Excursions in Computing Science: Book 9c. Heat: Histograms and Gases Part IV. Melt, Boil, Condense, Freeze

T. H. Merrett[*]
McGill University, Montreal, Canada

June 10, 2015

I. Prefatory Notes

1. Histograms.
2. Histogram arithmetic.
3. Distributions and densities.
4. Aggregates: the moments of distributions.
5. Quantum distributions: the density matrix.
6. The normal distribution.
7. Expectation, surprise and ignorance.
8. Does ignorance ever decrease?
9. Inside knowledge: the clients of Joe and Sue revisited.
10. Correlation and co-ignorance.
11. Conditional distributions and ignorance.
12. A gas simulation 1: the collisions
13. A gas simulation 2: statistics
14. The Boltzmann and Maxwell distributions.
15. Fluctuations, variations and samples.
16. Entropy.
17. Temperature.
18. Pressure.
19. State function for monatomic gases.
20. Thermostatic equations of state.
21. More on multivariate slopes.
22. Work and heat.
23. Correlation.
24. Collision theory.
25. Mobility, diffusivity and Brownian motion.
26. Three potentials and dissipation.

27. Active transport and biochemistry.
28. Combined transport.
29. Phase transitions. Solids, liquds and gases are *phases* of the substance in question, say H_2O: ice, water, steam. A *change of phase* does not change the molecule, but dramatically changes the collectivity of molecules.

Explanations at the microscopic level are difficult and the mathematics may require long, if interesting, indirect approaches. So I will be direct and use computer simulations based on the microscopic configurations.

I don't even have a definition of "phase transition" to start with, but hope the discussion of this Part will reveal the essentials. For this purpose, I'll start and end quite far afield from melting, boiling, etc.

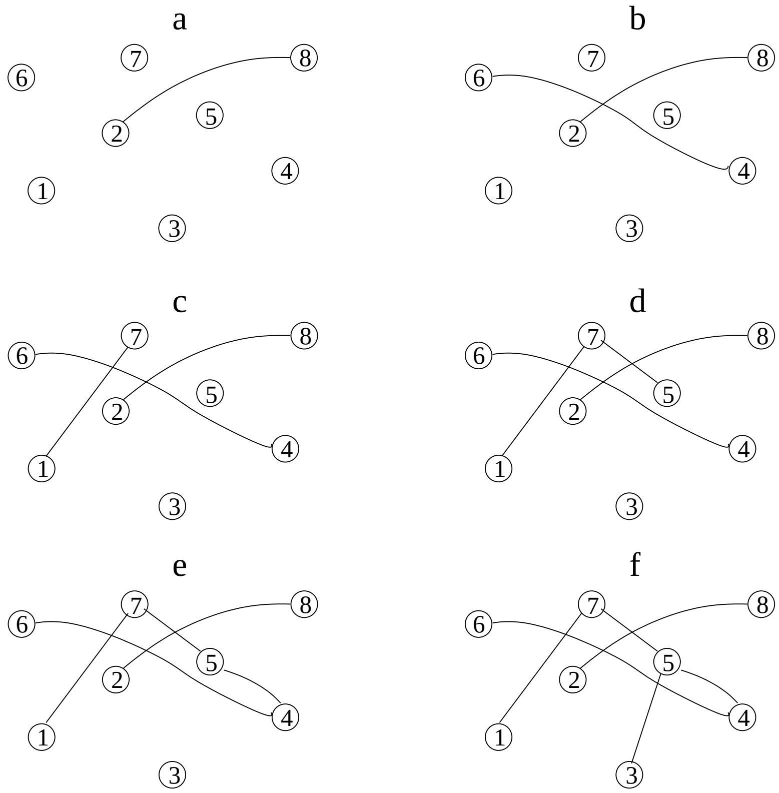
In fact, apart from one scarely microscopic treatment of the liquid and gaseous phases in Note 32, I won't mention any of the four transitions in the title. They seem to be too difficult. At least I do not understand them and the sources I have do not seem to either.

The present Part is the most speculative I've allowed myself to get in all of these Excursions in Computing Science. The results are at best qualitatively correct and may not even be that. Work remains to bring them to quantitative comparison with observation and experiment. So you should take the entire Part as a sketch, hopefully intuitively helpful but in possible need of complete replacement.

I have decided to include it because phase transitions are important, and not only for condensed matter physics. The pedagogical beauty is that there is a lot of room for you to improve it.

30. Phase transitions in random graphs. Imagine scattering a number of buttons on your table, then tying two of them together with a thread, then another two with another thread and so on. How many threads will it take before you can lift all the buttons from the table by picking up only one?

Here are eight buttons and the first six threads.



A button already tied to (an)other button(s) is just as eligible to be tied to a new thread as a completely free button. Indeed, once a number of threads have been tied, the chance of tying a new thread to an already-attached button soon becomes greater than the chance of threading an isolated button.

Moreover, some cluster of threaded buttons will soon outgrow all other clusters, and the dominant cluster will become overwhelmingly likely to receive new links, especially from the other clusters, which will be absorbed by it.

The absorption of all buttons into this dominant cluster will be our phase transition in this quite non-physical example. We will explore its dependence on the probability of forming links (i.e., tying threads) in the first place.

In a MATLAB program we can represent the threads as an n-by-2 matrix, e.g.,

2	8
4	6
1	7
5	7
4	5
3	5
1	5
2	4

and so on for as many threads as we've chosen to tie in. Each row stands for a thread and the order of the rows gives the sequence of tying the threads.

As we tie each thread we must keep a record of the connected buttons. We'll represent each connected set of buttons as a triple structure: the buttons themselves, the connecting threads in a list as above, and the connecting threads as a symmetric square matrix for future reference. (I'll call the buttons "nodes" and the threads "edges" from now on, in keeping with the terminology for graphs.)

Thus, after making the 2–8 connection we have

`nodeSet`	`edgeSet`	`edgeMatrix`
2 8	2 8	$\begin{matrix}0 & 1\\ 1 & 0\end{matrix}$

We'll put up with the redundancy of `edgeSet`.

Going on to make the 4–6 connection gives us two connected components, so two such structures.

`nodeSet`	`edgeSet`	`edgeMatrix`
2 8	2 8	$\begin{matrix}0 & 1\\ 1 & 0\end{matrix}$
4 6	4 6	$\begin{matrix}0 & 1\\ 1 & 0\end{matrix}$

And after adding edge 1–7 we have three.

When we add the fourth edge, 5–7, one of the connected components gets bigger.

`nodeSet`	`edgeSet`	`edgeMatrix`
2 8	2 8	$\begin{matrix}0 & 1\\ 1 & 0\end{matrix}$
4 6	4 6	$\begin{matrix}0 & 1\\ 1 & 0\end{matrix}$
1 5 7	1 7 5 7	$\begin{matrix}0 & 0 & 1\\ 0 & 0 & 1\\ 1 & 1 & 0\end{matrix}$

Note that the `nodeSet` is sorted and that the `edgeMatrix` is implicitly labelled with the sorted nodes.

$$\begin{array}{c|ccc} & 1 & 5 & 7\\ \hline 1 & 0 & 0 & 1\\ 5 & 0 & 0 & 1\\ 7 & 1 & 1 & 0\end{array}$$

The fifth edge merges two of the connected components.

nodeSet	edgeSet	edgeMatrix
2 8	2 8	0 1 1 0

1
4
5
6
7
	4 6
	1 7
	5 7
	4 5
		0 0 0 0 1
		0 0 1 1 0
		0 1 0 0 1
		0 1 0 0 0
		1 0 1 0 0

The eighth edge, in this example, finally brings all nodes into a single connected component.

nodeSet	edgeSet	edgeMatrix
1	2 8	0 0 0 0 1 0 1 0
2	4 6	0 0 0 1 0 0 0 1
3	1 7	0 0 0 0 1 0 0 0
4	5 7	0 1 0 0 1 1 0 0
5	4 5	1 0 1 1 0 0 1 0
6	3 5	0 0 0 1 0 0 0 0
7	1 5	1 0 0 0 1 0 0 0
8	2 4	0 1 0 0 0 0 0 0

(Note that `edgeSet` happens to hold the edges in order of appearance: in general the order of appearance may not be completely captured in `edgeSet`.)

If we were to carry on, the last possible edge would be number 28 = 8!2 and, apart from the 28 edges in `edgeSet`, we would have

nodeSet	edgeMatrix
1	0 1 1 1 1 1 1 1
2	1 0 1 1 1 1 1 1
3	1 1 0 1 1 1 1 1
4	1 1 1 0 1 1 1 1
5	1 1 1 1 0 1 1 1
6	1 1 1 1 1 0 1 1
7	1 1 1 1 1 1 0 1
8	1 1 1 1 1 1 1 0

The MATLAB function to do each of these steps could be

```
connCompCell = addEdge2connComp(edge,connCompCell)
```

Here, `connCompCell` is a MATLAB cell array, n-by-3, with each of the n entries containing the three numerical arrays representing `nodeSet`, `edgeSet` and `edgeMatrix`. Initially `connCompCell` would be empty, and `addEdge2connComp()` would give it a single row for the single connected component. (The remaining, isolated, buttons at this stage are not represented at all, even though they are each technically a "connected component".)

After adding a certain number of edges (8 in the example), `connCompCell` would once again have a single row because there would again be only one connected component: all the buttons.

The reason we're storing the matrix version of the edges, `edgeMatrix`, is that we'd like to have a single number to track the increasing connectivity. That number will be the *average distance* between buttons, for all pairs of buttons: two buttons directly connected by a thread are distance 1 apart; two buttons connected by threads to a single intermediate button, but not directly to each other, are distance 2 apart; and so on, counting links in the shortest paths between the two buttons in each pair.

Isolated buttons do not enter the distance calculation. Thus, after the very first edge is added, the average distance will be 1. After all 28 edges (in this example) are added, the average distance will again be 1. In between, the distance can be quite long: for n buttons the longest distance could be $n - 1$ if the buttons happen to get connected in a single chain, and so the average distance in the chain would be $(n-1)/2$.

The function addEdge2connComp() must itself have several subfunctions a) to add a whole new entry of two nodes and one edge as a connected component, b) to extend an entry by adding one node and one edge, c) to insert one new edge only into a connected component, and d) to add a new edge which combines two existing connected compponents. Lookup routines and a function to append a node and then sort the resulting nodeSet, as well as to build edgeMatrix are needed.

The edgeMatrix can be used to find the pairwise distances via its powers.

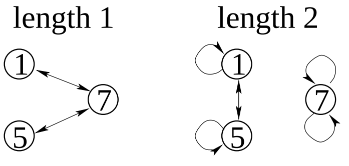
For example the matrix

$$\begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array}$$

immediately gives the distance between the only two nodes as 1: just read the triangle above the diagonal.

For the matrix

$$\begin{array}{c|ccc} & 1 & 5 & 7 \\ \hline 1 & 0 & 0 & 1 \\ 5 & 0 & 0 & 1 \\ 7 & 1 & 1 & 0 \end{array}$$



we must find its square

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

which says that paths of length 2 are as shown: one from 1 to 1, one from 5 to 5, one each from 1 to 5 and 5 to 1, and two from 7 to 7. The original matrix together with the square gives the distances: any nonzero entry in the original matrix indicates a distance of 1 between those two nodes; any nonzero entry in the square indicates a distance of 2 between the two nodes. The resulting distance matrix is

$$\begin{array}{ccc} 2 & 2 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 2 \end{array}$$

Another matrix

$$\begin{array}{ccccc} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{array}$$

must be taken to the fourth power, and yields the distance matrix

$$\begin{array}{ccccc} 2 & 3 & 2 & 4 & 1 \\ 3 & 2 & 1 & 1 & 2 \\ 2 & 1 & 2 & 2 & 1 \\ 4 & 1 & 2 & 2 & 3 \\ 1 & 2 & 1 & 3 & 2 \end{array}$$

The MATLAB function to combine a distance matrix, `A`, already found, with the d th power, `B`, of the original `edgeMatrix` could be

```
AB = distStep(A,B,d)
```

which creates the next-step distance matrix, `AB`, by choosing the entry from `A`, unless it is zero, in which case choosing d if the entry from `B` is nonzero, otherwise 0.

