

## Physics 234: Lab 9

Tuesday, March 22, 2011 / Thursday, March 24, 2011

1. Use the `curl` command to download from the class website everything you'll need for the lab.

```
$ WEBPATH=http://www.ualberta.ca/~kbeach/phys234
$ curl $WEBPATH/docs/Lab9.pdf -O
$ curl $WEBPATH/src/Lab9.tar.gz -O
$ evince Lab9.pdf&
$ tar xzf Lab9.tar.gz
$ cd Lab9
```

2. Consider a system of rotors  $\mathbf{n}_i = (\cos \theta_i, \sin \theta_i)$  arranged on a regular lattice. Let us assume that the pairwise interactions  $J_{ij} \mathbf{n}_i \cdot \mathbf{n}_j$  vanish unless the sites  $i$  and  $j$  are neighbouring and that the nonvanishing terms have constant values  $J_{ij} = J$ . The total energy is thus

$$E = J \sum_{\langle i,j \rangle} \cos(\theta_i - \theta_j),$$

where  $\langle i, j \rangle$  denotes a sum over all nearest neighbour pairs (no double counting).

In order to find the lowest energy configuration, we want to start the system in some initial angular configuration  $\{\theta_i^{(0)}\}$  and allow the rotors to evolve according to steepest-descent dynamics: i.e.,

$$\dot{\theta}_i \sim -J \sum_{j \in \mathcal{N}_i} \sin(\theta_i - \theta_j),$$

where  $\mathcal{N}_i = \{\text{all sites } j > i \text{ that are adjacent to } i\}$ . As we showed in class, this means that each rotor moves in the locally “downhill” direction of  $-\partial E / \partial \theta_i$ .

In the Lab8 directory, you'll find a program `rotor` that simulates a triangular network on 90 sites (with periodic boundary conditions), evolving according to the steepest-descent dynamics described above. Make and run the program as follows.

```
$ make rotor
g++ -c rotor.cpp -O2 -ansi -pedantic -Wall
g++ -c rotor_openGL.cpp -O2 -ansi -pedantic -Wall
g++ -o rotor rotor.o rotor_openGL.o -L/usr/X11R6/lib -lglut -lGLU -lGL -lXmu ...
$ ./rotor
```

A graphical display window will open. You should observe the rotors—randomly directed at first—begin to relax into perfect alignment. This is the ferromagnetic state that corresponds to  $J < 0$ .

Try playing around a bit. What happens if you flip the sign of  $J$ ? Is the lowest-energy configuration different if some of the sites are removed to create a honeycomb lattice? The program will respond to the following key strokes:

key	action
r	randomize rotors
a	align rotors
v	arrange spins in a vortex pattern (for you to implement)
l	toggle between triangular and honeycomb lattices
j	toggle between $J = 1$ and $J = -1$
q	quit

A global rotation of the spins can also be applied by holding down the mouse button and dragging the mouse right or left.

- The program is organized into three files: `rotor.cpp`, `rotor.h`, and `rotor_openGL.cpp`. The first contains the code that implements the lattice structure and steepest-descent updates. The second is a header that contains prototypes for the functions defined in `rotor.cpp`. The third contains all the visual code (using the Open Graphics Language).

In the file `rotor.cpp` file, complete the function `double energy(void)` so that it computes the total interaction energy between all rotors labelled `active`. You should be able to reproduce the energies given in the table below. What description of each rotor configuration would you write in the final column?

<i>lattice</i>	<i>interaction</i>	<i>lowest energy</i>	<i>rotor configuration</i>
triangular	$J > 0$	-135	
triangular	$J < 0$	-270	
honeycomb	$J > 0$	-90	
honeycomb	$J < 0$	-90	

- Run the `rotor` program and wait until the rotors are well-converged to a ferromagnetic configuration ( $J < 0$ ). Press 'j' to reverse the sign of  $J$  and watch the spins reorient themselves. Now, once again, start from a ferromagnetic configuration ( $J < 0$ ) but this time align all the spins by pressing the 'a' key. If you then press 'j' to reverse the sign of  $J$ , why do the spins not relax away from perfect alignment? (Hint: it has to do with metastability and floating-point numbers!)
- Set up the system on a triangular lattice with antiferromagnetic couplings and allow it to relax. Note that the rotors are *not* collinear. Using your mouse, rotate the arrows to lie along the connecting lines of the lattice. You should see that the arrows around each triangular plaquet circulate either clockwise or counterclockwise. The sense of circulation alternates between neighbouring plaquets. Here's another interpretation: rotate the arrows so they all point toward the centre of a triangular plaquet. Now the pattern is one of alternating plaquets of opposite flux (in or out).

What's so different about the triangular and honeycomb lattices to justify their very different antiferromagnetic ground states?

- Complete the function `rotor_vortex` in `rotor.cpp` so that it orients each rotor in a vortex pattern  $\mathbf{n}_i = (x_i, y_i) / \sqrt{x_i^2 + y_i^2}$ . This is equivalent to setting the angles to  $\tan \theta_i = n_{2,i} / n_{1,i}$ . Use the two-argument function `atan2` found in the `cmath` library and the functions `x_pos` and `y_pos` to convert the each lattice label to its spatial coordinates.

Set up a vortex configuration on the honeycomb lattice with ferromagnetic couplings (i.e., the spins are trying to align). You may find that the system won't relax to its ground state. What energy does the system converge to (instead of -90)? What does this say about the shape of the energy landscape?

