MSDM5004 Spring 2021

Final Project

Due June 1

Requirements of the report: Choose one project from the five projects given below. Summarize the problem and the major formulations. Show the results (data and figures) with analysis/discussion. Attach the codes.

A. Motion by Curvature and Image Segmentation

Motion by curvature is an important class of motions of interfaces (motion by mean curvature for surfaces in three dimensions), and has a wide range of applications in science and engineering.

For a curve in two dimensions $\mathbf{r}(u) = (x(u), y(u))$, where u is a parameter of the curve, the velocity of the motion by curvature is

$$\frac{d\mathbf{r}}{dt} = \mathbf{v} = \kappa \mathbf{N}, \tag{1}$$

where κ is the curvature and $\mathbf{N}$ is the normal direction of the curve. A typical physical meaning of such a motion is to reduce the total energy (which is proportional to the length of the curve if the energy density is constant) in a steepest descent way (gradient flow).

For the curve $\mathbf{r}(u) = (x(u), y(u))$, its curvature κ can be calculated by

$$\kappa = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{\frac{3}{2}}}, \tag{2}$$

and the tangent direction and normal direction of the curve are

$$\mathbf{T} = \frac{1}{(x'^2 + y'^2)^{\frac{1}{2}}}(x', y'),$$

and

$$\mathbf{N} = \frac{1}{(x'^2 + y'^2)^{\frac{1}{2}}}(-y', x').$$

Here we have used the notions $g' = \frac{dg}{du}$ and $g'' = \frac{d^2g}{du^2}$. Note that during the evolution, $\mathbf{r} = \mathbf{r}(u, t)$.

Numerically, a simple treatment is to discretize the curve into a number of nodal points associated with a discrete set of values of the parameter u and are connected by straight segments, i.e., $\mathbf{r}_i = (x_i, y_i)$, for $i = 1, 2, \cdots$, where (x_i, y_i) is the numerical value of the exact location $(x(u_i), y(u_i))$ for some value of the parameter u_i for each i.

An explicit numerical scheme for the evolution equation (1) is

$$\mathbf{r}_i^{n+1} = \mathbf{r}_i^n + \Delta t \cdot \mathbf{v}_i^n, \tag{3}$$

where $\mathbf{r}_i^n$ and $\mathbf{v}_i^n$ are the location of velocity of the i-th nodal point at time $t_n = n\Delta t$ and Δt is the numerical time step.

The curvature κ and the normal direction $\mathbf{N}$ in the evolution equation (1) is calculated by using Eq. (2) and central difference schemes of the first and second derivatives:

$$x_i' = \frac{x_{i+1} - x_{i-1}}{u_{i+1} - u_{i-1}}, \quad x_i'' = \frac{(x_{i+1} - x_i)/(u_{i+1} - u_i) - (x_i - x_{i-1})/(u_i - u_{i-1})}{(u_{i+1} - u_{i-1})/2}, \tag{4}$$

and same for the derivatives of y.

In the active contour method for image segmentation, a moving curve is locked by the boundary of the object to be identified (*M. Kass, A. Witkin, D. Terzopoulos, Snakes: Active contour models, Int. J. Comput. Vision 1, 321–331, 1988*). A simplified version of the evolution equation in the active contour method is

$$\mathbf{v} = \alpha\kappa\mathbf{N} + \lambda\mathbf{f}_{\text{edge}}, \tag{5}$$

with

$$\mathbf{f}_{\text{edge}} = \big[\nabla\big(\|\nabla I(x,y)\|^2\big) \cdot \mathbf{N}\big]\mathbf{N}, \tag{6}$$

where $I(x, y)$ is the image intensity function, and α and λ are two positive parameters. This evolution equation is obtained by minimizing the total energy that consists of the length of the curve and the energy due to its interaction with the edge of the object in the image whose energy density is $e_{\text{edge}}(x, y) = -\|\nabla I(x, y)\|^2$ (with coefficients α and λ, respectively).

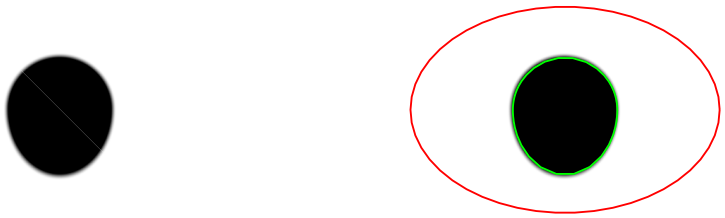
1. Consider an ellipse

$$\begin{cases} x = 0.5\cos\theta \\ y = 0.3\sin\theta \end{cases}$$

where θ is the parameter of the curve, $0 \le \theta \le 2\pi$. Evolve the curve by the curvature motion in (1) for some time t. Performing convergence tests by using different time steps, e.g., Δt and $\Delta t/2$, and compare the results quantitatively. Plot the results with some intermediate stages. (Remark: If there are oscillations in your results, they come from numerical instability and you need to choose a smaller time step.)

