Chapter 5

Sandpiles

{chap:sandpile

The sky is blue, the sun is high, and you are sitting idle on a beach, a cold beer in one hand and a handful of dry sand in the other. Sand is slowly trickling through your fingers, and as a consequence a small conical pile of sand is slowly growing below your hand. Sand avalanches of various sizes intermittently slide down the slope of the pile, which is growing both in width and in height *but maintains the same slope angle*.

However mundane this minor summer vacation event might appear, it has become the icon of *Self-Organized Criticality* (hereafter SOC), an extremely robust mechanism for the autonomous development of complex, scale-invariant behaviors and patterns in natural systems. SOC will be encountered again and again in subsequent chapters, hiding under a variety of disguises, but here we shall first restrict ourselves to an extremely simple computational idealization of that iconic summertime pile of sand.

5.1 Model definition

The sandpile model is a lattice-based cellular automaton-like system evolving according to simple, discrete rules, local in space and time. Here we consider a one-dimensional lattice made up of N nodes with right+left neighbour connectivity, as in 1D percolation (see Fig. 4.1). This lattice is used to discretize a real-valued variable S_j^n, where the subscript j identifies a node on the lattice and the superscript n denotes a temporal iteration. Initially ($n = 0$) we set

$$S_j^0 = 0 \ , \qquad j = 0, ..., N-1 \ . \tag{5.1}$$ {eq:sandp1}

This nodal variable is subjected to a forcing mechanism, whereby at each temporal iteration a small increment s is added to the variable S, at a single randomly selected node:

$$S_r^{n+1} = S_r^n + s \ , \qquad r \in [0, N-1] \ , \qquad s \in [0, \varepsilon] \ , \tag{5.2}$$ {eq:sandp2}

where r and s are extracted from a uniform distribution of random deviates spanning the given ranges, and the maximum increment ε is an input parameter of the model. The physical system inspiring this simple model is a pile of sand, so you may imagine that S_j^n measures the height of the sandpile at the position j on the lattice at time n, and the forcing mechanism amounts to dropping sand grains at random locations on the pile. Obviously, the sandpile will grow in height in response to this forcing... at least at first.

Now for the dynamics of the system; as the pile grows, at each temporal iteration the magnitude of the *slope* associated with each nodal pair $(j, j+1)$ is calculated:

$$z_j^n = |S_{j+1}^n - S_j^n| \ , \qquad j = 0, ..., N-2 \ . \tag{5.3}$$ {eq:sandp4a}

If this slope exceeds a preset critical threshold Z_c, then the nodal pair $(j, j+1)$ is deemed unstable. This embodies the idea of static friction between sand grains in contact, which can

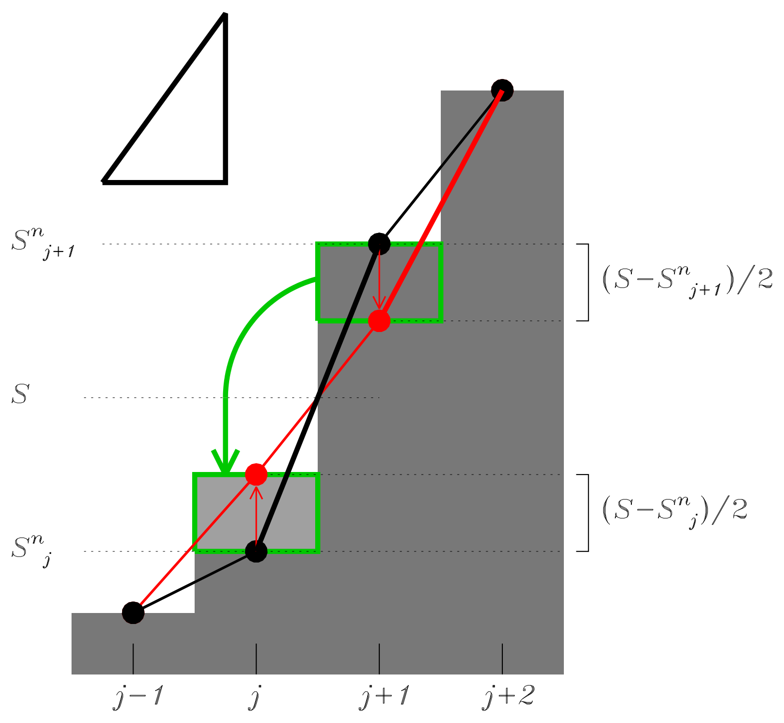

Figure 5.1: Action of the redistribution rules given by eqs. (5.4). The dark gray columns indicate the nodal values (sand height) for a quartet of contiguous nodes, with the black solid dots linked by solid lines indicating the slope, as given by eq. (5.3) and with thicker line segments flagging slopes in excess of the threshold Z_c (depicted by the triangular wedge at top left). Here the nodal pair $(j, j+1)$ exceeds this critical slope, so that the redistribution alters the nodal values as indicated by the two red vertical arrows. This is equivalent to moving by one nodal spacing downslope the quantity of "sand" enclosed by the upper green box, as indicated by the green arrow. This adjustment leads to the new slopes traced by the red dots and solid lines, which here is now unstable for the nodal pair $(j+1, j+2)$. This would lead to another readjustment at the next iteration (see text). {fig:slopedemo}

equilibrate gravity up to a certain inclination angle, beyond which sand grains start toppling downslope. A redistribution rule capturing this toppling process is applied so as to restore stability at the subsequent iteration. Here we use the following simple rule:

$$S_j^{n+1} = S_j^n + (\bar{S} - S_j^n)/2 \ , \qquad S_{j+1}^{n+1} = S_{j+1}^n + (\bar{S} - S_{j+1}^n)/2 \ , \tag{5.4}$$ {eq:sandp4b}

where

{???} $$\bar{S} = (S_{j+1}^n + S_j^n)/2 \ . \tag{5.5}$$

This rule displaces a quantity of sand from the node with the higher S_j^n value to the other, such that the local slope z_j^n is reduced by a factor of two. Figure 5.1 illustrates this redistribution process. If $\varepsilon \ll S_j, S_{j+1}$, then the critical slope is only exceeded by a small amount, and the above rule will always restore local stability. It is left as an easy exercise in algebra to verify that this rule is *conservative*, in the sense that sand is neither created or destroyed by the redistribution:

{eq:sandp4c} $$S_j^{n+1} + S_{j+1}^{n+1} = S_j^n + S_{j+1}^n \ , \tag{5.6}$$

and that the quantity δS_j^n of sand displaced is given by

{eq:sandp5}
$$\delta S_j^n = \frac{z_j^n}{4} \ , \tag{5.7}$$

as indicated by the green boxes on Fig. 5.1. But now, even if the pair $(j, j+1)$ was the only unstable one on the lattice at iteration n, the redistribution has clearly changed the slope associated with the neighbouring nodal pairs $(j-1, j)$ and $(j+1, j+2)$ since S_j^n and S_{j+1}^n have both changed; and it is certainly possible that one (or both) of these neighbouring pairs now exceeds the critical threshold Z_c as a result. This is the case for the pair $(j+1, j+2)$ in the specific configuration depicted on Fig. 5.1. The redistribution rule is applied anew to that unstable nodal pair; but then the stability of its neighbouring pairs must again be verified, and the redistribution rule applied once again if needed, and so on. This sequential process amounts to an *avalanche* of sand being displaced downslope, until every pair of contiguous nodes on the lattice is again stable with respect to eq. (5.3).

Now the boundary conditions comes into play. At the last node of the lattice, at every iteration n we remove any sand having accumulated there due to an arriving avalanche:

$$S_{N-1}^n = 0 \ . \tag{5.8}$$
{eq:sandp3}

This is as if the sandpile reached to the edge of a table, with sand simply falling off when moving beyond this position. No such removal takes place at the first node, which may be imagined as being due to the presence of a containing wall. The boundary condition (5.8) turns out to play a crucial role here. Because the redistribution rule is conservative, and in view of the inexorable addition of sand to the system mediated by the forcing rule, the boundary is the only place where sand can be evacuated from the system.

In light of all this, one may imagine that a stationary state can be reached, characterized by a global slope equal to Z_c, with avalanches moving sand to the bottom of the pile at the same (average) rate as the forcing rule is loading the pile. As we shall see presently, a stationary state is indeed reached, but presents some characteristics one would have been very hard pressed to anticipate on the basis of the simple rules introduced above.

5.2 Numerical implementation

The source code listed in Figure 5.2 gives a minimal numerical implementation of our one-dimensional sandpile model, "minimal" in the sense that it favors coding clarity over computational efficiency and coding economy. Note the following:

1. The array `sand[N]` is our discrete variable S_j^n, and contains the quantity of sand at each of the `N` nodes of the lattice at a given iteration. Here this is initialy set to zero at all nodes (line 10).

2. The simulation is structured as one outer temporal loop, and this loop is set up to execute a predetermined number of temporal iteration `n_iter` (starting at line 14);

3. Each temporal iteration begins with an inner loop over each of the $N-1$ pairs of neighbouring nodes on the lattice (starting on line 17). First the local slope is calculated (line 18), then tested for stability (line 19), and wherever the stability criterion is violated, the quantity of sand that must be added or removed from each node to restore stability, as per the redistribution rule (5.4), is accumulated in the array `move` (lines 21–22), *without updating array* `sand` *at this stage*. This update is only carried out once all nodes have been tested, by adding the content of `move` to `sand` (line 27). This *synchronous update* of the nodal variable is important, otherwise a directional bias is introduced in the triggering and propagation of avalanches;

```
# SLOPE-BASED SANDPILE MODEL IN ONE DIMENSION
import numpy as np
import matplotlib.pyplot as plt
#-----------------------------------------------------------------------------
N=101                                     # Lattice size
E=0.1                                     # Peak forcing increment
critical_slope=5.                         # critical slope
n_iter=200000                             # Number of temporal iterations
#-----------------------------------------------------------------------------
sand=np.zeros(N)                          # Lattice, initially empty
tsav=np.zeros(n_iter)                     # Avalanche time series
mass=np.zeros(n_iter)                     # Sandpile mass time series

for iterate in range(0,n_iter):           # Temporal iteration
    move=np.zeros(N)                      # Initialize diplaced sand array

    for j in range(0,N-1):                # Loop over lattice
        slope=abs(sand[j+1]-sand[j])      # Eq (5.3): slope between j,j+1
        if slope >= critical_slope:       # Pair j,j+1 is unstable
            avrg=(sand[j]+sand[j+1])/2.
            move[j]  +=(avrg-sand[j]  )/2. # Eq (5.4) sand moved to/from j
            move[j+1]+=(avrg-sand[j+1])/2. # Eq (5.4) sand moved to/from j+1
            tsav[iterate]+=slope/4.       # Eq (5.7) cumulate displaced mass
    # end of lattice loop

    if tsav[iterate] > 0:                 # At least one node avalanched
        sand+=move                        # Transfer sand
    else:                                 # No avalanche; drive lattice
        j=np.random.random_integers(0,N-1) # Pick random node
        sand[j]+=np.random.uniform(0,E)   # Eq (5.2): add sand increment

    sand[N-1]=0.                          # Eq (5.8): boundary condition
    mass[iterate]=np.sum(sand)            # Sandpile mass at this iteration
    print("{0}, mass {1}.".format(iterate,mass[iterate]))
# End of temporal iteration

# Now plot a simpler version of Figure 5.4
plt.subplot(2,1,1)                        # Set up first plot (top)
plt.plot(range(0,n_iter),mass)            # Sandpile mass vs iteration
plt.ylabel('Sandpile mass')
plt.subplot(2,1,2)                        # Set up second plot (bottom)
plt.plot(range(0,n_iter),tsav)            # Displaced mass vs iteration
plt.ylabel('Displaced mass')
plt.xlabel('iteration')
plt.show()
# END
```

Figure 5.2: A source code in the Python programming language for the one-dimensional sandpile model described in the text. This represents a minimal implementation, emphasizing conceptual clarity over programming elegance, code length, or run-time speed. {`code:sandpile`}

4. Addition of sand at a random node (lines 29–30) only takes place if the lattice was found everywhere stable at the current iteration. This is known as a "stop-and-go" sandpile, and is meant to reflect a *separation of timescale* between forcing and avalanching, the former being assumed to be a much slower process than the latter.

5. At the end of each iteration, the mass of the pile and mass displaced by avalanches, to be defined shortly in eqs. (5.9) and (5.10) below, are stored in the arrays `mass` and `tsav`; these time series will be needed in analyses to follow.