It would be invoked in a loop, after initializing

```
graphd = edgeMatrix
allDist = edgeMatrix
d = 1
```

as follows

```
while < condition >
  d = d + 1
  graphd = graphd × edgeMatrix
  allDist = distStep(allDist,graphd,d)
end
```

To specify the < `condition` > for the `while` loop, and to turn the `allDist` matrix into the average distance between all pairs of nodes, we need

```
[totDist,count] = triSumCount(allDist)
```

which a) sums and b) counts all nonzero entries above the diagonal of `allDist`. Then the average distance can be found from `totDist/count` and the `while` < `condition` > is

```
while count < n × (n − 1)/2
```

where `edgeMatrix`, `graphd` and `allDist` are all n-by-n matrices. This condition requires that a nonzero distance has been found between every pair of nodes.

These functions can be combined into a parent function

```
[allDist,totDist,count] = connectMatrix(edgeMatrix)
```

For the example we've been following, these average distances are, step by step,

After edge		Average distance		
2	8	1	=	1/1
4	6	1	=	2/2
1	7	1	=	3/3
5	7	1.2	=	(1+1+4)/(1+1+3)
4	5	1.9	=	(1+20)/(1+10)
3	5	2	=	(1+31)/(1+15)
1	5	1.75	=	(1+27)/(1+15)
2	4	2.2	=	62/28
	:		:	
<28th edge>		1	=	28/28

The peak of this average distance is what we must explore next. First, we must generate a set of edges, such as those in our example, randomly, given a probability. For instance, the eight edges of the example represent a probability 8/28 of selecting some edge from the $8!2 = 28$ possible edges connecting 8 nodes. If we had listed only seven edges, their probability would have been 7/28 = 1/4, given the same eight nodes.

The function

```
edges = edgesWithProb(n,p)
```

can do this, where n is the number of nodes (buttons) in the configuration and p is the probability of choosing from the $n!2$ possible edges. So `edgesWithProb(8,0.25)` might have generated the first seven edges of our example.

Next, for each probability in some range, say (0:10)/10, we must generate, say, 1000 sets of edges and from these find the average average-distances and their standard deviation.

I wrote a function

```
[avgDists,stdDists] = erdosRenyiProbLoop4()
```

which repeatedly invokes

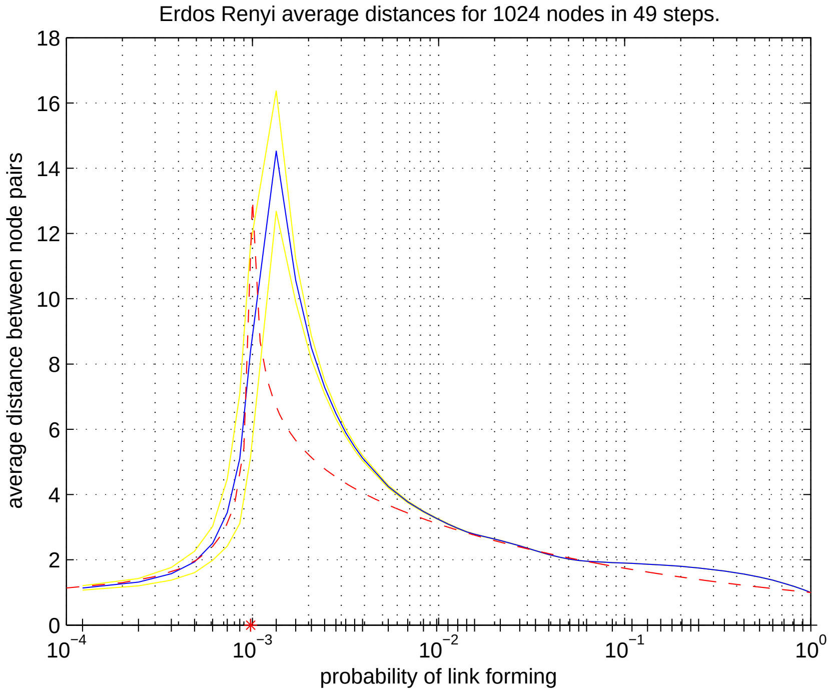
```
[avgDists] = erdosRenyiProbDist(n,p)
```

over n nodes for a range of probabilities, iterating the calculations in a complicated way for each probability, p. The output, `avgDists`, from the inner function is the average distance in the graph resulting after a set of edges has been generated corresponding to the probability p. The output of the same name, `avgDists`, from the outer function is a set of averages of the inner `avgDists`, averaged over a given number of iterations for each probability in a range specified in the outer function. The corresponding standard deviations are `stdDists`.

The input parameters of the outer function reflect a complicated dance I did to speed up execution (since I did not fuss much about running speeds of all the internals discussed above): execution takes a long time when many edges are generated at high probabilities, so I spaced the probabilities logarithmically (to base 4), and also reduced the numbers of iterations for the higher probabilities.

The parameters, which I omitted above, must specify n the number of nodes, the number of unequally spaced probabilities, and the number of iterations over each probability.

A run with 1024 nodes over 49 different probabilities took several days.



The blue plot is the average of average distances for these 49 probabilities. The yellow plots are these averages plus and minus one standard deviation. We can see the distance limits of 1 at both low and high probabilities and a clear spike at a *critical probability*.

I've marked the theoretical critical probability, $1/(n-1)$, with a red asterisk, and I've plotted in red a "critical function" which I now describe. First though, note that the fit is not as bad as first appears. a) The section of the blue curve directly above the red asterisk leading up to the spike (and the corresponding sections of the yellow curves) is an artefact of the plot and could be blanked out: as it is the spike is misleadingly shown as the highest point on the blue curve, but calculations for probabilities closer to the critical probability would give higher peaks closer to it.

b) The height of the red curve is arbitrary and could be adjusted for a better fit.

The red curve is plotted by

```
aCriticalFunction(gamma1,gamma2,n)
```

and essentially shows

$$\begin{aligned} &(p_c - p)^{-\gamma_1} && p < p_c \\ &(p - p_c)^{-\gamma_2} && p > p_c \end{aligned}$$

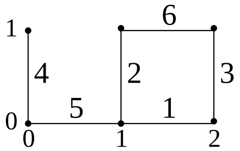
(The critical probability $p_c = 1/(n-1)$ and for $n = 1024$, I used `gamma1` $= 0.24 =$ `gamma2`.)

"Critical exponents" such as γ_1 and γ_2 figure importantly in some phase transitions, so the red additions to the plot are an introduction to this idea.

31. Point-to-point resistance in a network. We move to a more physical example by replacing arbitrary graphs by crystal lattices and by measuring point-to-point resistances instead of point-to-point distances. This discussion is based on the Excursion *Kirchoff lattices* of Part III.

Here we will construct *partial lattices*, i.e., lattices with some links missing, and we will find resistances between not just two specified endpoints but between all pairs of possible endpoints. Averages over all such point-to-point resistances will be our statistical quantity.

As an example, let's build up the following resistance network, edge by edge, with the edges in the order 1 to 6 as shown.



The coordinates of each node are two numbers in this 2-dimensional lattice. The edges (resistors, all of the same resistance), in order of insertion, are thus

```
   edges
1  0  2  0
1  0  1  1
2  0  2  1
0  0  0  1
0  0  1  0
1  1  2  0
```

We can suppose that these edges were generated from the full set of seven edges for `crystal2(2,1,4)` (see Excursion *Kirchoff lattice*) using

```
edges = edgesWithProb(p,crystal2(2,1,4))
```

and probability $p = 6/7$.

This gives an example of the partial lattice mentioned in the first paragraph: edge 1011 is missing from the full lattice.

As in the previous Note, we'll build up the network one edge at a time and maintain connected components in a cell array `connCompCell`, initially empty, using

```
connCompCell = addEdge2connComp(edge,connCompCell)
```

(Notice that I'm re-using some of the function names from the previous Note for corresponding operations on this resistance network: the MATLAB functions with these same names are *not* the same. Ditto for the name `connCompCell`.)

Here is what happens on the first iteration

```
connCompCell = cell(0,3)
connCompCell = addEdge2connComp(edges(1,:),connCompCell)
connCompCell{:}
```

```
nodeSet    edgeSet      resistMatrix
1   0    1  0  2  0    0       1
2   0                  0       0
```

Here, `nodeSet` and `edgeSet` play the same roles as their namesakes in Note 30, but each is doubled because of the 2-dimensional coordinates.

On the other hand, `resistMatrix`, while describing connectivity, also gives the node-to-node resistance. In this case there is one resistor of value 1 (as in Excursion *Kirchoff lattice* we suppose every edge to be a resistor of value 1) between node 1 0 and node 2 0.

The second iteration

```
        connCompCell = addEdge2connComp(edges(2,:),connCompCell)
        connCompCell{:}
```

adds edge 1 0 1 1 to this

```
nodeSet    edgeSet      resistMatrix
2   0    1  0  2  0    0  1    2
1   0    1  0  1  1    0  0    1
1   1                  0  0    0
```

and you can confirm that the resistance from 1 0 to 2 0 is 1, as is the resistance from 1 0 to 1 1, but the two-edge resistance from 2 0 to 1 1 is 2, as the matrix shows.

We see that the `nodeSet` may be rearranged, that it doesn't matter in which direction we take the resistance, and that the order of the `resistMatrix` rows and columns is determined by the order of nodes in `nodeSet`.

A third iteration

```
        connCompCell = addEdge2connComp(edges(3,:),connCompCell)
        connCompCell{:}
```

grows the connected component by one more node and edge

```
nodeSet    edgeSet      resistMatrix
1   1    1  0  2  0    0  1  2   3
1   0    1  0  1  1    0  0  1   2
2   0    2  0  2  1    0  0  0   1
2   1                  0  0  0   0
```

but the fourth iteration

```
        connCompCell = addEdge2connComp(edges(4,:),connCompCell)
        connCompCell{:}
```

starts a second connected component

```
nodeSet    edgeSet      resistMatrix
1   1    1  0  2  0    0  1  2   3
1   0    1  0  1  1    0  0  1   2
2   0    2  0  2  1    0  0  0   1
2   1                  0  0  0   0

0   0    0  0  0  1    0  1
0   1                  0  0
```

The fifth iteration merges the two connected components

```
        connCompCell = addEdge2connComp(edges(5,:),connCompCell)
        connCompCell{:}
```

```
nodeSet    edgeSet      resistMatrix
0   1    0  0  0  1   0  1  2  3  3  4
0   0    0  0  1  0   0  0  1  2  2  3
1   0    1  0  2  0   0  0  0  1  1  2
1   1    1  0  1  1   0  0  0  0  2  3
2   0    2  0  2  1   0  0  0  0  0  1
2   1                 0  0  0  0  0  0
```

Finally, the sixth iteration closes a cycle

```
        connCompCell = addEdge2connComp(edges(6,:),connCompCell)
        connCompCell{}
```

```
nodeSet    edgeSet            resistMatrix
0   0    0  0  0  1     0  1  1   7/4   7/4    2
0   1    0  0  1  0     0  0  2  11/4  11/4    3
1   1    1  0  2  0     0  0  0   3/4   3/4    1
2   0    1  0  1  1     0  0  0    0     1    3/4
1   1    2  0  2  1     0  0  0    0     0    3/4
2   1    1  1  2  1     0  0  0    0     0     0
```

