirically that the original, conservative BTW model does give us a power law relationship, and hence a straight line on a log-log plot, for $P(s, p)$ vs. s. However, this power law has exponent -1, but our mean-field theory gives an exponent $-3/2$. Thus, the reason why our simulation does not match that well with the mean-field results is because we are effectively trying to match a line with slope $-3/2$ to a curve which has a slope of -1, or at least for values of s which are not too large. In Prof. Korenaga's words, "This is how far we can go with a mean-field approximation." □

[2] (2) was mainly useful for qualitative arguments, but unfortunately, the approximation breaks down for small values of s because the simplifying assumptions we made are only valid for $s \gg 1$.

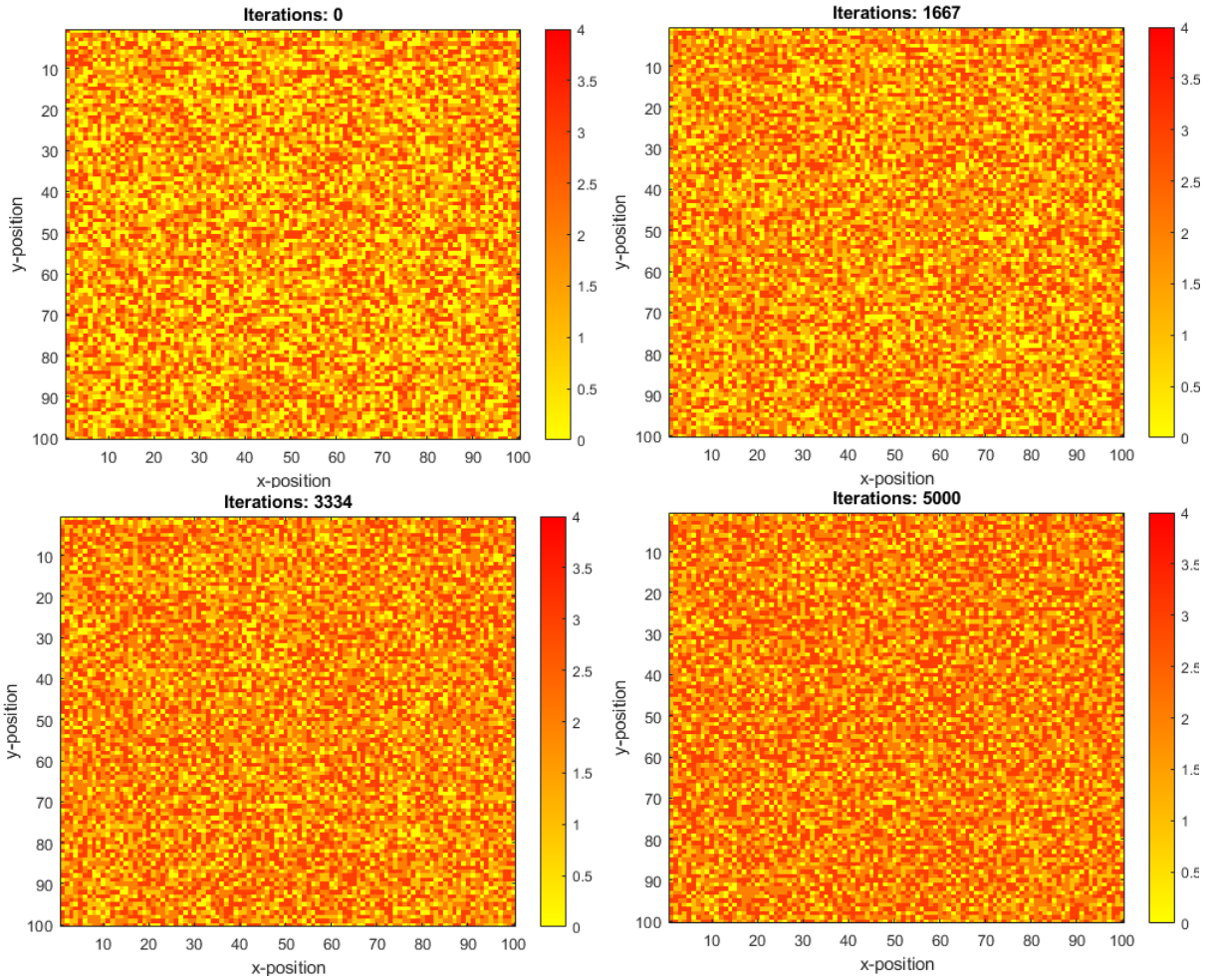
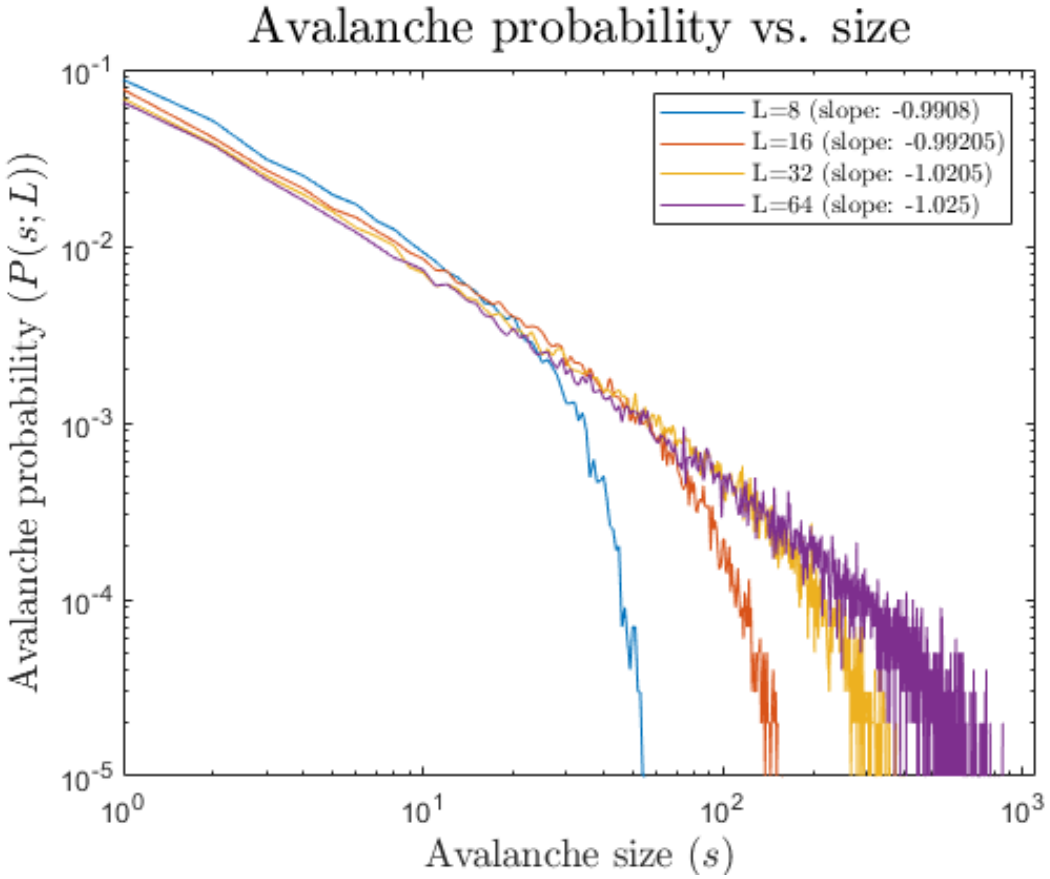


Figure 1: The above were the figures produced when we ran our code from Problem 1(a) with L=100, nmax=5e3, plotting=0, and disp=2, which ends up returning disp+2=4 plots.