6. Note another piece of Python-specific coding on line 33: the instruction `np.sum(sand)`, using the summing function from the `numpy` library, returns the sum of all elements of array `sand`; this could be easily replaced by a loop sequentially summing the elements of the array.

7. The `matplotlib` intructions on lines 38–45 produce a simplified version of Fig. 5.4 further below.

5.3 A representative simulation

Let's look at what this code does for a small 100-node lattice, initially empty (i.e., $S_j^0 = 0\ \forall j$), with the driving amplitude set at $\varepsilon = 0.1$ and the critical slope at $Z_c = 5$. Figure 5.3 illustrates the growth of the sandpile during the first 10^6 iterations. Recall that sand is being dropped at random locations on the lattice, but in a statistically uniform manner, so that at first the pile remains more or less flat as it grows. However, the "falloff" boundary condition imposed on the right edge drains sand from the pile, so that the pile develops a right-leaning slope, first close to its right edge but gradually extending farther and farther to the left. In contrast, at the left edge the "wall" condition imposed there implies that sand just accumulates without falling off. Consequently the pile remains flat there until the slope growing from the right reaches the left edge. This occurs here after some 850000 temporal iterations. In this *transient phase* the system has not yet reached statistical equilibrium: averaged over many iterations, more sand is added to the pile than is evacuated at the open boundary.

This all make sense and could have been easily expected, doesn't it, given the model's setup? So why having bothered to run the simulation? Well, to begin with, careful examination of Fig. 5.3 reveals that one very likely expectation did not materialize. The dotted line indicates the slope corresponding to the set critical slope $Z_c = 5$. In the statistically stationary state, the pile ends up with a slope significantly *smaller* (here by about 7%) than $Z_c = 5$. This equilibrium slope defines the *angle of repose* of the sandpile. But why is the pile stopping to grow *before* the critical slope is reached ? This is is due to the stochasticity imbedded in the forcing mechanism, which leads to some nodal pairs going unstable before the pile as a whole has reached the critical slope Z_c. As a consequence, the system stabilizes at an average slope smaller than Z_c, approaching Z_c only in the limit $\varepsilon \to 0$. But this is just the beginning of the story.

It will prove useful to define a few global quantities in order to characterize the temporal evolution of the lattice. The most obvious is perhaps *mass*, namely the total quantity of sand in the pile at iteration n:

$$M^n = \sum_{j=0}^{N-1} S_j^n \ . \qquad (5.9) \quad \{\texttt{eq:sandp9}\}$$

Figure 5.4A shows a time series of this quantity, starting at the beginning of the simulation. Mass first grows with time during the transient phase, but eventually saturates at a value subjected to zero-mean fluctuations. These are better visible on the inset, showing a zoom of a small portion of the time series. The shape is quite peculiar. In fact, the line defined by the M^n time series is self-similar, with a fractal dimension larger than unity. On this zoom mass

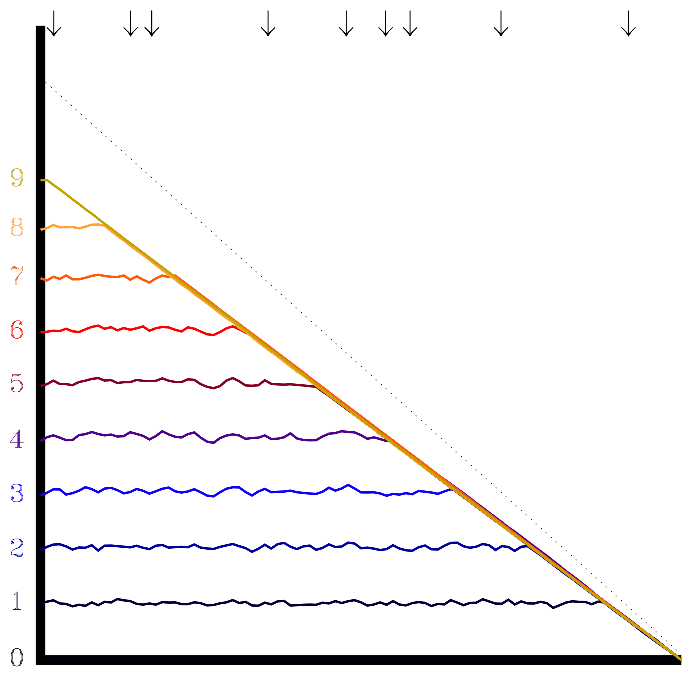

Figure 5.3: Growth of a one-dimensional sanpile constrained by a wall on its left edge, as produced by the code listed on Fig. 5.2, here starting from an empty $N = 100$ lattice and with parameter values $Z_c = 5$ and $\varepsilon = 0.1$. The dotted line indicates a slope of Z_c. Each curve is separated from the preceding one by 10^5 iterations, as color-coded from bottom towards the top. {`fig:pile`}

is seen to grow linearly, at a well-defined rate set by the magnitude of the forcing parameter ε, but this growth is episodically interrupted by sudden drops, occurring when sand is evacuated from the pile when avalanches reach the open boundary at the end of the lattice. The resulting fractal sawtooth pattern reflects the slow, statistically uniform loading and rapid, intermittent discharge. The sandpile is now in a *statistically stationary state*: the mass is ever varying, but its temporal average over a time span much larger than the mean time interval between two successive avalanches remains constant.

Another interesting quantity is the mass displaced at iteration n in the course of an ongoing avalanche:

{`eq:sandp10`}
$$\Delta M^n = \sum_{j=0}^{N-2} \delta S_j^n \ , \tag{5.10}$$

where δS_j^n is given by eq. (5.7). Keep in mind that this quantity is *not* necessarily equal to $M^{n+1} - M^n$, since an avalanche failing to reach the right edge of the sandpile will not lower the total mass of the pile, even though sand is being displaced downslope. Nonetheless, it is clear from Fig. 5.4 that the total mass of the sandpile varies very little even when a large avalanche reaches the right boundary; the largest drop visible in the inset on Fig. 5.4A amounts to a mere 0.2% of the sandpile mass. This is because only a thin layer of sand along the slope is involved in the avalanching process, even for large avalanches. The underlying bulk of the sandpile remains "frozen" after the sandpile has reached its statistically stationary state.