2. Segmentation of the image shown in Fig. 1 using the active contour method. We would like to find the edge of the object in Fig. 1(a) by a moving contour using Eq. (5), and a result is shown in Fig. 1(b). Use the ellipse in part 1 as the initial curve. Choose appropriate parameters α and λ in Eq. (5) in your simulations. A MATLAB function that can generates the image in Fig. 1(a) is enclosed at the end of this problem. In the function, the object is smoothed with parameter ep and choose $ep = 0.08$ in your simulations. The gradient in the force due to edge in Eq. (6) can be calculated by central differences: $\frac{\partial g}{\partial x}(x, y) = \frac{g(x+\Delta x,y)-g(x-\Delta x,y)}{2\Delta x}$ and $\frac{\partial g}{\partial y}(x, y) = \frac{g(x,y+\Delta y)-g(x,y-\Delta y)}{2\Delta y}$ with appropriate small

Δx and Δy. Note that you may need to use a smaller time step Δt when the curve is approaching the object boundary in order to avoid numerical instability. Plot your result as in Fig. 1(b) and with some intermediate configurations of the curve during the evolution.



(a) (b)

Figure 1: (a) The image. (b) Segmentation result using active contour method. The red curve is the initial contour and the green curve is the final, converged contour.

MATLAB function for the intesity of the image

```
function ii=intensity(x,y,ep)
a=0.15;
b=0.2;
if y>=0
    r=sqrt(x^2+y^2);
    if r<=a
        ii=0;
    elseif r>a+ep
            ii=1;
    else
            ii=sin((r-a)/ep*pi/2);
    end
else
    r=sqrt(x^2/a^2+y^2/b^2)*a;
    if r<=a
        ii=0;
    elseif r>a+ep
            ii=1;
    else
            ii=sin((r-a)/ep*pi/2);
    end
end;
```

B. Electric Potentials and Fields

In regions of space that do not contain any electric charges, the electric potential V, in a two-dimensional setting, obeys the Laplace's equation

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0.$$

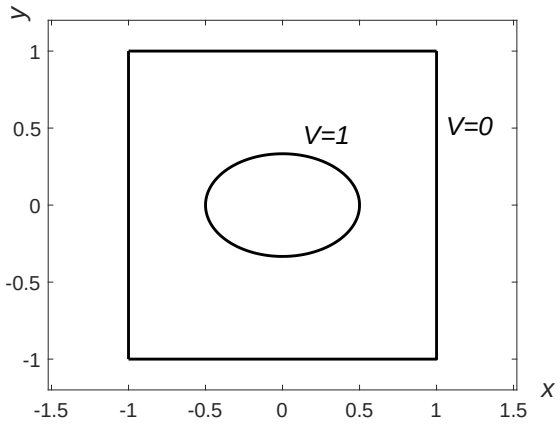


Figure 2: Schematic cross section of a hollow metallic prism with a solid, metallic inner conductor. The prism and inner conductor are presumed to be infinite in extent along z. The inner conductor is held at V = 1 and the walls of the prism at V = 0. The hollow metallic prism is the region $[-1, 1]^2$ excluding the inner conductor region $4x^2 + 9y^2 \leq 1$.

1. Consider the electric potential inside the prism in Fig. 2. It is an infinitely long, hollow prism, with metallic walls and a square cross-section. Inside this prism is a metal bar, also with a square cross-section. We assume that a voltage is applied between the bar and the outer walls, and we want to calculate the potential in the space between them. See the caption of Fig. 2 for details. Solve this problem numerically. Show convergence by using different numerical mesh sizes. When solving the linear system after discretization, use the Jacobi method and the Gauss-Seidel method and compare the results using these two methods. You are required to write the codes for the Jacobi and Gauss-Seidel iterations by yourself, and using available functions will not receive credits.

2. Calculate the electric field $\mathbf{E} = -\nabla V$. Plot the components of $\mathbf{E}$ and the vector field $\mathbf{E}$ (using, e.g., command "quiver" in MATLAB).

Remark. Finite difference approximations of partial derivatives at a point near the curved boundary can be derived using the Taylor expansions as follows.