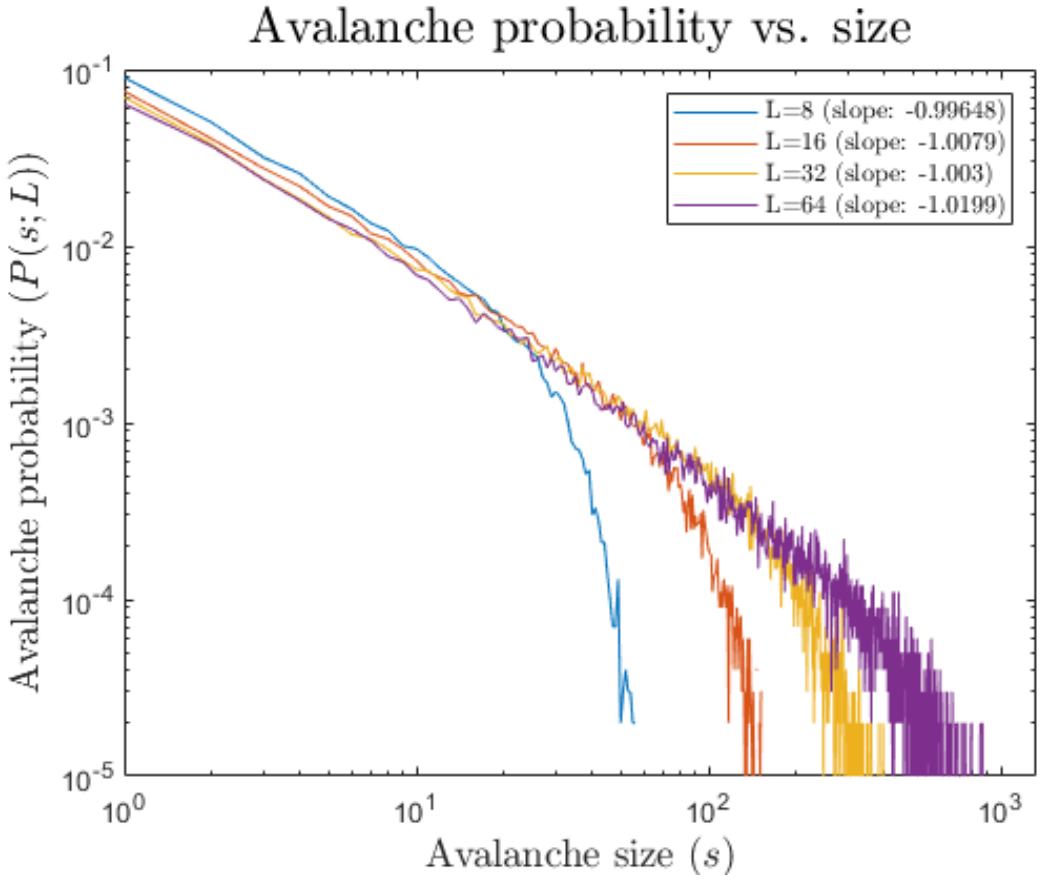


Figure 2: Two sample results for Problem 1(b).