Figure 5.4B shows the segment of the ΔM^n time series corresponding to the epoch plotted

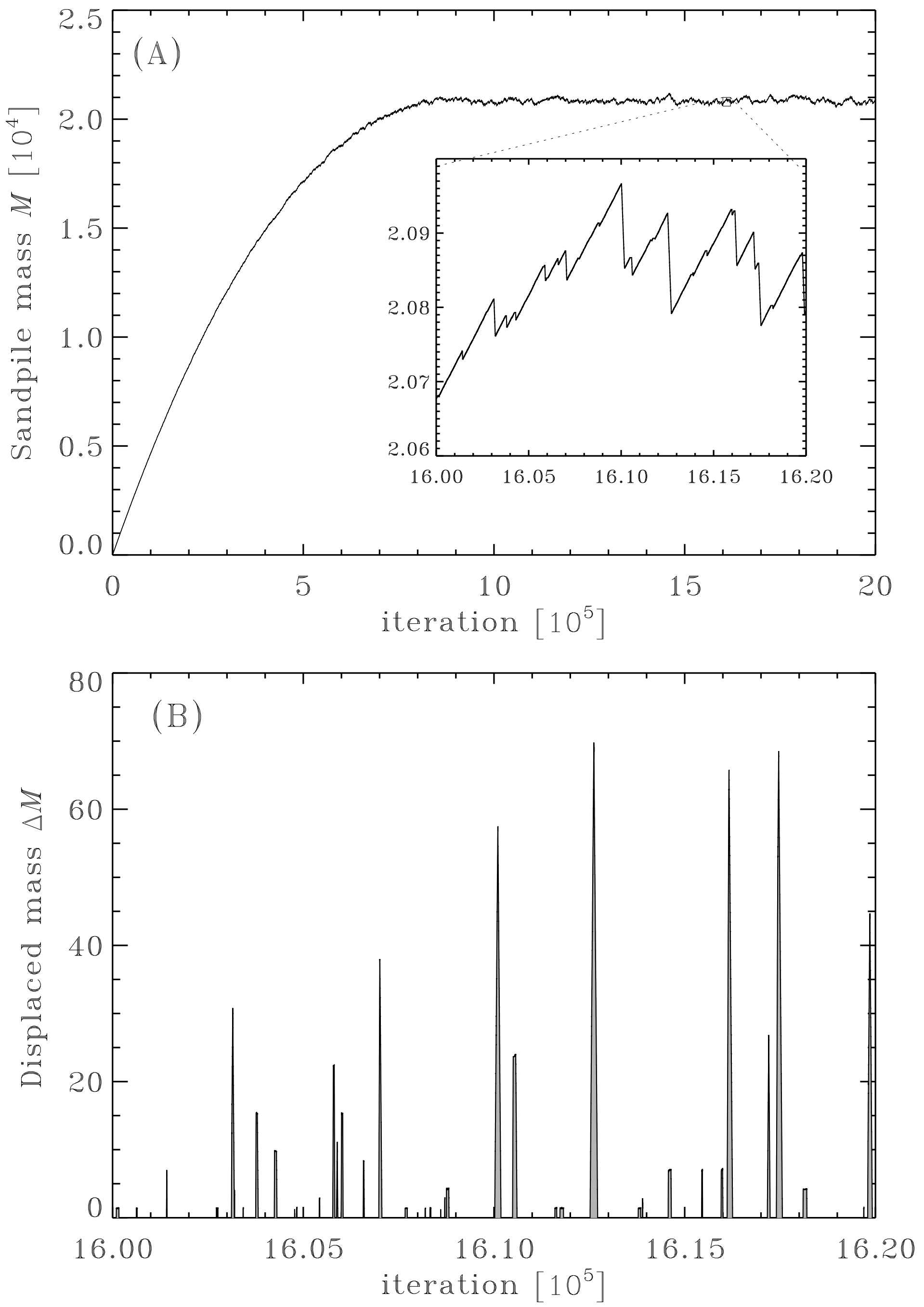

Figure 5.4: Panel (A) shows a time series of total mass M^n, as given by eq. (5.9)), for a simulation with parameter values $N = 100$, $Z_c = 5$, and $\varepsilon = 0.1$ and initial condition $S_j^0 = 0$. The inset shows a zoom of the time series in the statistically stationary phase of the simulation, highlighting its fractal shape. Panel (B) is a time series of displaced mass ΔM^n, as given by eq. (5.10), spanning the same time interval as the inset on panel (A). {`fig:sandpilets`}

in the inset on part (A). This time series is again very intermittent, in the sense that $\Delta M^n = 0$ except during short "bursts" of activity, corresponding to avalanches. These avalanches are triggered randomly, and have widely varying sizes, ranging from one pair of nodes to the whole lattice.

Figure 5.5 illustrates the spatiotemporal unfolding of avalanches over 2000 iterations in the statistically stationary state of the same simulation as on Fig. 5.4. The vertically-elongated images at center and right each show a 1000-iteration segment, the right being the continuation of the central one, with time running vertically upwards. The horizontal is the "spatial" dimension of the 1D lattice, the open boundary being on the right. The square pixellized images on the left are two closeups each capturing the onset and early development of an avalanche. The color scale encodes the quantity of displaced sand, with green corresponding to zero. The purple/pink shades delineate the avalanching regions. Note how avalanches start always at a single nodal pair, following the addition of a sand increment at a single node, and typically expand downslope (here toward the right) as well as upslope (towards left) in subsequent iterations. The smaller avalanches often remain contained within the slope (bottom of middle image), but the larger one typically reach all the way to the open boundary and discharge sand from the pile. The constant inclination angle of propagating avalanches in such diagrams reflects the one-node-per-iteration propagation speed of the avalanching front, as set by the local redistribution rule.

The aggressively pastel color scale used to generate Fig. 5.5 was chosen so as to visually enhance substructures building up within avalanching regions. The most prominent pattern at the lattice scale is checkerboard-like, and simply reflects the fact that the stability and redistribution rules introduce a two-node spatial periodicity in the lattice readjusment. Of greater interest are the long-lived substructures emanating from the avalanching front and propagating vertically upwards in the avalanching regions. These are quite striking on the central and right image on Fig. 5.5. They are triggered by small variations in the slope characterizing stable regions in which the avalanching is progressing. These irregularities are responsible for avalanches, even large ones, sometimes stopping prior to reaching one of the other lattice boundaries. Morphologically, they also bear some similarity to the spatiotemporal structures that can build up in two-states 1D cellular automata of the type investigated in §2.1.