Consider the Taylor expansion of the function u at the point P in Fig. 3. For the partial derivatives of u, we need to use the values at points A and B instead of those at points E and N. In order to approximate the partial derivatives with respect to x, we

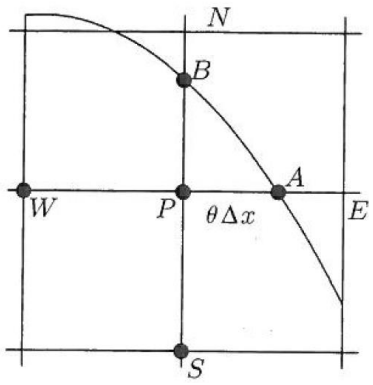


Figure 3: Points used in finite difference schemes for partial derivatives near a curved boundary.

write $PA = \theta\Delta x$. The Taylor expansions for the values of u at points A and W, i.e., u_A and u_W, can be written as

$$u_A = \left[u + \theta\Delta x\frac{\partial u}{\partial x} + \frac{1}{2}(\theta\Delta x)^2\frac{\partial^2 u}{\partial x^2} + \cdots\right]_P,$$
$$u_W = \left[u - \Delta x\frac{\partial u}{\partial x} + \frac{1}{2}(\Delta x)^2\frac{\partial^2 u}{\partial x^2} - \cdots\right]_P.$$

We can solve $\frac{\partial u}{\partial x}\Big|_P$ and $\frac{\partial^2 u}{\partial x^2}\Big|_P$ as

$$\left.\frac{\partial u}{\partial x}\right|_P \approx \frac{u_A - \theta^2 u_W - (1-\theta^2)u_P}{\theta(1+\theta)\Delta x},$$

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_P \approx \frac{u_A + \theta u_W - (1+\theta)u_P}{\frac{1}{2}\theta(1+\theta)(\Delta x)^2}.$$

Approximations of partial derivatives of u with respect to y at the point P can be obtained similiarly based on values u_B and u_S.

C. Image Compression and Denoising

In YCbCr format, computers store each pixel of a color image by a triple (Y, Cb, Cr), where Y corresponds to luminance or brightness, Cb corresponds to blue chrominance, and Cr corresponds to red chrominance. The intensity of each component is represented by a number in the range [0,255]. In this project we shall study Joint Photographic Expert Group (JPEG) algorithm, which has been the most widely used lossy image compression algorithm of the last two decades. The term lossy means that the original image is not recoverable after compression in contrast to lossless image compression in which the original image can be recovered. The JPEG compression algorithm compresses images of the YCbCr format by using the following broad steps on each of the three components of a YCbCr image:

(1) **Partition the image**

Each of the three components of the image is partitioned into 8×8 blocks of pixels **M**. The entries correspond to the intensities of each of the three components for a particular pixel and are in the range [0,255].

(2) **Apply Type II DCT**

In order to apply the DCT, 128 is subtracted from each entry of **M** so the entries are in the range [-128,127]. Each **M** is transformed using the two dimensional type II DCT into the coefficient matrix **D**.

(3) **Quantization**

Quantization is the step where the actual compression takes place. This process involves element by element division of the **D** matrices by a matrix known as the quantization matrix **Q** and then rounding the entries. The JPEG standard quantization matrix is

$$\mathbf{Q} = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \tag{C. 1}$$

This JPEG quantization matrix has been optimized through numerous experiments on human vision. The larger entries in the bottom right corner are a consequence of the fact that the human eye is poor at detecting large variations in color over short distances (i.e. high frequency changes), so when we divide **D** by these entries of **Q** and round off, most term of the high frequency will become zero. An example of the effect of quantization is given below:

$$\frac{\mathbf{D}}{\mathbf{Q}} = \frac{\begin{pmatrix} -154 & 58 & 2 & -15 & -1 & 8 & -1 & -6 \\ 1 & 17 & 23 & -4 & -11 & 3 & 5 & 0 \\ 29 & 8 & -4 & -8 & 3 & -6 & 0 & 5 \\ -11 & -10 & 8 & -5 & 2 & -1 & 2 & -6 \\ 2 & -7 & 0 & -2 & 0 & -3 & 0 & -3 \\ -3 & -11 & 2 & 0 & 6 & -2 & 0 & 2 \\ 3 & 0 & 5 & 0 & 2 & -2 & 2 & -3 \\ 2 & -2 & -1 & 0 & 0 & -1 & 0 & 2 \end{pmatrix}}{\begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}}$$

$$= \begin{pmatrix} -10 & 5 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{(C. 2)}$$

Note that the matrix division here is defined as element-wise division, but not multiplication by matrix inverse.

(4) **Compression**

The image can then be stored by entropy encoding, which records the number of zeros in a succession rather than recording each zero entry. We sill skip this step in this project. Instead, we will count the number of zero matrix elements before and after quantization to estimate the compression ratios.

In JPEG one can adjust the amount of compression by multiplying the quantization

matrix **Q** by a positive constant q. If $q < 1$, then the entries of **Q** will be smaller, and upon division by **Q** and rounding, more entries of **D** will remain non-zero. If $q > 1$, then the entries of **Q** will be larger, and upon division by **Q** and rounding, more entries of **D** will become zero. The human eye is more sensitive to luminance variations compared with chrominance variations. One may therefore use smaller q for the former and larger q for the latter. We shall denote the compression factors by $\vec{q} = (q_{\mathrm{Y}}, q_{\mathrm{Cb}}, q_{\mathrm{Cr}})$.