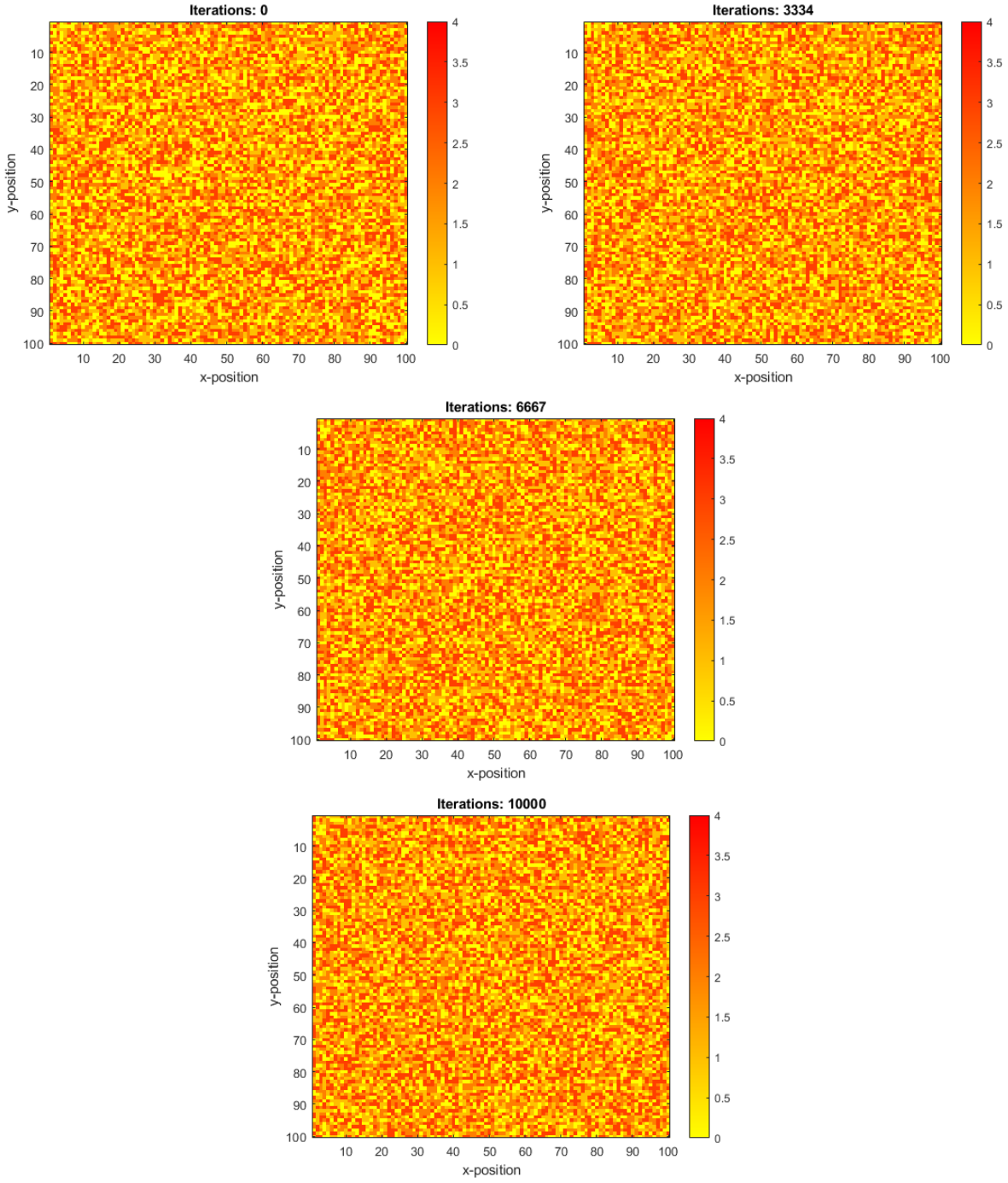


Figure 3: The above were the figures produced when we ran our code from the first part of Problem 2 (the dissipative BTW model) with L=100, nmax=1e4, plotting=0, disp=2, and $p = 0.5$.