{sec:measavs}

5.4 Measuring avalanches

Figures 5.4B and 5.5 illustrate well the disparity in avalanche size and shape. This is worth looking into in greater detail. We begin by defining three global quantities characterizing each avalanche, all computable from the time series of displaced sand (array `tsav` in the simulation code listed on Fig. 5.2):

1. Avalanche energy[1] E: the sum of all displaced mass ΔM^n over the duration of a given avalanche;
2. Avalanche peak P: the largest ΔM^n value produced in the course of the avalanche.
3. Avalanche duration T: the number of iterations elapsed between the triggering of an avalanche and the last local redistribution that follows;

These three quantities can be easily extracted from the time series of displaced mass (array `tsav` in the Python code listed on Fig. 5.2). The idea is to identify the beginning of an avalanche as a time step `iterate` for which `tsav(iterate)`>0 but `tsav(iterate-1)`$=0$; likewise, an avalanche ends at iteration `iterate-1` if `tsav(iterate-1)`>0 but `tsav(iterate)`$=0$. The following user-defined Python function shows how to code this up:

[1] "Energy" is used here somewhat loosely, yet clearly the redistribution rules involve displacing sand downslope, as indicated by the green boxes on Fig. 5.1, thus liberating gravitational potential energy, and justifying the analogy.

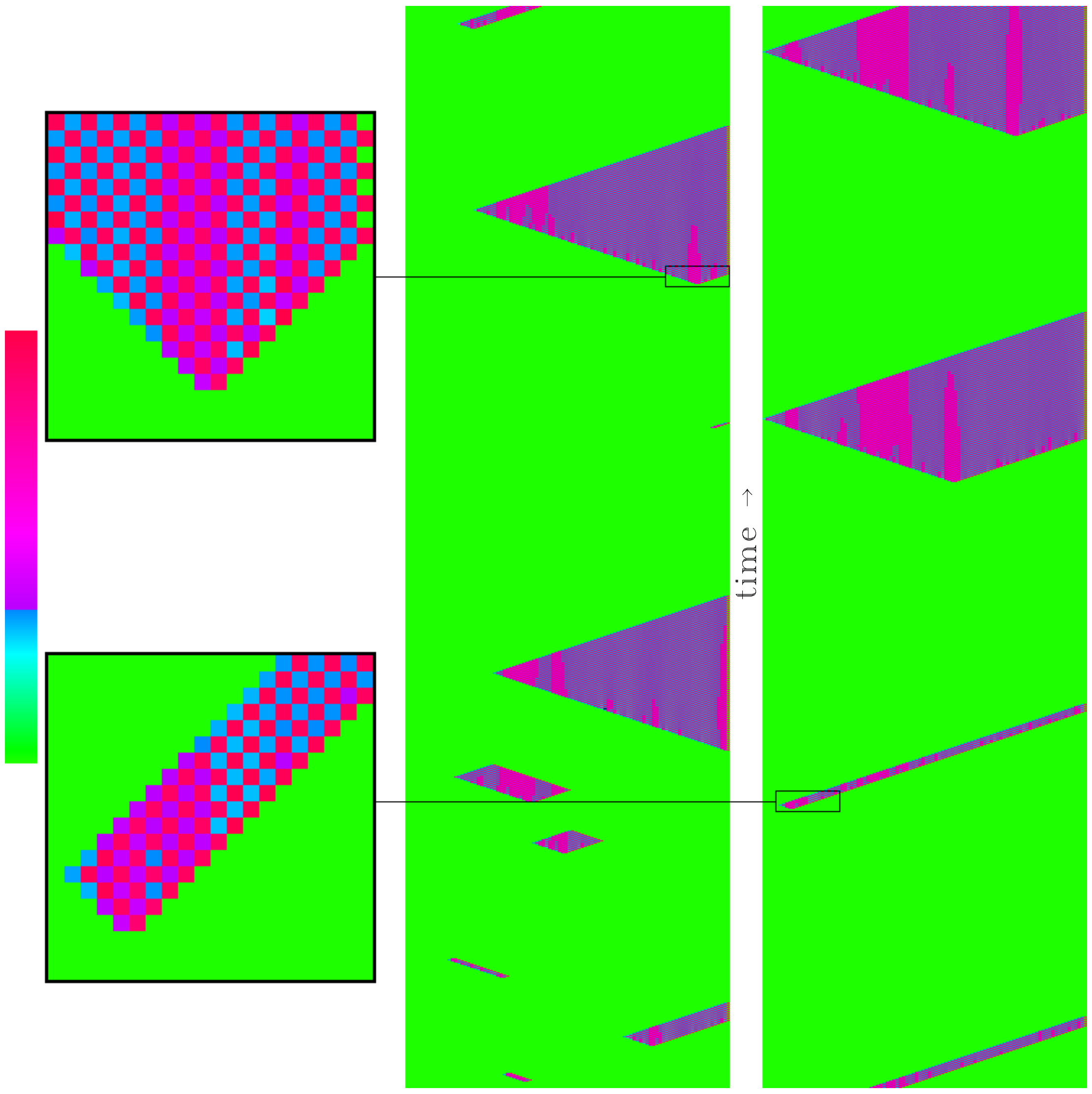

Figure 5.5: Spatiotemporal map of avalanches cascading across the lattice, in a 2000-iteration long segment in the statistically stationary phase of the simulation plotted in Fig. 5.4. The image displays the displaced mass δS_j^n as a function of node number running horizontally, and time running vertically from bottom to top. The open boundary coincides with the right edge of each image. The image on the right is the temporal continuation of that in the middle, and the two pixellized images on the left are closeups on the early phases of two avalanches. Green corresponds to zero displaced mass (stable slope), and shades light blue through purple to red are avalanching regions. This rather unusual pastel color scale was picked to better illustrate the substructures developing within avalanching regions (see text). {`fig:big1dav`}

```
 1 # FUNCTION MEASURE_AV: EXTRACTS ENERGY, PEAK AND DURATION OF AVALANCHES
 2 def measure_av(n_iter,tsav):
 3     n_max_av=10000                         # maximum number of avalanches
 4     e_av=np.zeros(n_max_av)                # avalanche energy series
 5     p_av=np.zeros(n_max_av)                # avalanche peak series
 6     t_av=np.zeros(n_max_av)                # avalanche duration series
 7     n_av,sum,istart,avmax=-1,0,0,0.
 8     for iterate in range(1,n_iter):        # loop over time series
 9         if tsav[iterate] > 0. and tsav[iterate-1] == 0.:
10             sum,avmax=0.,0.
11             istart=iterate                 # a new avalanche begins
12             if n_av == n_max_av-1:         # safety test
13                 print("too many avalanches")
14                 break                      # break out of loop
15             n_av+=1                        # increment avalanche counter
16         sum+=tsav[iterate]                 # cumulate displaced mass
17         if tsav[iterate] > avmax:          # check for peak
18             avmax=tsav[iterate]
19         if tsav[iterate] <= 0. and tsav[iterate-1] > 0: # this avalanche ends
20             e_av[n_av]= sum                # avalanche energy
21             p_av[n_av]= avmax              # avalanche peak
22             t_av[n_av]= iterate-istart     # avalanche duration
23 
24     # end of loop over time series
25     return n_av,e_av,p_av,t_av
26 # END FUNCTION MEASURE_AV
```