(5) **Decompression**

To decompress the stored image for viewing, first reconstruct the matrices by the stored numbers, then multiply the matrices by **Q** element-by-element, and finally apply inverse DCT to obtain the image.

In the above example, the matrix obtained after multiplication by **Q** (but before inverse DCT) is

$$\begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \begin{pmatrix} -10 & 5 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} -160 & 55 & 0 & -16 & 0 & 0 & 0 & 0 \\ 0 & 12 & 28 & 0 & 0 & 0 & 0 & 0 \\ 28 & 13 & 0 & 0 & 0 & 0 & 0 & 0 \\ -14 & -17 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \text{(C. 3)}$$

(a) Compress the given image file by JPEG with $\vec{q} = (1, 1, 1)$, $(1, 10, 10)$, and $(10, 20, 20)$. Show the compressed images. In each case, count the percentage of Fourier coefficients in each of the three channels that has been discarded (originally non-zero but becomes zero after quantization). Also calculate the average relative difference of each of the three channels in each case. The average relative difference is the absolute value of the ratio of the difference

divided by the original value, average over all entries. Comment on your findings.

(b) Redo part (a) but with DFT instead of DCT. Compare and comment on the performance of DFT to DCT.

(c) Explain why the algorithm is lossy. In which step(s) is information lost?

Image Denoising

Sometimes, the information at some pixels may be corrupted due to errors. One possible way to remove the artefacts due to such random errors is to transform the image to frequency domain and set all Fourier coefficients above a certain threshold frequency to zero. The cutoff frequency has to be chosen carefully so that most of the spurious artefacts can be removed while not losing too much of the true image.

(d) A corrupted image of the image in part (a) and (b) is generated by adding random errors to the pixels. Denoise the image using the method described above.

(e) Explain why the method works.

Note: Make sure that the array you obtain by reading the image is in YCbCr format. For example, in python, this can be ensured by using

```
from PIL import Image
img = numpy.array(Image.open("x.jpg").convert('YCbCr'))
```

To show or save image in YCbCr, you may use

```
Image.fromarray(img,'YCbCr').show()
```

or

```
Image.fromarray(img,'YCbCr').save()
```

D. Neural Network

The human brain consists of an extremely large number ($\sim 10^{12}$) of basic units called neurons, each of which is connected to many other neurons in a relatively simple manner. A biologically complete discussion of neurons and how they function is a long story. In this project we will consider how a neural network can function as a memory. Many aspects of biological memories are not understood. For example, it is not yet known how information is stored (and forgotten) or how it is recalled. However, some ideas from physics have contributed greatly to recent progress in this area, and the answers to these questions may not be far away.

It is found that a neuron network is in many aspects similar to a network of spin-1/2 particles. A spin-1/2 system has two states, up and down. Similarly, each neuron can be considered as a simple two-level system, with the two possible states being firing or not firing. In our model the two states of neuron i are assigned different values of "spin" s_i, which corresponds to its firing rate of electrical signal. By convention we take $s_i = +1$ to correspond to a firing rate of 1, and $s_i = -1$ to 0 firing rate.

We will assume that there are interactions between every pair of spins (that is neurons; we will use the two terms interchangeably). This is necessary in order to mimic the highly interconnected nature of a real neural net, which is believed to be crucial for its operation. We will find it very useful to consider the effective energy of our neural network/spin system, which can be written as

$$E = -\sum_{i,j} J_{i,j} s_i s_j, \tag{D.1}$$

where the $J_{i,j}$ are related to the strengths of the synaptic connections. The sum here is over all pairs of spins i and j in the network. The exchange, or more properly the synaptic, energies $J_{i,j}$ describe the influence of neuron i on the firing rate of neuron j.

The connections in a real neural network are not symmetric ($J_{i,j} \neq J_{j,i}$ in general). The biological importance of this asymmetry in $J_{i,j}$ is not known. For now we will assume that the connections are symmetric, as this will allow us to make use of some ideas and results from statistical mechanics.

However, we have not yet specified how the exchange energies should be chosen, or even how our lattice of spins can function as a memory. A useful memory must be able to store, recall, and display patterns, so let us consider how these operations can be implemented. The display operation is the easiest and is illustrated in Figure D.I. On the left we show a lattice of spins with a particular spin configuration. This configuration was chosen so that it stores the letter A, which is seen more clearly on the right, where we show the same lattice, but with the spins for which $s = -1$ replaced by blanks. In this way the spins can be used as pixels to display whatever character or other type of pattern is desired.