- (Assignment 5:** to be submitted before the next lab period.) The shape of a string suspended from two fixed points is called a *catenary* and has the well-known form

$$y(x) = a[\cosh((x - x_0)/a) - 1] + y_0.$$

The parameters  $x_0$  and  $y_0$  are translations, and  $a = \rho g$  is a product of the line tension of the string and the local gravitational acceleration. We're going to use measurements to fit these three unknowns (of which only  $a$  is physically meaningful).

To start, compile and run the `catenary` program:

```

$ make catenary
g++ -c catenary.cpp -O2 -ansi -pedantic -Wall
g++ -c catenary_openGL.cpp -O2 -ansi -pedantic -Wall
g++ -o catenary catenary.o catenary_openGL.o -framework OpenGL -framework GLUT -lobjc
$ ./catenary

```

Maximize the window so that it covers most of your screen. Then, hang a piece of string from the two sides of your monitor. Using your mouse, carefully trace the path of the string using 20 to 50 clicks. Press ‘n’ to start a new set of measurements. Repeat this at least ten times. Press ‘q’ to quit the program. You should find that your measurements are stored in a file named `string.dat`:

```

$ more string.dat
$ gnuplot
> plot "string.dat"
> plot "string.dat" index 0
> plot "string.dat" index 1
> plot "string.dat" index 2
> quit

```

The non-negative quantity

$$\chi^2 = \sum_{i=1}^N (y - y_i)^2 = \sum_{i=1}^N (\Delta y_i)^2 = \sum_{i=1}^N \left[ a \cosh\left(\frac{x_i - x_0}{a}\right) - a + y_0 - y_i \right]^2$$

measures the deviation of the measured data from the predicted curve for a particular set of parameters  $(x_0, y_0, a)$ . The best possible fit corresponds to the smallest value of  $\chi^2$ . The derivatives of  $\chi^2$  along each of the parameter directions are summarized below.

$$\begin{aligned} \frac{\partial \chi^2}{\partial x_0} &= - \sum_{i=1}^N \Delta y_i \sinh\left(\frac{x_i - x_0}{a}\right) \\ \frac{\partial \chi^2}{\partial y_0} &= \sum_{i=1}^N \Delta y_i \\ \frac{\partial \chi^2}{\partial a} &= \sum_{i=1}^N \Delta y_i \left[ \cosh\left(\frac{x_i - x_0}{a}\right) - 1 - \left(\frac{x_i - x_0}{a}\right) \sinh\left(\frac{x_i - x_0}{a}\right) \right] \end{aligned}$$

We can think of the three parameters as constituting a vector  $\mathbf{v} = (v_1, v_2, v_3) \equiv (x_0, y_0, a)$  whose local downhill direction is given by the gradient  $-\partial\chi^2/\partial v_i$ .

The program `string_fit` is designed to read in your measurements from `string.dat` and fit a catenary curve to it. A bootstrapping technique is used in which the fit is performed repeatedly on a random sample of the data. The mean and variance of the parameters then provide a best fit value and a statistical uncertainty.

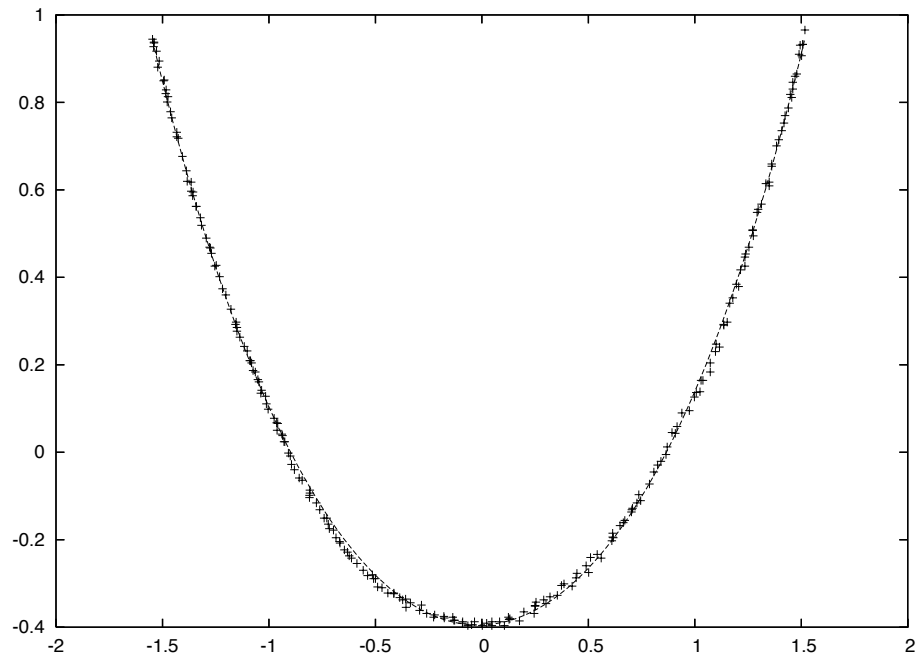
In the file `string_fit.cpp`, write routines `steepest_descent` and `statistics` that (i) compute the best-fit value of  $\mathbf{v}$  using steepest descent and (ii) compute the mean and variance of a list of bootstrap samples. A completed program will give you results something like the following and a nice graph.

```

$ ./string_fit
Processing...
iteration 1
iteration 2
iteration 3

```

```
.  
.  
iteration 95  
iteration 96  
iteration 97  
iteration 98  
iteration 99  
iteration 100  
x0 = -0.01583404978 +/- 0.00045  
y0 = -0.3950524019 +/- 0.0032  
a = 1.044980718 +/- 0.003  
$ gnuplot -persist view7.gp
```



Please submit these files for grading:

```
$ submit234 string_fit.cpp  
$ submit234 string.dat
```