This function could be called, for example, after the outer loop in the sandpile code of Fig. 5.2. Note the safety test (lines 12–14) exiting the loop so as to avoid the avalanche counter `n_av` becoming larger than `n_max_av`, which would cause out-of-bounds indexing of the arrays `e_av`, `p_av` and `t_av`. Upon exiting from the loop, the variable `n_av` contains the number of avalanches in the time series array `tsav`, and the arrays `e_av`, `p_av` and `t_av` contain the associated energy E, peak displaced mass P, and duration T of each of these avalanches.

Although large avalanches moving more sand tend to last longer and reach higher peak discharge rates, the quantities E, P and T are correlated only in a statistical sense. Figure 5.6 shows the correlation between avalanche size E and duration T for 15019 avalanches having occurred in a 5×10^6 iteration segment of a simulation on a $N = 1000$ lattice. Overall E does increase with T, but the distribution of avalanche data shows some rather peculiar groupings, most notably along diagonal lines in this correlation plot. Moreover, all data fall within a wedge delimited by lines with slopes of +1 and +2 in this log-log plot.

Consider a lattice everywhere at the angle of repose, with the addition of a small random increment at node j bringing one nodal pair infinitesimally beyond the stability threshold. Equation (5.7) then yields a displaced mass $\delta S^n_j = Z_c/4$; this is the smallest avalanche that can be produced on the lattice; it is the "quantum" of displaced mass (or energy) for this system, hereafter denoted δM_0. Now, suppose that this redistribution destabilizes the downslope pair $(j, j+1)$, but not its upslope counterpart $(j-1, j)$; with the lattice everywhere at the angle of repose, our quantum of displaced mass will move down the slope, one node per iteration, until it is evacuated at the open boundary. If the original unstable nodal pair is M nodes away from the open boundary, this avalanche will have duration $T = M$ and energy $E = M \times \delta M_0$; consequently, $E = \delta M_0 T$, a linear relationship. If the initial avalanche destabilizes both neighbouring pairs but no other pair upslope, then two quanta of mass will move down the slope, leading to $E = 2\delta M_0 T$. And so on for higher numbers of mass quanta. The duration of

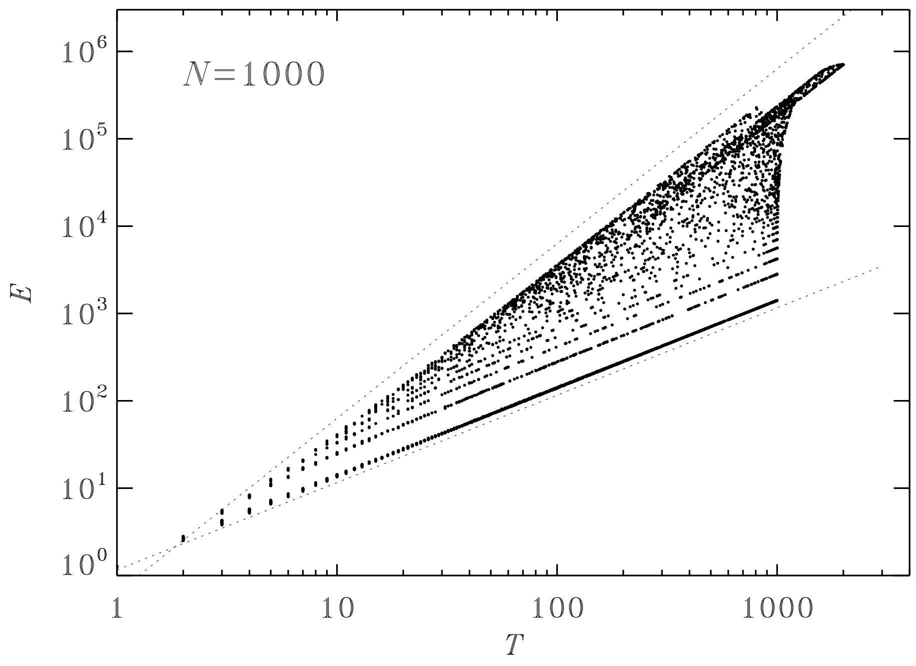

Figure 5.6: Correlation between avalanche size E (displaced mass) and duration T in the statistically stationary phase of a sandpile simulation on a $N = 1000$ 1D lattice. The dotted lines bracketing the avalanche data have slopes of +1 and +2 in this log-log plot, corresponding respectively to the relationships $E \propto T$ and $E \propto T^2$. {`fig:sandpilecorr`}

such avalanches is clearly bounded by the size of the lattice. These are the line-like avalanches on Fig. 5.5, and they map onto the straight line groupings with slope +1 on Figure 5.6. The avalanche whose onset is plotted on the bottom left closeup on Fig. 5.5 belongs to the fourth such family (four mass quanta moving out to the open boundary). These families represent the quantized "energy levels" accessible to the avalanches. The upper bounding line with slope of +2 is associated with avalanches spreading both upslope and downslope; all nodes in between avalanche repeatedly until stabilization occurs at the ends of the avalanche front, or mass is evacuated at the boundary. These are the avalanches taking the form of solid wedges on Fig. 5.5. In such cases the number of avalanching nodes increases linearly with T, so that the time-integrated displaced mass will be $\propto T^2$. The locality of the redistribution rules precludes avalanches from growing faster on this 1D lattice, which then explains why the avalanche energies are bounded from above by a straight line of slope +2 on Figure 5.6. Of course, any intermediate avalanche shape between lines and wedges is possible, and so the space between the two straight lines is also populated by the avalanche data. Incidentally, there is a lesson lurking here: just because a system is deemed to exhibit "complexity" does not mean that some aspects of its global behavior cannot be understood straightforwardly !