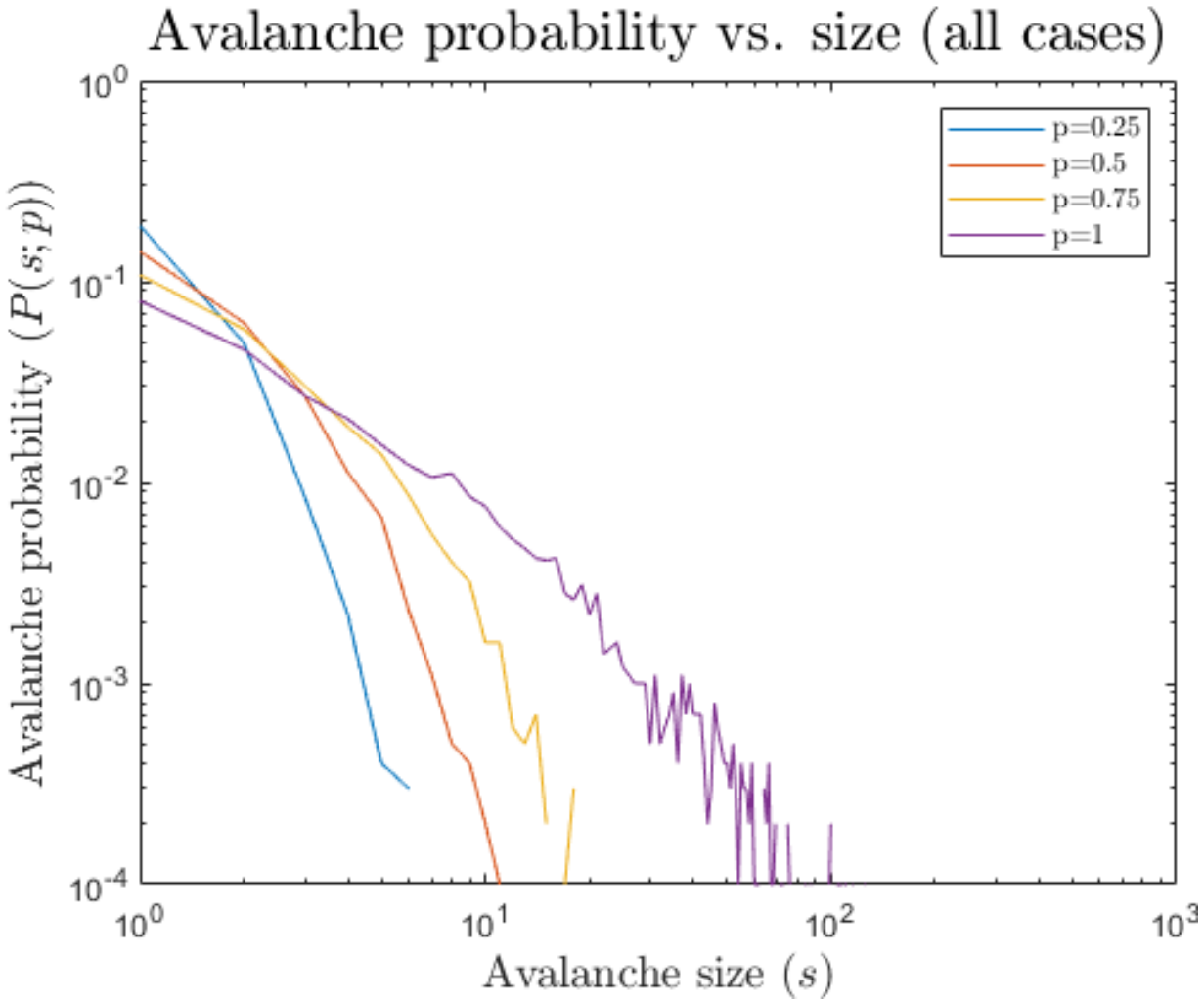


Figure 4: One sample simulation for the results of the dissipative BTW model, ran for $p = 0.25$, $p = 0.5$, $p = 0.75$, and $p = 1$, with $L = 128$ in all cases.