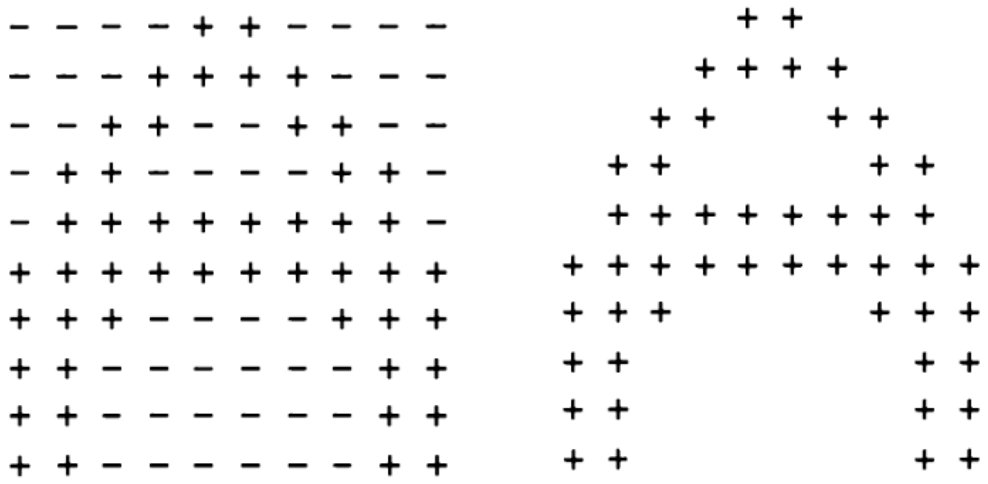


Figure D.I: Left: a 10 X 10 lattice of spins with a particular configuration of + and – spins; right: the same lattice but with the spins for which $s = -1$ replaced by blanks. This network holds (i.e., displays) the letter A.

In order for our memory to recall a pattern, we require that the spin directions change with time in such a way that the spin configuration eventually ends up in the desired state. For example, if we want to recall the letter A, we want the system to take on the configuration in Figure D.I. The recall process thus involves first giving the spins a configuration, then allowing the spin arrangement to evolve with time until the system settles into a new, and hopefully stable, configuration. This time evolution can be accomplished using the Monte Carlo method. Starting from some particular configuration of the entire network, a spin is chosen and the energy required to flip it, ΔE_{flip} is calculated using Eq. (D.1). If ΔE_{flip} is negative, that is, if flipping the spin would lower the energy, the spin is reversed. If $\Delta E_{\text{flip}} \geq 0$, the flipping is accepted with a probability $\exp(-\Delta E_{\text{flip}} / k_B T)$. The Metropolis algorithm ensures that the system will have higher probability to evolve in time to states with lower energy. If conditions are right, the memory will end up in a state in which the spins cease changing with time. This state corresponds to the pattern that is recalled by our memory.

The Monte Carlo procedure is used repeatedly, giving every spin a chance to flip. The spins can be chosen at random, or they can be selected by systematically moving through each row and column of the lattice. The two choices both work well. The most important thing is that each spin be given at least one chance to flip, so the systematic approach is often preferred.

An important feature of human brains is that they are able to generalize in a reasonable manner. For example, you can usually recognize a letter even if it is shown to you in a *different* TYPE **face** OR STYLE. There is some property of A-ness that you are able to

recognize, and all patterns that fall into this class will cause you to recall the same fundamental letter A. The human brains seem to operate as a content addressable memory. This is in contrast to the type of memory we are familiar with in connection with a conventional computer. In the case of a computer memory, a piece of information, such as the value of a particular variable, is given a name or address. In order to recall the value of the variable, the address must be presented to the memory, which is then able to retrieve the desired value. However, with a content addressable memory, information is retrieved by giving a rough description of the information itself. For example, the value of π could be retrieved by giving only the first few digits. This is a very important property of human brains. As another example, we might want to recall a friend's face given only a vague recollection of the shape of her chin and her hair style. In addition, such a memory should be able to recognize her face even if she cuts her hair or dyes it green. Human brains are able to handle such tasks, which are difficult for the more conventional computer-style memories.

For Our model to exhibit similar behavior, it is required that some patterns have energy which are locally minimum, so that if the network starts with nearby patterns in the phase space, time evolution will bring the system automatically to these minimum-energy states. In this way, a set of similar patterns will result in the same final state, giving you the same perception.

For example, on the left side of Figure D.II we show our lattice in a state that is close to the pattern for the letter A that we considered earlier. Even though a few of the spins are in the wrong state, that is, have been flipped with respect to Figure D.I, it should be clear to your (human) brain that this pattern represents the letter A. We hope that if we start with the lattice in this configuration, the system will evolve to the stable state shown on the right side of Figure D.I. The system thus found the ideal letter A, and recalled it. The memory worked as it should.