From this sequence of adding edges one at a time you can see that `resistMatrix` might be calculated afresh each step using `buildSolveKirchoff()` from Excursion *Kirchoff lattice*. The structure, and indeed the code, of `addEdge2connComp()` is almost identical to its namesake in Note 30, delegating tasks respectively to

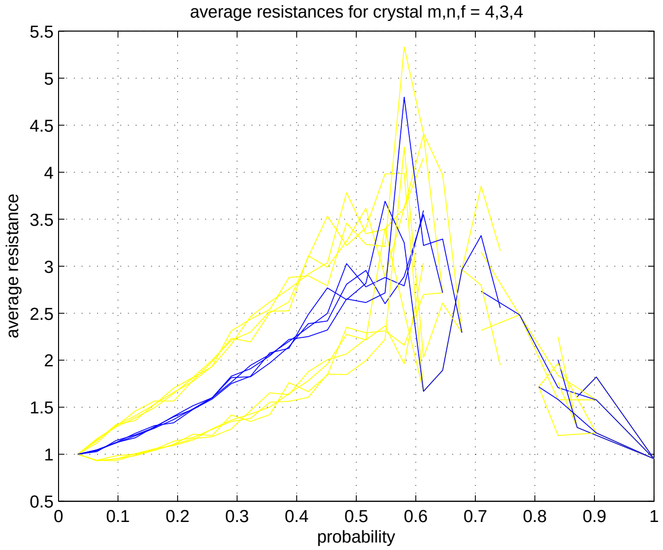
```
        addEntry()
        extendEntry()
        insertEdge()
        mergeEntries()
```

(each of which does quite different things from its namesake in Note 30. Of these, only `insertEdge()` actually uses `buildSolveKirchoff()`. Functions `extendEntry()` and `mergeEntries()` can actually recombine existing resistance matrices, and `addEntry()` just produces the starting matrix

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.)$$

Now we can build the resistance matrix between all pairs of nodes in a partial-lattice subset of a full 2-D lattice, with the edges selected randomly according to a given probability. The average resistance over all pairs can be found from this.

The next step is to choose a set of different probabilities and to repeat the above procedure enough times for each probability to get a meaningful average of averages and the corresponding standard deviations. My code had problems with the higher probabilities, but here is a result for a range of ten probabilities each repeated 1000 times on a 4-by-3 lattice of squares.



I ran it four times to cover the range. We see a peak in average resistance at a critical probability, and behaviour to either side which could be consistent with critical exponents as in Note 30.

32. Van der Waals. If you squeeze a gas enough, its molecules compress until it is indistinguishable from a liquid. In this Note we look at a classical adjustment to the Ideal Gas Law which limits the compressibility and allows molecules to attract each other. This is not a microscopic model and so this Note is not a probabilistic simulation, unlike the other Notes in this Part.

The Ideal Gas Law

$$PV = Nk_BT$$

can be written for specific volume $v = V/N$

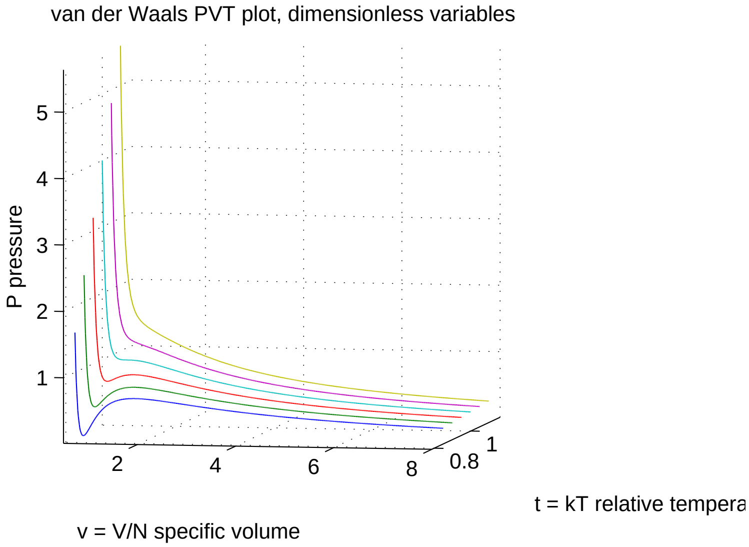
$$Pv = k_BT$$

The van der Waals adjustment is 1) to limit the smallest volume possible for the whole gas to $V_0 = Nb$ (i.e., b is the smallest volume per molecule) and 2) to reduce the pressure by an attractive force a/v^2 (which is inversely proportional to the specific volume squared because the closer the molecules are squeezed together the stronger their mutual attraction can be, and, the attraction being between pairs of molecules, there is one volume factor for each molecule).

The van der Waals gas law is thus

$$(P - \frac{a}{v^2})(v - b) = k_BT$$

This involves two parameters, but fortunately the equation can be rearranged into a dimensionless form. Here is what the isotherms (constant-temperature lines) look like.



We see that below a certain temperature the isotherms do a dip while above it they do not. This is called the *critical temperature*. The critical isotherm (the P-v line at this temperature) contains a *critical point* which is characterized by $\mathrm{slope}_v P = 0$ on the line at this point and also by $\mathrm{slope}_v \mathrm{slope}_v P = 0$ because it is a "point of inflection" of the curve (see Excursion *point of inflection* for this Part).

The critical point fixes each of temperature, $T = T_c$, volume, $v = v_c$, and pressure, $P = P_c$. Setting to zero the slopes at $T = T_c$, $\mathrm{slope}_v P$ and $\mathrm{slope}_v^2 P = \mathrm{slope}_v \mathrm{slope}_v P$, gives

$$\begin{aligned} v_c &= 3b \\ k_B T_c &= \frac{8}{27}\frac{a}{b} \\ P_c &= \frac{a}{27b^2} \end{aligned}$$

(and also the *dimensionless ratio*

$$\frac{k_B T_c}{P_c v_c} = \frac{8}{3}$$

as opposed to 1 for an Ideal Gas).

This calculation allows us to convert the van der Waals equation into dimensionless units, $T/T_C, v/v_c$ and P/P_c, and these are what I plotted above:

$$P = \frac{8T}{3v-1} - \frac{3}{v^2}$$

with the critical point $(T, v, P) = (1, 1, 1)$.

There is a problem with these van der Waals curves. Below the critical temperature, volume is not a function of pressure: there are places on each subcritical isotherm where two different volumes are shown for one pressure. This is a region of phase transition, and the isotherms must be corrected so that pressure is constant. For example, water boils at constant temperature and pressure ($T = 100^o$C at sea-level pressure of 1 atmosphere).

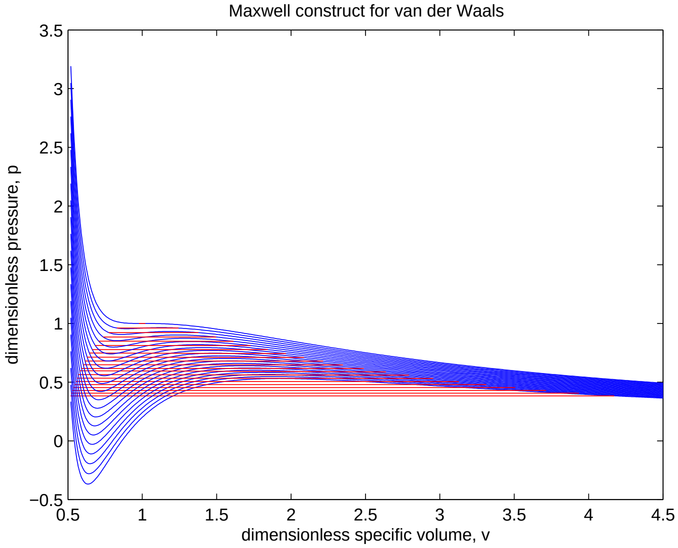
The change to the van der Waals' isotherms to make them realistic is called the Maxwell construction. It consists of replacing the S-shaped part of each isotherm by a constant-P (horizontal) line between the two volumes v_1 and v_2 which give rise to the same pressure.

To choose those two volumes, Maxwell specified that the overall energy should not be changed by his construction. That is, the area, Pv (which is energy), between the S-curve and the horizontal line should total zero: the positive S-curve-to-Maxwell area at lower volumes should equal the negative S-curve-to-Maxwell area at higher volumes.

So we have two expressions whose zeros we must find with respect to two variables v_1 and v_2:

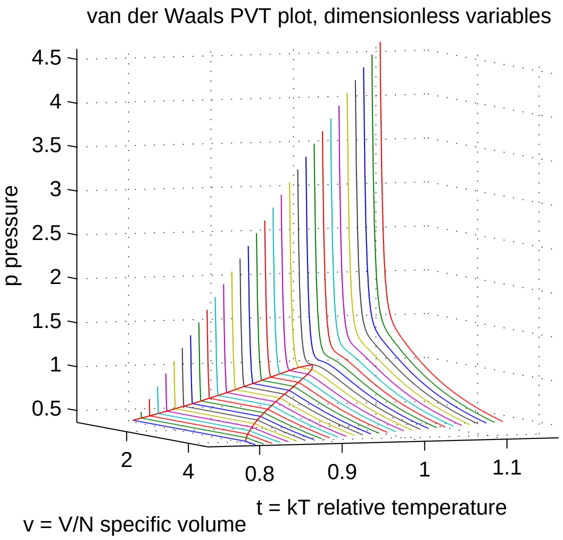
$$\text{antislope}_v P \text{ from } v_1 \text{ to } v_2$$
$$P(v_2) - P(v_1)$$

The Excursion *Newton's method 2* of Part I tells us how to solve these once we've worked out the antislope.



Here is the result. The van der Waals' isotherms are shown as blue lines. The Maxwell construction for each is shown as a red horizontal line. Look at the lowest blue and lowest red lines as an example: you should see that the blue-below-red area on the left equals the red-below-blue area on the right, so the energies represented by the two curves are the same.

Here are the same data shown as a 3-D PVT surface.



The phase transition region has been outlined in red.

The way to interpret this region is as a mixture of two phases, liquid and gas, all at the same pressure for any given temperature: at lower volumes the mixture is mostly liquid; at higher volumes it is mostly gas.

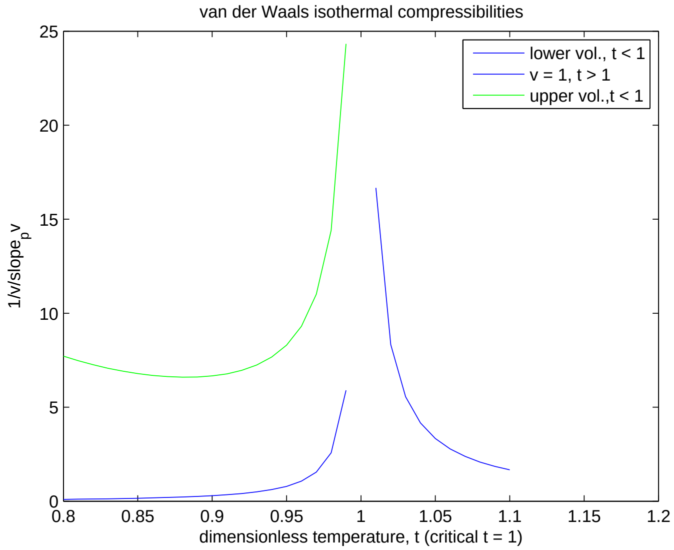
At still lower volumes, below the transition region, the pressure shoots up immensely as the volume decreases (or, it takes immense squeeze to reduce the volume a little): the compressibility is small, that of a liquid. At volumes above the transition region, pressure does not vary much: the gas is very compressible.