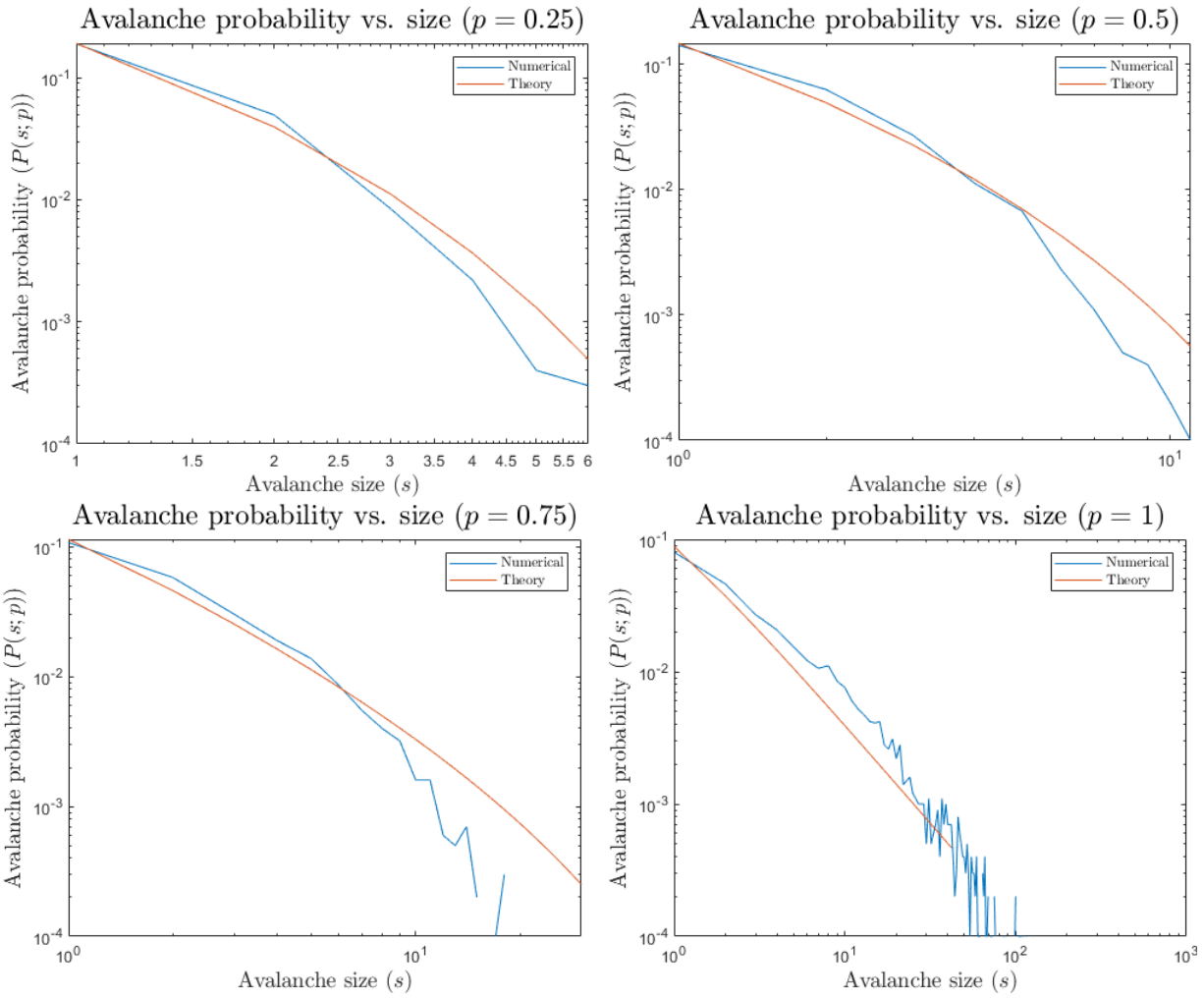


Figure 5: A comparison between the results from mean-field theory and the numerical simulations for the dissipative BTW model, ran with $L = 128$ for $p = 0.25$, $p = 0.5$, $p = 0.75$, and $p = 1$. The curves generally show decent agreement, with the least agreement in the $p = 1$ case, as explained in the text.