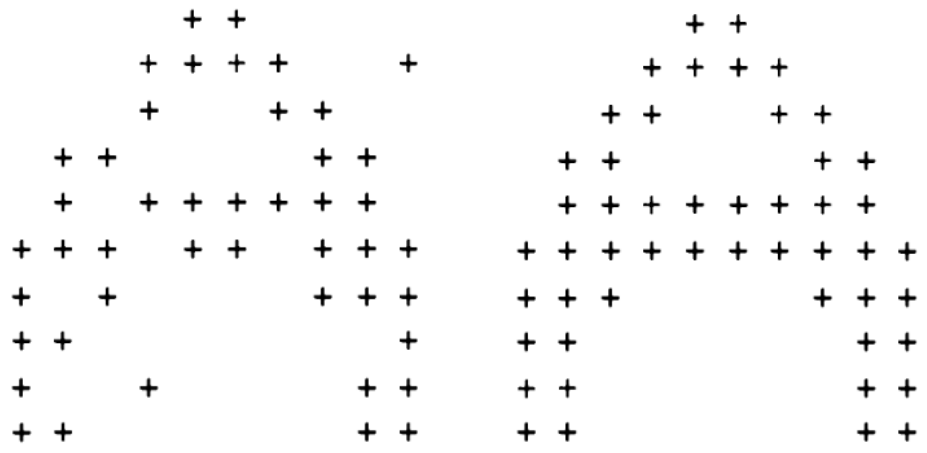


Figure D.II: Left: spin arrangement close to the letter A, but with a few spins flipped to the wrong state; right: final spin configuration when the system evolves.

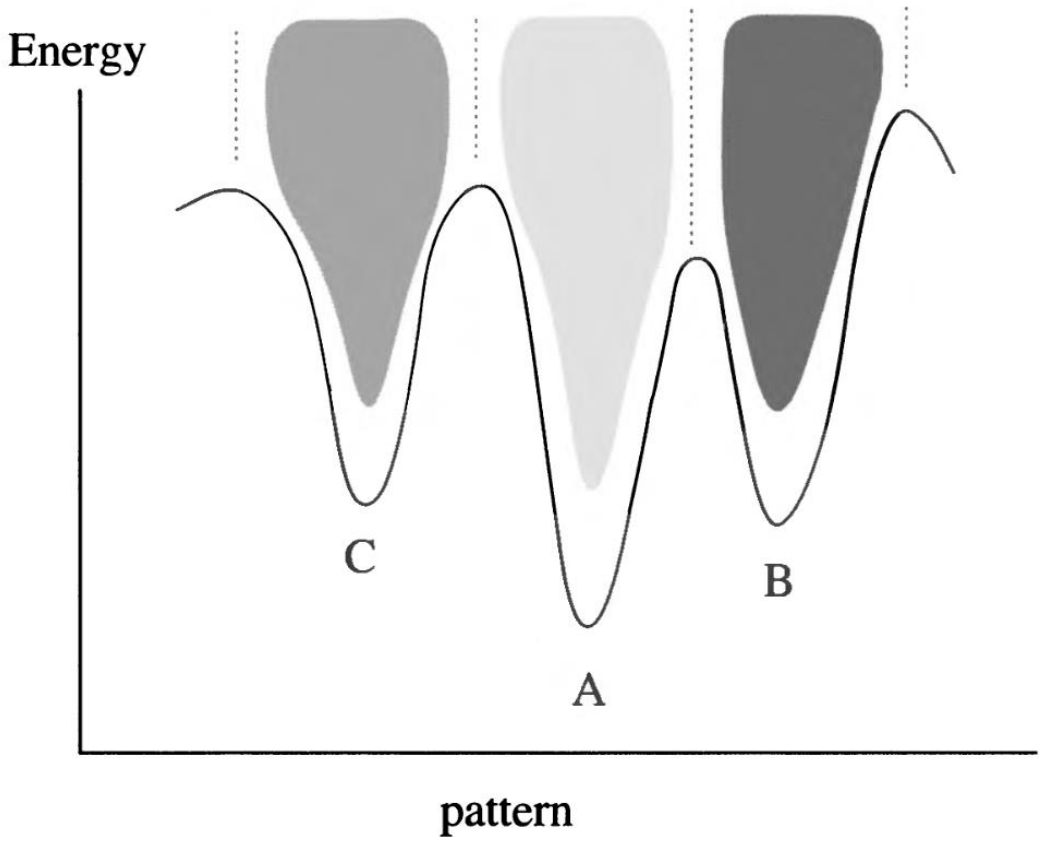


Figure D.III: Schematic energy landscape. The vertical axis is energy as calculated from Eq. (B.1), while the horizontal axis corresponds to the spin configuration. The basins of attraction of several stored patterns are indicated by the shaded areas.