The critical point is the top of the red loop marking the transition region. Pressure is fairly high here and the substance has been squeezed to close to its minimum volume: the distinction between gas and liquid is lost at and above the critical temperature.

We can study the compressibility along the isotherms (isothermal compressibility) near the critical point. The compressibility varies inversely as the $\text{slope}_v P$ because the steeper the slope the harder it is to squeeze the substance. It also varies inversely as the volume because compressibility should be per unit volume or else it becomes arbitrarily big with arbitrarily large volumes.

$$\kappa = \frac{\text{slope}_P v}{v} = \frac{1}{v\ \text{slope}_v P}$$

Thus the smaller the slope the higher the compressibility, and at the critical point where $\text{slope}_v P = 0$ it is infinite. As we move away from the critical temperature, staying on the red boundary of the transition region—on either the low-volume or high-volume sides—the slope gets steeper and the compressibility smaller.



We can see that this plot behaves near the critical temperature something like the critical exponent of Note 30.

33. Sublimation. The closest I can come to a microscopic simulation of an ordinary phase transition is that from a crystal lattice to a gas: the direct solid-to-gas transition as shown by dry ice or by snow on a sunny spring day.

The critical probability in this simulation will also be a critical temperature, from the relationship

$$p_c = \frac{1}{Z} e^{-\Delta E/(k_B T_c)}$$

with Z the partition function and ΔE the microscopic energy at which the phase transition takes place.

The original solid takes the form of a two-dimensional lattice with square symmetry and cyclic boundary conditions. This is represented as an array of 0s and 1s, with 1 meaning a molecule is present at the site and a 0 meaning the molecule has left the site and become a molecule in the gas. The gas is not represented because it is assumed to be elsewhere.

Thus a 3-by-3 crystal would appear as

$$\begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array}$$

We will need to count the number of nearest neighbours for each site because we are going to assume a molecule is held in place by the combined restraint of its nearest neighbours, each acting on it with a potential energy ϵ. Non-nearest neighbours are supposed too far away to have any influence.

The numbers of n.n. (nearest-neighbours) for the above lattice are

$$\begin{array}{ccc} 4 & 4 & 4 \\ 4 & 4 & 4 \\ 4 & 4 & 4 \end{array}$$

We can see why the centre site has 4 n.n—the sites to its left, right, up and down are all occupied. The cyclic boundary conditions also make this true of every other site. For example, site (1,1) has n.n. at sites (1,2), (2,1), (1,3) and (3,1). Or, for an n-by-n crystal, these neighbours will be at sites (1,2), (2,1), (1,n) and (n,1).

Now suppose the centre molecule sublimes, i.e., leaves the crystal to join the vapour. We have

$$\begin{array}{ccccccc} \multicolumn{3}{c}{\texttt{sqlatt}} & \quad & \multicolumn{3}{c}{\texttt{numNN}} \\ 1 & 1 & 1 & & 4 & 3 & 4 \\ 1 & 0 & 1 & & 3 & & 3 \\ 1 & 1 & 1 & & 4 & 3 & 4 \end{array}$$

(where we could have put a 4 in the centre position of `numNN` because that site has 4 occupied n.n. sites, but I left the entry blank because that molecule has gone and thereby left the calculations).

The binding energy at eacn site is

$$\Delta E = \texttt{numNN} \times \epsilon$$

so that a site with 4n.n. has $\Delta E = 4\epsilon$ and so on. A site with 0 n.n.s has $\Delta E = 0$ and so the molecule is free to leave the lattice (probability of leaving = 1) and we suppose it has done so.

Thus we'll need to calculate the probability that each molecule leaves the crystal (sublimes). For the second example above the probabilities are

$$p \propto \exp\left(- \left(\begin{array}{ccc} 4 & 3 & 4 \\ 3 & 0 & 3 \\ 4 & 3 & 4 \end{array} \right) \frac{\epsilon}{k_B T} \right)$$

which depends on the temperature. To get a semirealistic idea of temperatures which may be involved, we'll take ϵ to be 10.4 meV, an energy which could be of the right magnitude for Argon.

The "proportional to", $\propto$, in the above equation becomes = when we divide by the partition function, which also depends on the number of nearest neighbours. If we take the numerator of the probability to be, say,

$$\text{numerator} = \exp(-\tfrac{3\epsilon}{k_B T})$$

for the case that a molecule with 3n.n. has acquired sufficient energy (3ϵ) to escape, then the other case is that it remains at the bottom of the potential well (energy 0), which is exp(0) = 1; so the denominator

$$\text{denominator} = 1 + \text{numerator}$$

and the probability of escape is

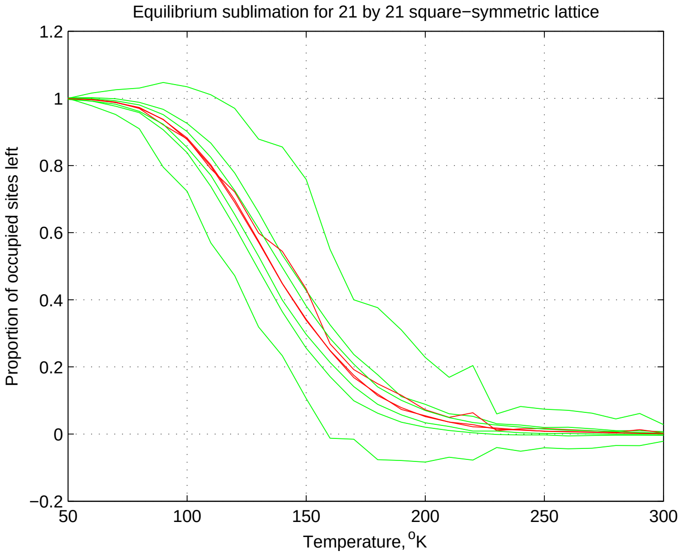
$$p = \frac{\exp -\frac{3\epsilon}{k_B T}}{\exp 0 + \exp -\frac{3\epsilon}{k_B T}}$$

At a given temperature, starting with a solid crystal, we calculate the probabilities of escape from each site and iterate the process of simulating escapes and rebuilding the lattice without the sublimed molecules until we reach equilibrium.

We do this sufficiently often at each of a range of temperatures that reliable averages and standard deviations of the proportion of occupied sites can be built up and plotted.

Here are those combined plots for "argon" (ϵ = 10.4 meV) at temperatures from 50 to 300^{o}K with 100 iterations at each temperature for a) 3-by-3, b) 11-by-11 and c) 21-by-21 lattices:

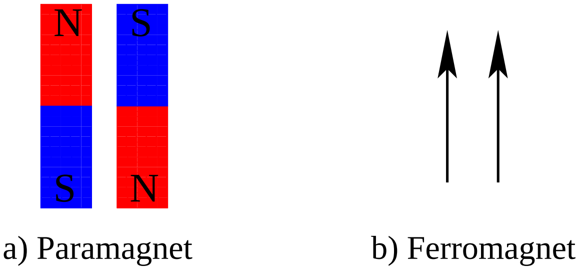
```
prpsLeft = sublimLoops(3,3,10.42,[50:10:300],100)
prpsLeft = sublimLoops(11,11,10.42,[50:10:300],100)
prpsLeft = sublimLoops(21,21,10.42,[50:10:300],100)
```



The sloppy average (red curve) and the largest spread of standard deviation (green curves) are the 3-by-3 lattice. The averages for the 11-by-11 and the 21-by-21 are indistinguishable, but the standard deviations are tighter in the latter case.

In this kind of phase transition, from 100 % to 0 % solid, nothing obviously goes "to infinity" the way the critical exponents showed in previous Notes. What actually does go "to infinity" in this transition is the inverse of the rate of increase of internal energy (and hence temperature) of the solid as heat is poured in: the model should include various intermediate levels of microscopic energy for each molecule below the actual point of sublimation. We would then expect a region of zero change in these energy levels at the highest excited state of the solid: the sublimation step no longer contributes energy to the solid. I do not understand this well enough to write the programs.

34. Ferromagnets. Two bar magnets placed side by side will line up "north" to "south". This is their configuration of least energy: you must force them into any other configuration, You'll do the most work trying to align them "north" to "north" (equivalently "south" to "south"). The preferred alignment is shown on the left (and labelled "paramagnet").



Atoms often behave as quantum bar magnets, owing to the spins of their electrons (and indeed of the protons in their nucleii): classical electromagnetism shows that a circulating charge will have a magnetic effect, and a "spinning" electron or proton does too—opposite in sign for the two types of particle because of their opposite charges.

A *paramagnet* consists of atoms aligned the way the bar magnets are in the figure. For reasons of the configuration of the electrons around the atoms of some elements, notably iron, *ferromagnets* prefer the "north" to "north" alignment of neighbouring atoms, as shown on the right of the figure: here the "north"–"south" bar magnet is symbolized as an arrow pointing "north". These arrows can also represent the spin of the atom, with the caveat that if the atomic magnet results from a negative charge then the spin is in the opposite direction from the "south"-to-"north" arrow.

Electron- and proton-spin being a quantum effect, these arrows have only two relative alignments, supporting or opposing. This is unlike the infinity of directions we can point a bar magnet in relative to another bar magnet.

Consequently, it is easy to write down the energy of interaction of two spins σ_1 and σ_2

$$-J\sigma_1\sigma_2$$

where $\sigma_1 = 1$ for spin "up" and -1 for spin "down" and where J is the energy of interaction, positive for a ferromagnet and negative for a paramagnet. These signs will favour opposing alignment for paramagnets and supporting alignments for ferromagnets.

A paramagnet does not become magnetized except under the influence of an external magnet which overcomes the tendency to opposing alignment: if the some of the spins that point down, say, are coerced to point up, then the body will be magnetized in its own right. This disappears on removing the external influence.

Under an external magnetic "field", H, each spin σ has an additional energy

$$\mu H\sigma$$

aligning it with the field, where μ is a parameter analogous to J. In the simulation I take μH to be a single parameter.

A ferromagnet, on the other hand, magnetizes spontaneously, without external influence. But if the temperature is increased, thermal agitation will overcome this by averaging out the spin directions. So we can expect a temperature-dependent phase transition from ferromagnet to paramagnet.

An external magnetic field will modify this behaviour but only slightly unless the field is enormous.

We can simulate all this in a two-dimensional lattice of square symmetry with the simplifying supposition that only nearest-neighbour spins interact via the $-J$ term—just as we did for sublimation in the previous Note. The four nearest-neighbours of any site in the lattice combine in five possible ways to give the interaction energy.

nn configuration	energy $-J\sigma_{\text{nn}}$
4 spins up	$-4J$
3	$-2J$
2	$-0J$
1	$2J$
0	$4J$

shown assuming the spin at the site itself is up (change the signs if it is down). I'll take $J = 3\times 11.9$ meV in an attempt to make the 2-D simulation reproduce the 1043^{o}K transition temperature for the 3-D "body-centred-cubic" lattice of iron.

As in the sublimation simulation of the previous Note, I used periodic boundary conditions, so every site is effectively in the interior of the lattice and there are no surface effects. (Besides, it's easier to code.)