Even though the correlations between avalanche parameters exhibit odd structure, their individual statistical distributions are noteworthy. Figure 5.7A and B show the probability density functions (see Appendix C) for E and P, for simulations carried out over lattices of size $N = 100$, 300, 1000 and 3000, but otherwise identical ($Z_c = 5$, $\varepsilon = 0.1$, and redistribution given by eq. (5.4)). The PDFs take the form of power laws, with logarithmic slope independent of lattice size; as the latter increases, the distribution simply extends farther to the right.

This behavior we have encountered before in chapter 4, in the size distribution of clusters on 2D lattices at the percolation threshold. (cf. Fig. 4.8). Here this invariant power-law behavior of materializes only in the statistically stationary phase of the simulation. It indicates that

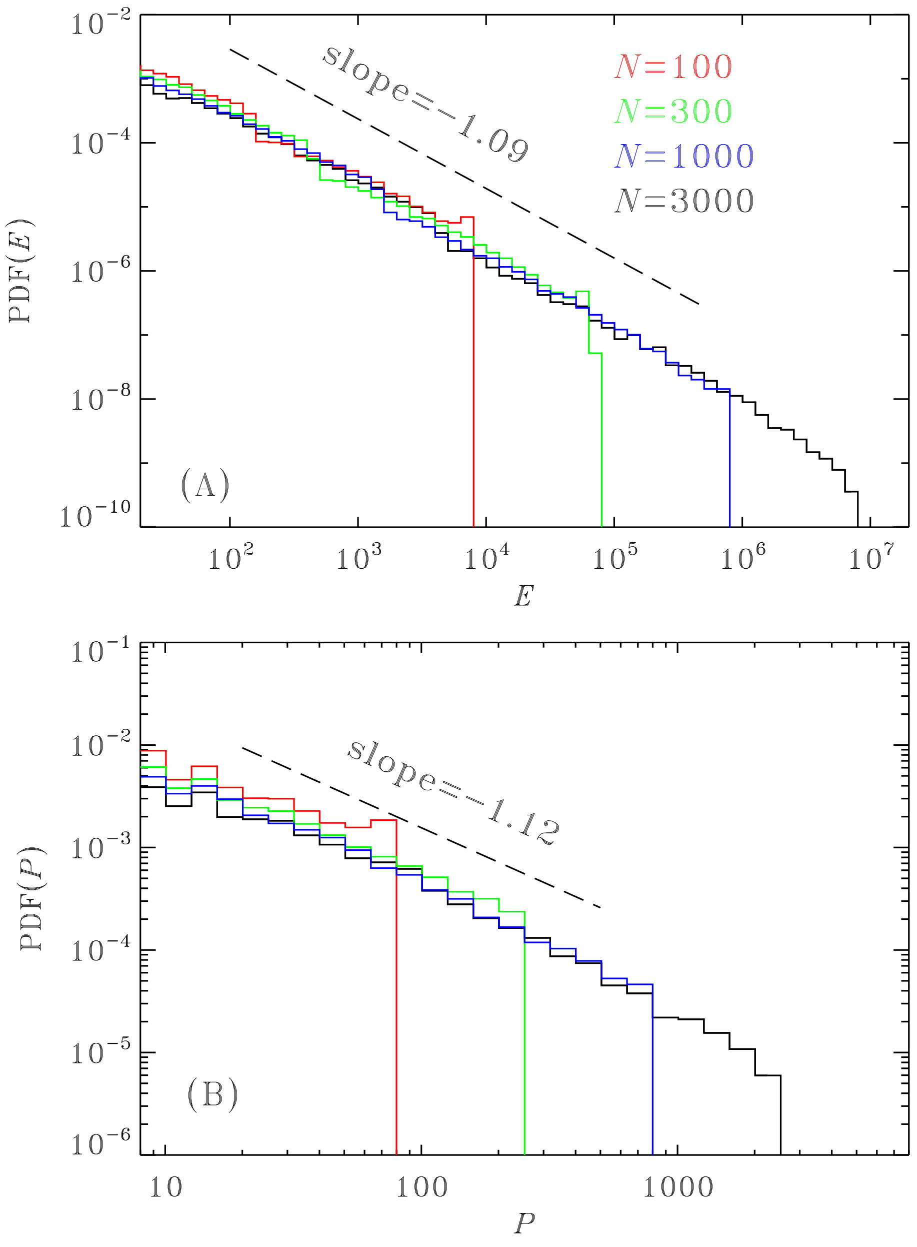

Figure 5.7: Probability density function of (A) avalanche energy E and (B) avalanche peak P, in the statistically stationary states of the sandpile model for varying lattice sizes, as indicated. The PDF of avalanche duration T resembles that for P in (B), except for a steeper logarithmic slope. Note the logarithmic scales on both axes. In all cases the PDFs take the form of power laws, with a flattening at small values of E and P, and a sharp drop at high values, occurring at progressively larger values of E and P for larger lattices. Note, however, that the logarithmic slope is independent of lattice size. Compare this to Fig. 4.8. {`fig:sandpilehisto`}

avalanches are self-similar, i.e., they do not have a characteristic size. This scale invariance reflects the fact that at the dynamical level, the only thing distinguishing a large avalanche from a small one is the number of lattices nodes involved; the same local rules govern the interaction between nodes. But in the percolation context, we also argued that scale-invariance appeared only when the system had reach a critical state; could this also be the case here ?

5.5 Self-organized criticality

{sec:SOC}

It is truly remarkable that of all the possible ways to move sand downslope at the same average rate as sand addition by the forcing rule, so as to achieve a statistically stationary state, our sandpile model "selects" the one characterized by scale-free avalanches. Because many natural systems behave in this manner, the sandpile (real or idealized) has become the icon for avalanching behavior in general, and for the concept of *self-organized criticality* in particular.