In other words, when presented with a pattern that is similar but not identical to one that it "knows," the memory is able to choose the correct response in a reasonable way. This behavior can be understood in terms of the energy landscape of the spin system. The energy of the network depends on the specific spin configuration, that is, memory pattern, at that time. Since the interactions $J_{i,j}$ have widely varying magnitudes and signs, this energy function will depend in a complicated way on the spin configuration. Schematically it will resemble the landscape shown in Figure D.III. Each stored pattern, e.g. the letters A, B, and C the figure, corresponds to minima in the energy. The Monte Carlo procedure guarantees that the system will evolve in such a way that it is more likely to be found in patterns with lower energies.

In general, the pattern presented to the network will not be one of the stored patterns, so the system will initially be situated on one of the slopes in the energy landscape. The Monte Carlo flipping rules then take it to the bottom of the nearest valley, and it thus ends up "recalling" the stored pattern associated with that minima.

In terms of the energy landscape there will be a basin of attraction in the neighborhood of each minima. If the network is within the basin corresponding to a particular minima, the associated stored pattern is the one that it will eventually locate. Topologically, these basins are the "valleys" surrounding each minima in Figure D.III.

The interaction energies $J_{i,j}$ are the key elements in our model memory. They should be suitably chosen so as to yield a useful memory. Denote the various patterns of the network by m, and let $s_i(m)$ be the configuration of spin i in pattern m, and assume that we want our network to have this configuration as one of its stored patterns. That means that the $J_{i,j}$ must be chosen so as to make the energy a minima for this spin configuration. While there is no unique solution for this problem, a popular and very convenient choice is

$$J_{i,j} = s_i(m)s_j(m). \tag{D.2}$$

We can see that this choice for $J_{i,j}$ will make m an energetically stable pattern as follows. Inserting Eq. (D.2) into Eq. (D.1) yields

$$E(m) = -\sum_{i,j} J_{i,j}s_i s_j = -\sum_{i,j} s_i(m)s_j(m)s_i s_j\ . \tag{D.3}$$

If the network is in pattern m, we will have $s_i = s_i(m)$ and $s_j = s_j(m)$, so each term in this sum will be unity, making the energy large and negative. On the other hand, a random pattern will, on average, have half of its spins flipped with respect to $s_i(m)$, so roughly half of the terms in Eq. (D.3) will be positive and half negative, leading to $E \sim 0$. Thus, our desired pattern will have a much lower energy than a random pattern. Furthermore, the energy of our stored pattern will correspond to a local minima of the landscape, since flipping anyone spin from the pattern value $s_i(m)$ will increase E.

The prescription of (D.2) tells us how to store a single pattern, but a useful memory must be able to store many patterns. In that case we choose the $J_{i,j}$ according to

$$J_{i,j} = \frac{1}{M}\sum_m s_i(m)s_j(m), \tag{D.4}$$

where the index m refers to the stored patterns, and there are a total of M such patterns. The arguments we just gave can be used to show that the energy associated with each stored pattern will be much lower than the energies of a random pattern. In addition, if the stored patterns are sufficiently different from one another, they will each correspond to distinct, well-separated minima in the energy landscape.

Programming

Step 1: Use a 10×10 network and define your own pattern which corresponds to the letter A. Initialize the spins according to the patterns you defined. Also define the elements of $J_{i,j}$ according to Eq. (D.2).

Step 2: Choose 5 spins randomly in the pattern you defined for the letter A. Flip these spins to obtain a pattern that "looks like" A. Set $T = 0$ and run the Monte Carlo steps. You can select the spins to flip systematically by moving through each row and column of the lattice. Observe if the network will return to the origin pattern. Repeat by flipping 10, 20, 30 spins etc. How many spins you need to flip so that the pattern fails to return to the one defined for A?

Step 3: Add two new patterns, for the letters U and V. Define the elements of $J_{i,j}$ according to Eq. (D.4).

The two new patterns look similar. In this case, the memory recall may not be ideal. The system may then recall a pattern that is different from any of the stored patterns (often an approximate mixture), or the network might never locate a time independent state. In the latter case it may switch back and forth in time between two or more patterns; such behavior is known as a limit cycle. In human terms we might think of this as corresponding to confusion!

Start with some initial patterns obtained by flipping some spins in the patterns of U and V, and see whether you can observe this.

Step 4: We next consider the effects of damage on the operation of our model memory. Since the information is stored in the interactions $J_{i,j}$, it is these connection values that will be damaged. We start with the $J_{i,j}$, calculated using Eq. (B.4), and the three stored patterns, A, U, and V. We then "damage" these values by randomly setting the elements of the $J_{i,j}$ matrix to zero with probability P_{damage}.

Set $P_{\text{damage}} = 0.8$ and start with a pattern which resembles A but with 30 spins flipped. Run the code and compare the result with that in step 2 using the same parameters.