The probability of a site flipping its spin to up is given by the interaction energy and is proportional to

$$\text{numerator} = \exp((J\sigma_{\text{nn}} - \mu H)\sigma/(k_B T))$$

Since the only alternative, flipping to spin down, is the inverse of this, the partition function is

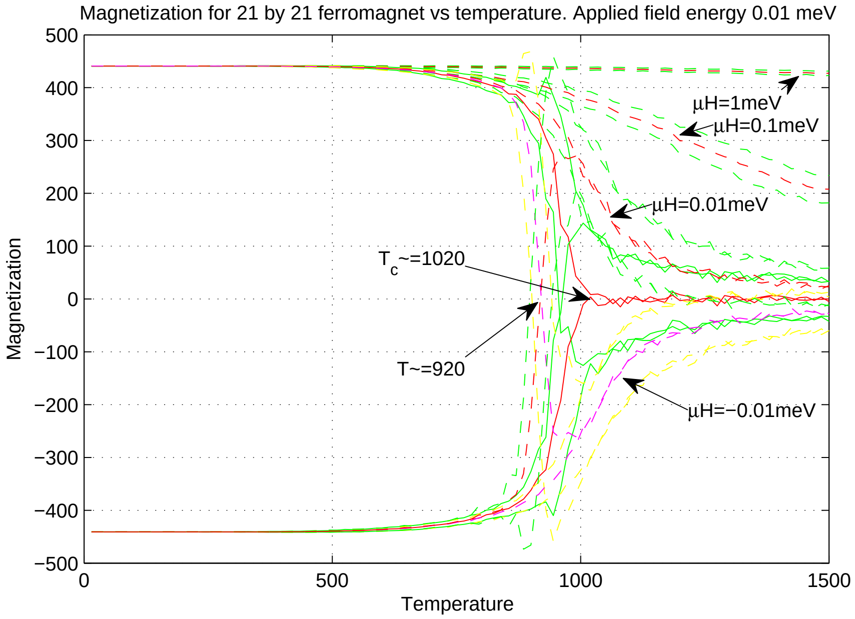
$$\text{denominator} = \text{numerator} + 1/\text{numerator}$$

and we have probability

$$p = \text{numerator}/\text{denominator}$$

As in the sublimation simulation we loop over a range of temperatures, iterating each temperature enough to gather averages and standard deviations. Before we take data at any temperature we must find the equilibrium for that temperature, which costs additional iterations in the innermost loop of the simulation.

The resulting magnetization is the number of spins up less the number of spins down.



The solid lines in this figure show magnetization vs. temperature with no external field. The sublimation starts with either all spins up or all spins down, hence the two branches. These branches are quite abruptly pulled down/up to zero magnetization at the transition (Curie) temperature: we can almost feel in these curves the "entropic force" countering the spontaneous magnetization.

(Results from the 21-by-21 lattice are shown. A 41-by-41 lattice has indistinguishable average results—the red curves—but the standard deviations have about half the spread—the green curves.)

The dashed lines show what happens when an external magnetic field is applied: red (average magnetization) and green (± 1 standard deviation) if the field is "up"; magenta (average) and yellow (± 1 standard deviation) if the field is "down".

The usual situation is given by an applied μH of 0.01meV: the magnetization starts, at 0^oK, at saturation in whichever direction was specified by the initial conditions, then drops/rises to a curve which appears to depend on the temperature inversely as $\pm(T - T_c)$ with the $\pm$ depending on the direction of the applied field.

Applied fields ten and a hundred times larger than this are clearly slower to respond to increasing temperature.

35. Particle individuality and Bose-Einstein condensation. Quantum particles have no individuality (to answer the final question of Excursion *Spin 3/2* in Week 6). This is why a linear combination of two or more particles is just as good as any of the particles by itself.

This is significant for multiple particles occupying a given set of particle states. For example, two *individual* particles, each allowed four states, could jointly have any one of the following sixteen states.

		state for particle 1			
		0	1	2	3
	0	0	1	2	3
state for	1	4	5	6	7
particle 2	2	8	9	10	11
	3	12	13	14	15

But this distinguishes between, say, state 6 (in which particle 1 is in state 2 and particle 2 is in state 1) and state 9 (in which particle 1 is in state 1 and particle 2 is in state 2).

For nonindividual particles, these two situations are not different. So such particles, which include all sufficiently small physical objects, can really jointly have only the following ten states.

		state for particle 1			
		0	1	2	3
	0	0	1	3	6
state for	1		2	4	7
particle 2	2			5	8
	3				9

where we've arbitrarily used the upper triangle of the matrix.

Note that the joint states are enumerated as nested triangles. Addressing for these elements of the matrix (i.e., the mappings from the states $j = 0,..,3$ for particle 1 and $k = 0,..,3$ for particle 2 to the joint state $a = 0,..,9$) is the topic of Excursion *Simplex arrays* of Week iv: $a = k + \Delta_j$ where Δ_j is the jth triangular number ($\Delta_0 = 0$).

For p particles this construct becomes p-dimensional, e.g., $p = 3$ and, again, $s = 4$ states

states for particle 3		state for particle 1			
	state for particle 2	0	1	2	3
0	0	0	1	4	10
	1		2	5	11
	2			7	13
	3				16
1	0				
	1		3	6	12
	2			8	14
	3				17
2	0				
	1				
	2			9	15
	3				18
3	0				
	1				
	2				
	3				19

in which $a = \ell + \Delta_k + \diamondplus_j$ for the indices (j, k, ℓ) for the states of particles (1,2,3).
In the general case of p particles and s states, we will need the reverse calculation: given a find $j, k, \ell, \ldots$. The recursive function

```
indices = simploc2indx(p,Sp,a,indices)
```

will do this, adding one index per invocation: `indices` is initially an empty vector, a is the input address as above, p is the number of dimensions (particles) and `Sp` is enough p-dimensional simplex numbers to reach the address a (i.e., triangular numbers for $p = 2$, tetrahedral numbers for $p = 3$, etc.; e.g., if $s = 4$, $S_1 = 1, 2, 3$; $S_2 = 1, 3, 6; S_3 = 1, 4, 10$; etc.) After s invocations of `simploc2indx()` ($s - 1$ of these recursive) `indices` will have s state numbers identifying uniquely the location addressed by a. For example

```
simploc2indx(3,[1,4,10,20,35,56],43,zeros(1,0))
```

gives

```
5   3   2
```

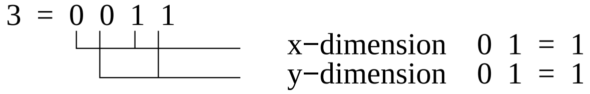
as the indices for address 43.

The states represented above either as single indices, say 14, or as a set of indices, say (equivalently) (3,2,1) for three particles, would be quantum states. For example, in one physical dimension, the (3,2,1) could represent the combination of a particle with wavenumber 4, a particle with wavenumber 3 and a third particle with wavenumber 2. (I've added a 1 to each index in (3,2,1) because the smallest wavenumber must be 1 not 0 so the state 0 with 3-particle triple (0,0,0) must represent the three wavenumbers (1,1,1).)

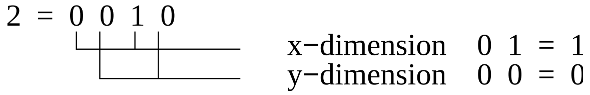
The energy of this state 14, or 3-particle indices (3,2,1), or 3-particle wavenumber (4,3,2), would be $\epsilon_0(4^2 + 3^2 + 2^2)$ where we square the wavenumbers and multiply by some constant ϵ_0 to get the energy of each particle in state 14.

Because systems of one physical dimension do not (usually) allow phase transitions (see the Excursions), we must be prepared to go to higher physical dimensions. Let's take the 2-particle state 8, or 2-particle indices (3,2), and think about two physical dimensions. Each of these indices now labels a particle in a 2-dimensional box and so having two wavenumbers, ν_x and ν_y.

In Week iii the Excursion *Z-order* tells us an easy way of representing two numbers by one: interleave the bits. Or, to go from one number to two, do this backwards. Here's how we turn 3 into two wavenumbers.



And 2.



So 3 is the 2-dimensional state with wave numbers (2,2) (remember: (1,1) plus 1) and 2 is the 2-dimensional state with wave numbers (2,1). The energy of state 8, or 2-particle indices (3,2), or 2-D 2-particle wavenumbers (2,2,2,1) is $\epsilon_0(2^2 + 2^2 + 2^2 + 1^2)$.

This can be extended to any number of physical dimensions, although we'll stop at 2. Thus we can model any number of different states, s, for each of p particles moving in d dimensions, using d-dimensional Z-order to unpack each of p indices which can be found from an overall single state index using up to s and p p-dimensional simplex numbers S_s^p.

(One caveat on Z-order: it will be best if s is a power of 2 or else the Peano curve—see Excursion *Z-order* of Week iii—will fall partly outside the region of allowed states and some of the Z-order results will have to be filtered out as irrelevant.)

Although we are focussing on bosons in this Note, we can easily include fermions in the picture. The difference between the two was stated in Week 5, Notes 8 and 9, in terms of whether the amplitudes must be added or subtracted, before squaring the results to get probabilities, on collision of two particles.

The consequence of this is that *two fermions cannot occupy the same state* while two bosons can.

It is easy to use simplex arrays to generate wavenumbers for fermions. For the example of three particles, just add (3,2,1) to the generated 3-particle indices. For instance, indices (0,0,0), which in one physical dimension give wavenumbers (1,1,1) for bosons, will give wavenumbers (3,2,1) for fermions. All three fermions will be in different states. And similarly for any other of the generated 3-particle indices.

Energies are calculated as we did for bosons, and Z-order is applied in the same way if the particle moves in two or more physical dimensions.

One more thing about energy: in addition to the energies calculated above we subtract a term μp where p is the number of particles (usually called N in the literature) and μ is the chemical potential for those particles (not to be confused with the magnetic μ of the previous Note). For fermions, μ is also called the Fermi energy and ϵ_0 may be bigger or smaller than μ. For bosons, μ must be smaller, often much smaller, than ϵ_0.

The probability of being in a combined state, say 3-particle wavenumbers (3,2,1) for particles moving in one dimension, thus has numerator

$$\exp \frac{3\mu - \epsilon_0(3^2 + 2^2 + 1^2)}{k_B T}$$

and denominator $Z =$ the sum of these numerators over all possible combined states. This could be calculated by a MATLAB function

```
indxprbs = probsFBpartDims(fb,dimens,numParts,T,mu,e0)
```

as in the example

```
indxprbs = probsFBpartDims('boson',3,3,100,100,300)
```

Here is the resulting `indxprbs`, giving the probabilities for each triple (`numParts` = 3 particles) of wavenumbers:

state	state	state	prob
1	1	1	0.2662
2	1	1	0.1880
2	2	1	0.1327
2	2	2	0.0937
3	1	1	0.1052
3	2	1	0.0743
3	2	2	0.0525
3	3	1	0.0416
3	3	2	0.0294
3	3	3	0.0164

Here the inputs are largely self-evident: the values for μ and ϵ_0 (100 and 300, respectively) are in meV and the actual numbers are less important than the ratio; and the `dimens` parameter, 3, says there are three one-dimensional states. (By contrast, 3 × 3 two-dimensional states, say, would be indicated by the vector [3,3].)

The output is the set of all 3-particle wavenumbers, each together with its calculated probability. So 3-wavenumbers (1,1,1) have probability 0.2662 in this calculation and so on. I've labelled the columns "state" three times—for particle 1, particle 2 and particle 3—and "prob".

Our task now is to move from combined states of all particles to single states for any one particle and to calculate the *expected occupancy* for each state, i.e., how many particles are occupying that state. This number should be 1 or less for fermions but is not constrained for bosons.

To find the expected occupancies for each state we must rearrange and aggregate the `indxprbs` calculated above.