We saw in chapter 4, in the context of percolation, that a system is deemed critical when the impact of a small, localized perturbation can be felt across the whole system. Recall how at the percolation threshold, occupying one more node on the lattice can connect two pre-existing clusters, forming a single large cluster spanning the whole lattice; as a result the system suddenly becomes permeable, electrically conducting, whatever, whereas prior to that it was impermeable, or insulating, etc. You should also recall that this extreme sensitivity only materialized at the percolation threshold, so that critical behavior required external fine tuning of a control parameter, which in the case of percolation is the occupation probability p. Moreover, it is only at the percolation threshold that clusters on the lattice exhibited scale invariance (viz. Fig. 4.7).

So where is the criticality here ? With the sandpile, the equivalent of the percolation threshold is the angle of repose of the pile. If the slope is inferior to this, as when the sandpile is still growing, then local addition of sand may trigger small, spatially confined avalanches, but certainly nothing spanning the whole lattice. If the global slope angle is larger than the angle of repose, then the lattice is already avalanching vigorously. Only at the angle of repose can the addition of a small bit of sand at a single random node do anything between (1) nothing, and (2) trigger an avalanche running along the whole slope. However, and unlike with percolation, here the angle of repose is reached "naturally" as a consequence of the dynamical evolution of the system —namely the forcing, stability, and redistribution rules— through interactions between a large number of lattice nodes over time, without any fine tuning of external parameters. The critical state is here an *attractor* of the dynamics. For this reason, systems such as the sandpile are said to be in a state of *self-organized* criticality, to distinguish them from conventional critical systems which rely on external fine tuning of a control parameter.

Much effort has gone into identifying the conditions under which a system can exhibit self-organized critical behavior. At this writing there exist no general theory of self-organized critical systems, but the following characteristics appear sufficient —and possibly even necessary. A system must be:

1. open and dissipative,
2. loaded by slow forcing,
3. subjected to a local threshold instability...
4. ...which restores stability through local readjustement.

However restrictive this may appear, the number and variety of natural systems that in principle meet these requirements is actually quite large. Joining avalanches and other forms of landslides are forest fires, earthquakes, hydrological drainage networks, geomagnetic substorms, and solar flares, to mention but a few. Some of these we will actually encounter in subsequent chapters. More speculative applications of the theory have also been made to species extinction and evolution by punctuated equilibrium, fluctuations and crashes of stock markets, electrical blackouts on power grids, and wars. For more on these topics, see the references listed in the bibliography at the end of this chapter and of chapter **??**.

5.6 Exercises and further computational explorations

1. Verify that the redistribution rule given by eq. (5.4) does lead to eq. (5.7).

2. Modify the 1D sandpile simulation code of Fig. 5.2 to keep track of the mass falling off the pile at its right edge. This will be a distinct avalanching time series from the displaced mass time series `tsav`. Once the statistically stationary state has been reached, use this new "falloff" time series to calculate the corresponding avalanche parameters E, P and T, as in §5.4 above, and construct the corresponding probability density functions (as on Fig. 5.7). Are falloff avalanches scale invariant? How well does the "falloff E" correlate with the "avalanching E" as defined in §5.4 ?

3. Use the 1D sandpile simulation code of Fig. 5.2 to verify that the statistically stationary self-organized critical state is independent of the initial condition; more specifically, try various types of initial conditions such as, for example, an initial sandpile at the angle Z_c, or already at the angle of repose, or an initial sandpile loaded uniformly at some fixed height, etc.

4. Carry out 100-node simulations using different ε ($\varepsilon = 0.01$, 0.1 and 1, say). Are the angles of repose the same ? Making sure to have reached the statistically stationary state before beginning your analyses, construct PDF of slope values (as given by eq. 5.3) as extracted from a single non-avalanching iteration of each simulation; are these PDFs dependent on the value of ε ? Then construct the PDF of avalanche energy E for the same three simulations; are they the same ?

5. The 1D sandpile code listed on Fig. 5.2 is very inefficient from the computational point of view; most notably perhaps, at every iteration it checks *all* lattice nodes for stability, even if a perturbation s has only been added at a single randomly selected node at the preceding iteration (see eq. 5.2). An easy way to improve on this is to modify the start and end points of the loop over the lattice nodes so that stability is checked only at the three nodes $[r-1, r, r+1]$, where r is the random node at which a perturbation is added. The reader with prior coding experience may instead try the really efficient algorithmic approach, which is to keep a *list* of nodes either avalanching or subject to forcing, and run the stability checks and redistribution operations only on list members and their immediate neighbours. This is fairly straightforward in Python, which contains a number of computationally efficient list manipulation operators and functions. This may sound like a lot of work to speed up a simulation code, but when generalizing the avalanche model to two or three (or more) spatial dimensions, such "trick" will mean waiting 10 minutes for the simulation to run, rather than 10 hours (or more). Which takes us naturally to...

6. The Grand Challenge for this chapter is to design a two-dimensional version of the sandpile model introduced herein. Your primary challenge is to generalize the stability criterion (eq. 5.3) and redistribution rule (eq. 5.4) to 2D. Begin by thinking how to define the slope to be associated with a 2×2 block of nodes. Measure the avalanche characteristics E, P and T once the SOC state has been reached, and verify that these are distributed again as power-laws. Are their index the same as in the 1D case ? You should seriously consider implementing in your 2D sandpile code at least the first of the speedup strategies outlined in the preceding exercise.

5.7 Further readings

The concept of Self-Organized Criticality was coined by Per Bak, who became its most enthusiastic advocate as a theory of (almost) everything. His writing on the topic are required reading:

Bak, P., Tang, C., Wiesenfeld, K., *Physical Review Letters*, **59**, 381 (1987),
Bak, P., *How Nature Works*, New York: Springer/Copernicus (1996),

but see also:

Jensen, H.J., *Self-Organized Criticality*, Cambridge University Press(1998).

and, at a more technical level:

Turcotte, D.L., *Rep. Prog. Phys.*, **62**(10), 1377–1429 (1999)
Sornette, D., *Critical phenomena in natural sciences* , Berlin: Springer (2000)
Hergarten, S., *Self-organized criticality in Earth systems*, Berlin: Springer (2002)
Aschwanden, M.J. (ed.), *Self-organized criticality systems*, Berlin: Open Academic Press (2013)

Finally, for a good reality check on the behavior of real piles of real sand:

Duran, J., *Sands, Powders, and Grains*, New York: Springer (2000)

It turns out that real piles of real sand seldom exhibit the SOC behavior characterizing the idealized sandpile models of the type considered in this chapter. However, some granular materials do, including rice grains; see chapter 3 in the book by Jensen listed above.