Here is the process in three steps, illustrated for the three 3-state particle data above. First, we expand each row into a row for each different state in that row and record the multiplicity of that state, i.e., the number of times it appears in the row. We focus for clarity in the following on state 1 of any of the above particles, but must of course extend the thinking to all particles for the full calculations.

state	state	state	prob		state	mult	prob
1	1	1	0.2662	→	1	3	0.2662
2	1	1	0.1880	→	1	2	0.1880
					2	1	0.1880
2	2	1	0.1327	→	1	1	0.1327
					2	2	0.1327
⋮							
3	1	1	0.1052	→	1	2	0.1052
					3	1	0.1052
3	2	1	0.0743	→	1	1	0.0743
					2	1	0.0743
					3	1	0.0743
3	3	1	0.0416	→	1	1	0.0416
					3	2	0.0416

The second step is to aggregate: grouping by state and multiplicity, sum the probabilities.

state	mult	prob
1	3	0.2662
1	2	0.2932 = 0.1880 + 0.1052
1	1	0.2486 = 0.1327 + 0.0743 + 0.0416

And the third step is to multiply `mult`×`prob` and sum to get the expected occupancy for each state.

state	ex.occ.
1	1.6336

Similarly from the data in `indxprbs` above we get the expected occupancy for states 2 and 3 so the final result is

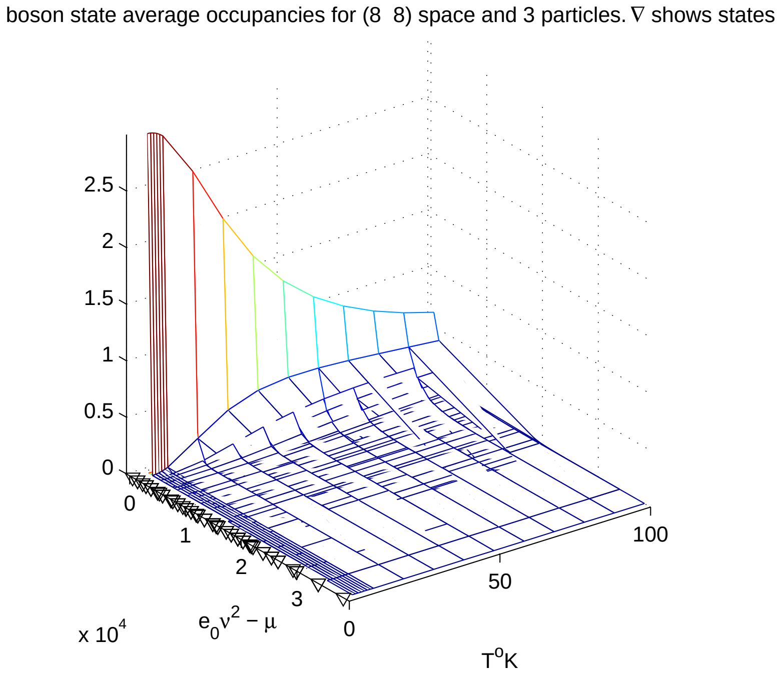
state	ex.occ.
1	1.6336
2	0.9433
3	0.4232

Note that these expected occupancies sum to 3, the total number of particles, as they should.

The MATLAB functions for these three steps could be
1) `statesMult = findMultiplicDims(statesRel,numStates,numDims)`
(taking the above example to `numDims`, which, if more than one dimension, requires each state to be a vector of that many dimensions). This could use a subroutine which expands each row of `indxprbs` before taking the union of all into the final result.
2) `occ = equivPlus(statesMult,numDims+1)`
which groups the rows of `statesMult` by equal values of the first `numDims+1` columns—do this by sorting—and sums the numbers in the `numDims + 2`nd column within each of these groups. (This is a special case of "equivalence reduction" operation on relations, hence its name.)
3) `equivOcc = equivHorzWeight(occ,numDims,numDims+1,numDims+2)`
which multiplies the `numDims + 1`st and `numDims + 2`nd columns of `occ` then sums the results within groupings which have all the same value of the first `numDims` columns. (This is also derived from more general relational operations.)

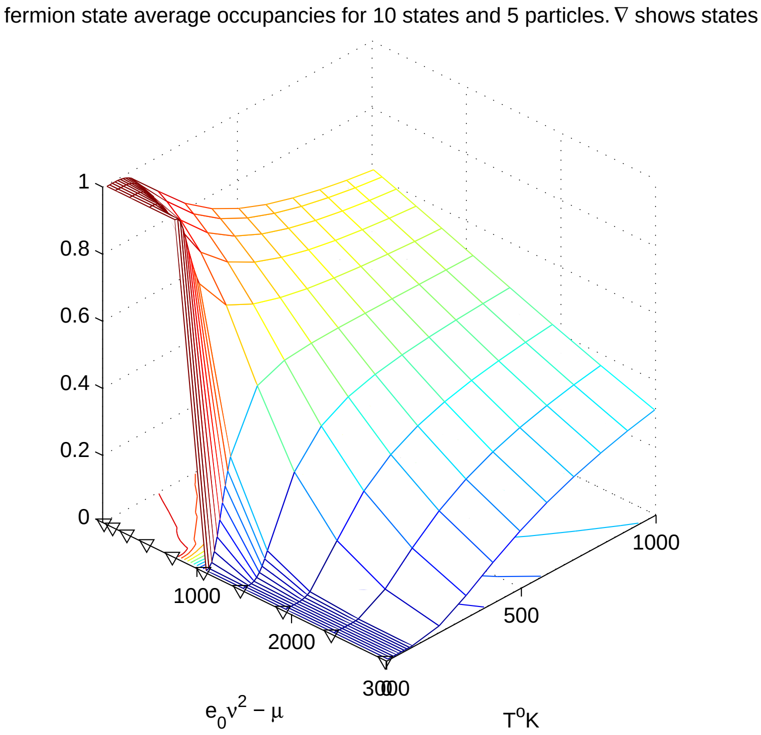
Putting all this together, here is a plot for 3 bosons in a 2-dimensional box of 8×8 states, with $\mu = 100$meV and $\epsilon_0 = 300$meV as before, over a range of temperatures from 1^oK to 100^oK paying close attention to the lower part of the range. It was produced by the invocation

```
        occupancies = occsFBpartDims('boson',[8,8],3,[1:9,10:10:100],100,300)
```



Since this code is also capable of finding fermion occupancies, here are 5 fermions in a 1-dimensional arrangement of 10 states with $\mu = 10$meV and $\epsilon_{)} = 30$meV (it's the ratio that counts) for a temperature range from 10 to 1000^{o}K, again focussing on the lower part of the range.

```
        occupancies = occsFBpartDims('fermion',10,5,[10:10:1000,200:100:1000],10,30)
```



In the fermion run, we can see the saturation of the lower-energy states at low temperature, with each state having an occupancy of 1 and no particle left over for higher energies.

In the boson run, we can see all the particles condensing into the lowest-energy state(s). There appears to be a definite phase transition. This is not seen in the 1-dimensional case, which just gives a spike at low T and low E.

36. Summary

(These notes show the trees. Try to see the forest!)

II. **The Excursions**

You've seen lots of ideas. Now *do* something with them!

1. Write the MATLAB functions leading up to the sequences of average distances in Note 30 and confirm the numbers there.

2. In Note 30 why do the nodes get closer together as we add more edges?

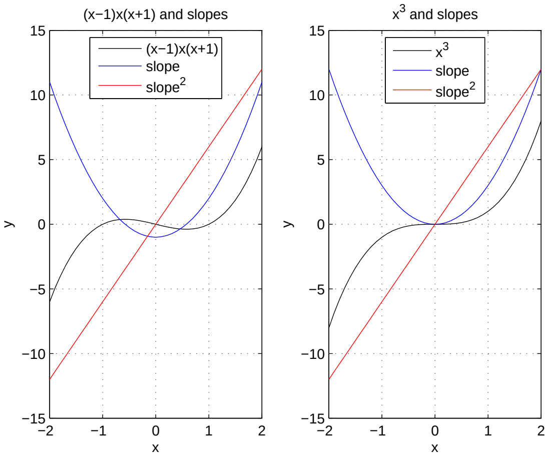
3. The random graphs of Note 30 were explored mathematically by Paul Erdös and Alfréd Rényi in 1960 [ER60]. They discuss five different phases in random graphs of n nodes and $N(n)$ edges (N is a function of n). Their results are hard to describe in imprecise language, and are derived in terms of probabilities approaching 1 as n approaches infinity ("almost surely"). One observation is that if N is about $n/2$ a "giant" component appears with all other components relatively small, which "melt" (larger ones first) into the giant.
   Another is that if N is about $(1/2)n\log\, n$ we almost surely get a connected graph. Look up their paper and use it to improve the simulations of Note 30.

4. Critical exponents, first mentioned in Note 30, crop up explicitly in the next two Notes and implicitly in the subsequent Notes in this Part. They are extensively discussed by Robertson [Rob93] (§6.4, defined p.288) and by Schwable [Sch06] (pp.336 and 259 give tables of possibilities).

5. Build `addEdge2connComp()` for resistance networks (Note 31) and use it to build `resistancesProbDist()` and then `resistancesProbLoop()` to simulate partial networks with various probabilities of forming edges.

6. **Percolation.** Reverting to resistances between two *given* points in a network, use code from the previous Excursion to simulate *percolation* of an electric current through a partial lattice from, say, a point on top to a point on the bottom of the network: as edges are added (or as the probability of an edge being present increases) a situation is reached in which the two points become connected and the current is no longer zero. See [Sch06, §7.5]

7. **Point of inflection.** The figure shows two cubic curves (in black). The



second has a "point of inflection" where both slope$_x y$ and slope$_x$slope$_x y$ equal zero. The first does not, because the two slopes equal zero at different values of x. Calculate the slope and slope^2 for $(x-1)x(x+1)$ and for x^3 and show that the second has, but the first has not, a point of inflection by this definition. Examine the black curves to see what a point of inflection looks like. What is the significance of slope$^2 = 0$?

8. For $T = T_c$ find the slopes slope$_v P$ and slope$^2_v P$ where

$$P = \frac{k_b T}{3v-1} - \frac{3}{v^2}$$

(Note 32) and show that the critical point (T_c, v_c, P_c) is

$$\begin{aligned} v_c &= 3b \\ k_B T_c &= \frac{8}{27}\frac{a}{b} \\ P_c &= \frac{a}{27b^2}, \end{aligned}$$

that the dimensionless ratio is 8/3 as given in Note 32, and that the dimensionless equation is that given in Note 32. (Or see [Sch06, p.246]).

9. Why would the Gibbs free energy (Note 20, Part II) be the most useful thermodynamic function to describe phase transitions?

10. Look up Johannes Diderik van der Waals (1837–1923) and James Clerk Maxwell's (1831–1879) fix of the van der Waals isotherms (Note 32).

11. Write the programs in Note 32 a) to plot the dimensionless van der Waals' isotherms,
b) to calculate and plot the Maxwell construction, and c) to calculate and plot the isothermal compression near the critical point.

12. Zemansky [Zem57, pp.204–5] shows three-phase PVT diagrams which extend the theory of Note 32, as well as experimental PVT diagrams for water and for helium. How can the additional aspects be explained?

13. Write the program `sublimLoops()` to calculate the sublimation curves of Note 33. It could be based on the function
`[numLeft,sqlatt0] = sublimEquilib(sqlatt,eps,T)`
for the square-symmetric lattice `sqlatt` (which becomes `sqlatt0`) at temperature T with the single n.n. energy threshold `eps`. (`numLeft` is the number of crystal lattice sites still occupied at equilibrium.)

14. In Note 34 the table of interaction energies vs. nearest-neighbour configurations shows the energy increasing by $2J$ for each additional neighbour whose spin turns down. Why 2?

15. The numbers I've used in Notes 33 and 34 are from [Kit66]: p.81 for $\epsilon = 10.4$ meV for argon (converted from ergs), and pp. 458, 461 for $J = 11.9$ meV and $T_c = 1043^o$K (the "Curie temperature" for iron), respectively. Check this against other sources.

16. Write the program to simulate the ferromagnetic phase transition in 2-D. Mine was
`[avgMag,stdMag,fail] = ferroMbyNequilVsTH(Tmprturs,muH,m,n,tol,ud)`
which gives averages and standard deviations of magnetization versus the set of temperatures in `Tmprturs`, for an m by n lattice initially magnetized `ud` = 1 (up) or 0 (down) with applied field `muH`. The parameter `tol` is a tolerance specifying equilibrium, which the program sometimes fails to reach, so `fail` reports the circumstances—in this case the attempt to find

equilibrium is pushed to 1000 times and the final result accepted anyway. Other parameters, such as J, are hard-wired into the code.

The program invokes

```
        [mag,noEquil] = ferroMbyNequilAtTH(T,muH,m,n,maxRepeats,tol,ud)
```

which finds equilibrium at a given temperature `T`, trying to achieve equilibrium in `maxRepeats = 10` attempts and otherwise reporting through `noEquil`, which becomes aggregated into `fail` in the upper program. It also invokes functions to calculate probabilities for that temperature and field strength, and to find the configuration of spins across the lattice.

Note that failure to reach equilibrium happens increasingly as temperatures rise through 1000^oK and almost always above that, but does not seem to spoil the results.

Compare the curves to experimental results.

Rewrite the simulation for a body-centred cubic (b.c.c.) lattice in three dimensions.

17. The simulation of the previous Excursion can be altered to start with zero magnetization and show the emergence of spontaneous magnetization at temperatures below the Curie point. A checkerboard, alternating up- with down-spins, is not the way to start, though: show that each iteration looking for equilibrium will just alternate the checkerboard—i.e., up sites become down sites and vice-versa in the next iteration. This will stop the equilibrium-seeking process artificially. Even with a randomized start, checkerboard regions can evolve, finding an artifical "equilibrium" prematurely. What can be done?

18. Show that there can be no phase transition in one dimension: think of, say, solidification, or spontaneous magnetization, as the onset of a long-range ordering or correlation among the molecules or atoms of the substance, across a body with effectively an infinite number of molecules of atoms; then think about the requirement to break this correlation in 1-D as opposed to 2-D or 3-D. [Wan87, p.4] but see [Rob93, p.327].

    The energy model for magnetism given in Note 34 was suggested by Wilhelm Lenz (1920) to his doctoral student Ernst Ising, who solved it exactly in one dimension (1925) and concluded from its failure to show a phase transition that the interaction term was wrong [Rob93, p.371]. It was subsequently solved exactly in two dimensions with great effort by Lars Onsager (1942): Wannier [Wan87, ch.16] streamlines this original solution and Robertson [Rob93, §§7.6–8] gives the modern analysis based on mathematical breakthroughs in combinatorics.

19. Study and compare the mathematical derivations of Bose-Einstein statistics and Fermi-Dirac statistics, say in [Wan87, §§9.2, 9.4] and [Rob93, §§4.2–5, 4.7]. Note the restrictions on μ introduced in Note 35.

20. Write the programs outlined in Note 35 to compute the expected occupancies of energy states for bosons and for fermions in a specified number of dimensions, and experiment with them.

21. **Infinities.** In the approximate language of slopes, the Ehrenfest classification of phase transitions calls a transition "nth-order" if at least one of the nth slopes of its energy dependence on temperature is discontinuous at the transition while all $n-1$st and lower slopes are continuous [Sch06, p.332]. Study Zemansky's [Zem57, §§15.1, 15.6] discussions of first-order and second-order transitions and classify the transitions in these Notes accordingly.

22. **Gibbs' paradox.** Why must the classical (pre-relativity, pre-quantum) partition function be divided by $p!$ where p is the number of particles? Willard Gibbs was forced to do this to get the entropy correct for ideal gases, although without the $p!$ the energy comes out right and so does the Ideal Gas Law [Rob93, §3.8.2].

    a) Suppose there are $p = 2$ particles, each with $s = 3$ possible states. Let the energies associated with the states for the first particle be E_1^1, E_2^1 and E_3^1, and those for the second particle be E_1^2, E_2^2 and E_3^2. Show that the combined partition function is

$$Z_{12} = \sum_{j,k} e^{-\beta(E_j^1+E_k^2)} = Z_1 Z_2$$

where

$$Z_1 = \sum_j e^{-\beta E^1_j}$$

and

$$Z_2 = \sum_k e^{-\beta E^2_k}$$

are the individual partition functions.
b) Since the above expressions do not depend on there being only 3 states, and since we can generalize to any number of particles, p, show that the p-particle partition function

$$Z = \prod_{k=1}^{p} Z_k$$

c) Rewrite this (back to $p = 2$ particles and $s = 3$ states) using

$$a_j = e^{-\beta E^1_j}$$

and

$$b_k = e^{-\beta E^2_k}$$

as

$$Z = (a_1 + a_2 + a_3)(b_1 + b_2 + b_3)$$

d) Now suppose the two particles are identical, $a_j = b_j$

$$Z = (a_1 + a_2 + a_3)^2$$

But particles have no individuality, as discussed in Note 35, so instead of

$$Z = a_1^2 + a_2^2 + a_3^2 + 2a_1a_2 + 2a_2a_3 + 2a_3a_1$$

we must write

$$Z = a_1^2 + a_2^2 + a_3^2 + a_1a_2 + a_2a_3 + a_3a_1$$

Generalize again to any number of particles, p, and to a much larger number of states, s—pre-quantum physics expects an infinite number of possible states, all in a continuum: it is overwhelmingly likely that all the particles will be in different states, so the combined partition function is dominated by terms such as

$$a_{j_1}a_{j_2}..a_{j_p}$$

These have coefficient $p!$ in the expansion of $\prod_{k=1}^{p} Z_k$ because there are $p!$ possible permutations of the same subscripts $j_1, j_2, .., j_p$. But lack of individuality requires us to suppress the factor $p!$, so, neglecting all the other terms in the product, we have essentially

$$Z = \frac{1}{p!}\prod_{k=1}^{p} Z_k$$

e) You can link this discussion to Excursion *Multinomial coefficients* in Week ii, as Robertson [Rob93, §3.3] does, but it is overkill.

23. **Ovations.** Is a standing ovation a phase transition?
a) Let's represent an auditorium as an array with seats numbered from top to bottom within rows receding to the right from the stage on the left. Here is a possible configuration in which 50% of the audience has spontaneously stood (1 for standing, 0 for still sitting).

											seat
s	0	1	1	1	0	1	0	0	0	0	5
t	1	0	1	1	1	1	1	1	0	1	4
a	0	1	0	0	1	1	0	1	0	1	3
g	1	0	0	0	0	1	0	1	0	0	2
e	1	1	0	1	0	1	0	0	1	0	1
row	1	2	3	4	5	6	7	8	9	10	

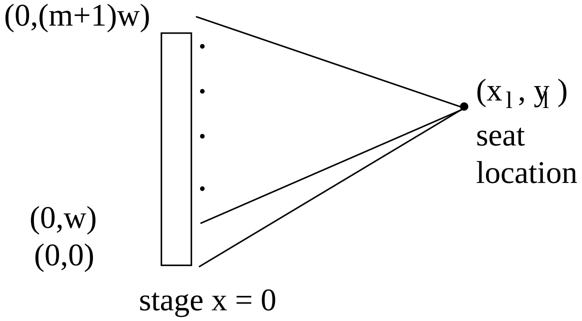
The function invocation might be

```
auditorium = spontaneous(5,10,.5)
```

b) Let's suppose, after the spontaneity, that more people will rise if they can't see the stage

```
auditorium(j,k) = cantsee(k,j,auditorium,1,1)
```

for any j, k for which $\texttt{auditorium}(j, k) == 0$. (The last two parameters do not turn out to be important: they are the separation, s, between rows and the width, w, of the seats.)
To find out if somebody sitting in seat j of row k cannot see the stage at all, we draw $m+2$ lines from the seat location $(x_\ell, y_\ell) = (ks, jw)$ to the stage at $x = 0$ and $y = 0, w, 2w, .., (m+1)w$, where m is the number of seats in each row (5 in the above example).



Show that to find a and b in the equation $y = ax + b$ for each line we must solve

$$\begin{pmatrix} y_\ell \\ jw \end{pmatrix} = \begin{pmatrix} x_\ell & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

For each of these lines of sight we work from the row in front of the seat location leftwards to the stage to see if anybody standing ahead of the location is blocking that line of sight, rounding off, where the line meets the row, to the nearest seat. If all lines of sight are blocked, report 1, forcing the person at (x_ℓ, y_ℓ) to stand.
c) We can experiment with a second mechanism to get people up or down: peer pressure. Within a given row of seats we can make someone stand if at least so many of heir neighbours are standing.

```
auditorium = neighbours(auditorium,m,ofn)
```

will do this if m out of n ("m of n") neighbours are standing.
The core routine within `neighbours()` is

```
yn = match(column,loc,m,ofn)
```

in which `column` is the column of the `auditorium` matrix telling who is standing or not in the row of seats (sorry: column of matrix for row of seats), `loc` is the location of the seat concerned in this row, and m and `ofn` are the above parameters. Let's see what happens for 1 of 2 neighbours in the column shown.

loc									yn
8	1							×	0
7	0						×	✓	1
6	0					✓	×		1
5	1				×	×			0
4	0			✓	✓				1
3	1		✓	×					1
2	1	×	✓						1
1	0	✓							1

For instance at `loc` = 2: the neighbour below (at 1) is not standing so there is an ×; but the neighbour above (at 3) is standing, so there is a $\surd$. Since one of these cases meets the 1 of 2 criterion, the result is `yn = 1`. Only when we get to `loc` = 5 are there no neighbours standing, with result `yn`= 0.
The extremes, `loc` = 1 and `loc` = 8, allow us to check only on one side: `loc` = 1 has the one neighbour up so `yn` = 1 but `loc` = 8 has the one neighbour down so `yn` = 0.
Show that the following is right for 2 of 3 neighbours in the same column.

loc								yn
8	1						×	0
7	0					×	×	0
6	0				×	×	×	0
5	1			×	×	×		0
4	0		✓	✓	×			1
3	1	×	×	×				0
2	1	×	×					0
1	0	✓						1

What I actually did with `neighbours()` was use it inverted

```
auditorium = ∼neighbours(∼auditorium,m,ofn)
```

This forced people to sit down again, after rising spontaneously, if at least 1 of the 2 neighbours was still sitting: don't embarrass your spouse!
d) A third mechanism

```
auditorium = allBehind(auditorium)
```

applies peer pressure from behind: if an entire row is standing then all rows in front of it stand too,
e) Packaging all the above into

```
iterCts = standOvEquilib(m,n,prob)
```

gives the number of people finally standing in an auditorium of n columns ("rows" of seats) of m seats each if the probability of initial spontaneous standing is `prob`. This routine

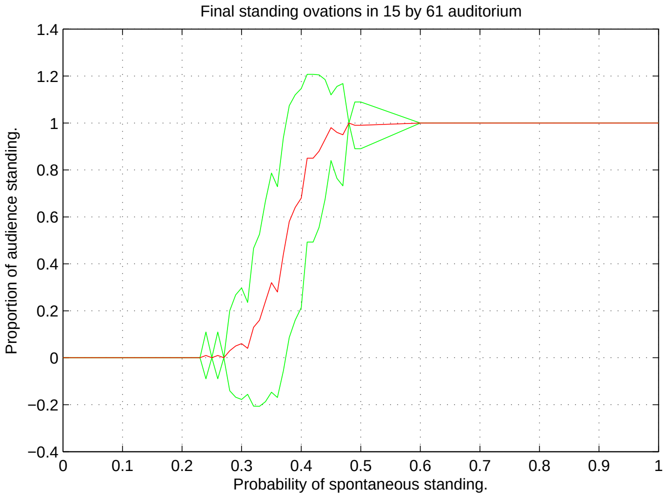
- finds spontaneous standing (`spontaneous()`)
- uses `neighbours(∼,1,2)` to prevent people standing by themselves
- iterates the following until equilibrium (numbers of people standing do not change) plus 5 iterations
  - uses `allBehind()` so the back of the auditorium influences the front
  - uses `cantsee()` so the front of the auditorium influences the back
  - uses `neighbours(∼,1,2)` so people do not stand alone.

It outputs the sequence of counts of standing people so that the convergence to equilibrium can be checked, but only the last of these numbers is needed for the next step.
f) Finally

```
                    [probs,avgProps] = standOvIter(m,n,probs,numIter)
```

iterates `standOvEquilib()` `numIter` times for each of a set of probabilities in `probs`, converting the number of standing people into a proportion of the full auditorium, and calculates averages and standard deviations of these proportions for plotting.
Here is the result of

```
    [probs,avgProps] = standOvIter(15,61,[0,0.1,0.2:0.01:0.5,0.6:0.1:1],100)
```



There seems to be a phase transition around probability 0.4. How do the shape and size of the auditorium influence the transition probability? Is the transition first order?

24. Are there phase transitions in human history and pre-history?
Did the Agricultural Revolution happen the first time somebody had the idea of planting out grain or did this happen many times before population growth forced the abandonment of the hitherto easier and less artificial methods of hunting and gathering?
Was World War I *caused* by the assassination in Bosnia of the heir to the Austro-Hungarian throne or was that just a "crystallizing seed" in a supersaturated situation of ideological and economic stress? [Som57, Tuc66]
Did Communism fall in 1989–91 *because* of political protest in Poland or exodus from East Germany, or did such triggers in these and other East European countries release economic and ideological pressures amplified by communication technologies? How did China in 1989 differ? [GK91]
Was the surprising onset of suicide bombing in Iraq under Western pressure predictable? [**?**, pp.1178,592,865]

25. **Power laws.** a) Write a MATLAB program which plots $x^{1.1}$ using `loglog()`. Why is it a straight line with slope 1.1, i.e., equal to the exponent?
b) Replot the results in Note 30 for Erdös-Rényi average distances and in Note 32 for van der Waals compressibilities using `loglog()`. Do these numbers really follow power laws?

26. **Self-Organized Criticality.** The *power law distibution* is discussed by Per Bak [Bak]. It appears to have a universality comparable to the Gaussian (normal) distribution, and it bears on phase transitions because it applies in the critical regime. Bak and colleagues have investigated the power law distribution in the context of phase transitions that do not depend on some external variable such as temperature or probability but on the internal organizaton of the collective system. So they refer to "self-organized" criticality as opposed to the criticality of a thermal phase transition.

For instance, the number of earthquakes is a power of their magnitude: Bak shows an example of this "Gutenberg-Richter" law based on 1974–83 data in the New Madrid zone of southeastern U.S. (Missouri, Illinois, Kentucky, Arkansas, Tennessee) for which the exponent appears to be about −0.86: this would correspond to about 2000 quakes, in the ten-year period, of magnitude 1 on the Richter scale (same C. F. Richter) (which has about the energy of a passing truck), about 275 quakes of magnitude 2 (which is ten times as energetic), about 38 quakes of magnitude 3 (100 times), 5 of magnitude 4 and so on.

Another, quite unrelated, example of a power law distribution is the number of 4-million-year periods which lost given percentages of species to extinction over the last 600 million years. So is the size of cities versus their rank, and the number of occurrences of words in a text versus word rank.

Bak's minimal model of such a system is a sandpile, which, as grains of sand are added to it, builds up until it reaches a critical state in which sand avalanches happen at all scales—and the number of avalanches of a given size is a power of the size.

Bak's simulation—the mathematics of even this simple case are as yet largely intractable—reduces the model to still greater simplicity. He imagines a two-dimensional grid to which sand grains are added to squares selected at random. If a square has a critical number of grains, say 4, or more, it topples by distributing its grains to its four nearest-neighbour squares. These in turn will accumulate until they topple in the same way. Grains toppling from squares on the edges fall off and are lost.

a) Write the program to simulate this and plot the numbers of the resulting avalanches versus their sizes using `loglog()`.

Bak [Bak, p.53] gives an example of the breadth-first algorithm he uses which is useful in designing and testing your code. A grain of sand is added to the middle square of a 5-by-5 grid, making it critical. The nine **4**s in the following seven steps are the locations that topple from one step to the next.

1	2	0	2	3
2	3	2	3	0
1	2	3	3	2
3	1	3	2	1
0	2	2	1	2

1	2	0	2	3
2	3	2	3	0
1	2	**4**	3	2
3	1	3	2	1
0	2	2	1	2

1	2	0	2	3
2	3	3	3	0
1	3	0	**4**	2
3	1	**4**	2	1
0	2	2	1	2

1	2	0	2	3
2	3	3	**4**	0
1	3	2	0	3
3	2	0	**4**	1
0	2	3	1	2

1	2	0	3	3
2	3	**4**	0	1
1	3	2	2	3
3	2	1	0	2
0	2	3	2	2

1	2	1	3	3
2	**4**	0	1	1
1	3	3	2	3
3	2	1	0	2
0	2	3	2	2

1	3	1	3	3
3	0	1	1	1
1	**4**	3	2	3
3	2	1	0	2
0	2	3	2	2

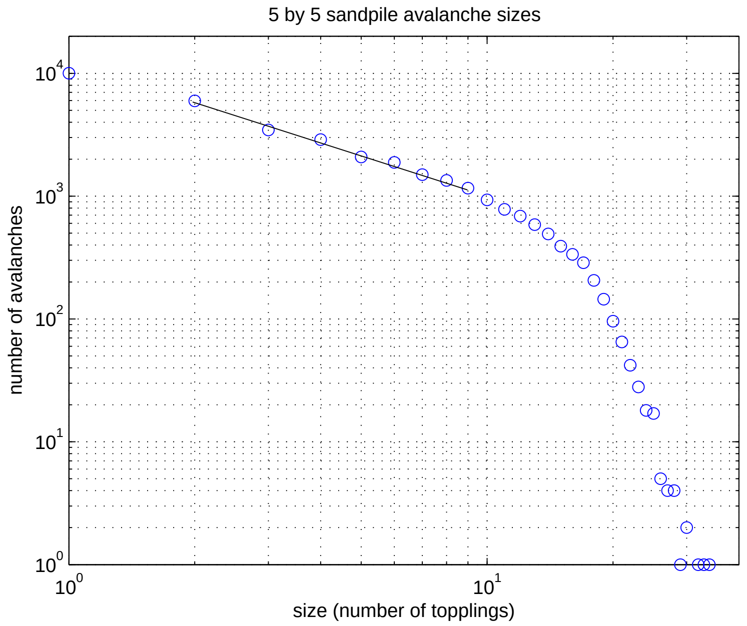
1	3	1	3	3
3	1	1	1	1
2	0	**4**	2	3
3	3	1	0	2
0	2	3	2	2

giving the final configuration

1	3	1	3	3
3	1	2	1	1
2	1	0	3	3
3	3	2	0	2
0	2	3	2	2

The size of this avalanche is 9, the number of topplings. Its *duration* is 7, the number of steps taken: Bak says that durations also follow a power law distribution.

Here is the result of 100,000 iterations on a 5-by-5 grid, starting empty, with one grain of sand falling on a randomly-placed square each iteration.



The number of sand avalanches is clearly decreasing exponentially with their size until we get to sizes that the simulation has not generated enough of to give good statistics. The slope there measures to be about -1.09. Note that the largest avalanche has 35 topplings, more than the squares in the grid.

Bak describes the numerous experiments that have been done and the simulations written to explain those experiments on sandpiles, as well as earthquakes and evolutionary "punctuated equilibrium" with its occasional mass extinctions.

How does this power law distribution relate to the critical exponents of second-order phase transitions?

b) Add a second plot to your sandpile program to track the total number of grains in the pile at each iteration. How "efficient" is the sandpile, in the sense of what proportion of its maximum capacity does it typically reach?
c) Start your program with a completely full sandpile, with 3 grains at each location: 100% "efficient". How big is the avalanche on the first addition? What does this tell you about, say, planned economies or roads used at 100% efficiency?

27. The Ising model (Note 34, and see Ising in a previous Excursion), although it is unsolved in three dimensions and labourious in two, gives the best entrée to seond-order phase transitions—those with critical points. The difficulty with an exact numerical calculation—and probably with the attempt to get mathematical results—is that the fluctuations, which appear once the temperature is raised from the completely ordered state of 0^oK or lowered from the completely random state of ∞^oK, appear increasingly at all possible scales of length as the temperature approaches critical. That is, there are islets of spin-up (or down) in puddles of spin-down (or up) in islands of spin-up (or down) in lakes of spin-down (or up), and so on, at all possible sizes, from single spins to spanning the whole crystal. This is the source of the power law distributions leading to the critical exponents, and also why the simulations of Note 34 and of the rest of this Part will never succeed in matching the experimental measurements.
A way to approximate the calculations while still taking into account the full range of scales at the critical point is by using "renormalization groups". The approach divides the lattice into small blocks of spins, for each of which it calculates an average spin. After adjusting the

coupling factor, this reduces the original lattice to a coarser one of the blocks. This operation is repeated until all scales are tractably accounted for. Onsager's 2D solution can be used to check the results, which are pretty good.

A remarkable hypothesis, "critical-point universality", arises from this and says that the critical exponents don't depend on the structure of the crystal or the microscopic physics at all but only on the dimension, which governs the results in two guises: the dimension of the space and the dimension of the values responsible for the interactions. For example, in the 2D Ising model the space has dimension 2 but the spins have dimension 1 because there is only one direction they can align in. (These numbers are 3 and 1 for the 3D Ising model.) The same numbers apply to the density difference between phases of a 2D (or 3D) fluid near the critical point (Note 32), so the hypothesis says the critical exponents will be the same—and we can use the Ising model to study it.

Kenneth Geddes Wilson [Wil79] provides a good introduction, from which you can approach the other references in this Part and attempt your own programs.

28. Any part of the Preliminary Notes that needs working through.

References

[Bak] Per Bak. *How Nature Works: The Science of Self-Organized Criticality.* Springer-Verlag, New York.

[ER60] P. Erdös and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5:17–61, 1960.

[GK91] Bernard Gwertzman and Michael T Kaufman, editors. *The Collapse of Communism.* Random House Inc (Times Books), New York, 1991. Written by the Correspondents of The New York Times.

[Kit66] Charles Kittel. *Introduction to Solid State Physics.* John Wiley & Sons Inc., New York, 1966. 3rd ed.

[Rob93] Harry S Robertson. *Statistical Thermophysics.* P T R Prentice-Hall, Inc., Engelwood Cliffs, NJ, 1993.

[Sch06] Franz Schwabl. *Statistical Mechanics.* Springer-Verlag, Berlin, Heidelberg, New York, 2006. Translation by William Brewer of "Statistiche Mechanik" 2006.

[Som57] D C Somervell. *A Study of History.* Oxford University Press, Oxford, 1946,1957. A J Toynbee abridged from 12 to 2 volumes.

[Tuc66] Barbara W Tuchman. *The Proud Tower: A Portrait of the World Before the War 1890-1914.* Ballantine Books, 1966.

[Wan87] Gregory H Wannier. *Statistical Physics.* Dover Publications, Inc., Mineola, N.Y., 1987. Originally New York 1966 John Wiley and Sons, Inc.

[Wil79] Kenneth G Wilson. Problems in physics with many scales of length. *Scientific American*, 241(2):158–79, Aug. 1979.

[Zem57] Mark W Zemansky. *Heat and Thermodynamics: An Intermediate Textbook for Students of Physics, Chemistry, and Engineering.* McGraw-Hill Book Company, Inc., New York, 1957.