Repeat with different values of P_{damage}. What is the largest value of P_{damage} beyond which the network ceases to function properly?

Step 5: Up to now we have used the Monte Carlo procedure for a system at zero temperature. When $T = 0$, the Metropolis algorithm takes the system to the nearest accessible energy minima. A drawback with this method is that a deeper minima will not be located unless it is directly "downhill" from the initial pattern. One way to avoid this problem is to use the Monte Carlo method with $T > 0$. If the effective temperature is comparable to the energy barrier separating a metastable pattern from a more stable one, the Monte Carlo algorithm will enable the system to spend more time near the stable pattern. If the temperature is then slowly reduced, the network can sometimes locate the more stable pattern.

Carry out the procedure and study its performance.

You should find that if T is too small, the network will be trapped in undesired states. On the other hand, if T is too large, the system will quickly move far from the initial pattern and the final state will not resemble the initial one. This is not the way a memory should function. The best performance is obtained with an intermediate value of T, which is just large enough that the system can overcome the smallest energy barriers.

E. Protein Folding

A polymer is constructed by linking together a collection of short molecular segments. Many biological molecules are constructed in this way, including DNA (and its relatives) and proteins. You have probably heard much about the wonders of DNA and the double helix, etc. Here we will focus, instead, on proteins. These are key participants in a very wide variety of biological processes. They are involved in energy storage and conversion, handle communication between cells, are important structural components in several parts of the body including tendons and bones, and are catalysts in many biochemical reactions. Proteins are built from amino acids, which are relatively short molecules containing 10s of atoms. Nature uses only 20 different amino acids to construct all of the known proteins. These 20 monomers are very similar to each other and thus form a fairly homogeneous set of building blocks. A typical protein is made up of a few hundred amino acids linked together end-to-end, although the overall length of a protein can vary greatly. The shortest proteins contain only on the order of 50 monomers, while the longest are comprised of several thousand.

A particular protein is composed of a particular sequence of amino acids. This sequence is known as the primary structure of the protein. It is believed that all of the properties and functions of a protein are uniquely determined by its primary structure.

Proteins can be viewed as chains in which the links are the amino acids. It turns out that the connections between these links are somewhat flexible. In a living cell proteins exist in a solution that is mostly water, and this makes it possible for the chain to assume different shapes. Possible shapes include the two shown schematically in Figure E.I. In one case the protein is folded into a compact "glob," while in the other it is unfolded with an end-to-end length that is close to the maximum possible.

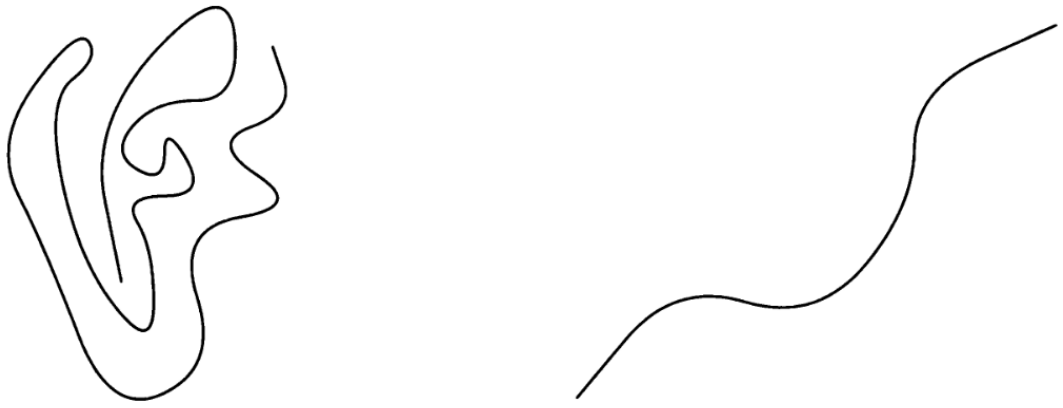


Fig. E.I: Schematic protein in a folded (left) and unfolded state (right).

The shapes of real proteins are generally more complicated than these two very simple illustrations. For example, some proteins form helical sublengths that then fold into sheets or globs. Our point here is only that a long chain molecule with flexible connections

between monomers has the possibility of taking on many different shapes. Which shape is preferred will depend on temperature and the chemicals present in solution. The structure of a protein when it is in its biologically active state is known as the tertiary structure, and this is generally some sort of folded state. The biological functions of a protein are a result of its tertiary structure, since this determines what other molecules a protein can bind to, the kinds of spaces it can fit into, etc. Hence, if we want to understand how a protein works, we must understand its tertiary structure. In addition, if we want to design a new protein with a specific (presumably beneficial) property, we must be able to predict the tertiary structure from knowledge of the primary structure.

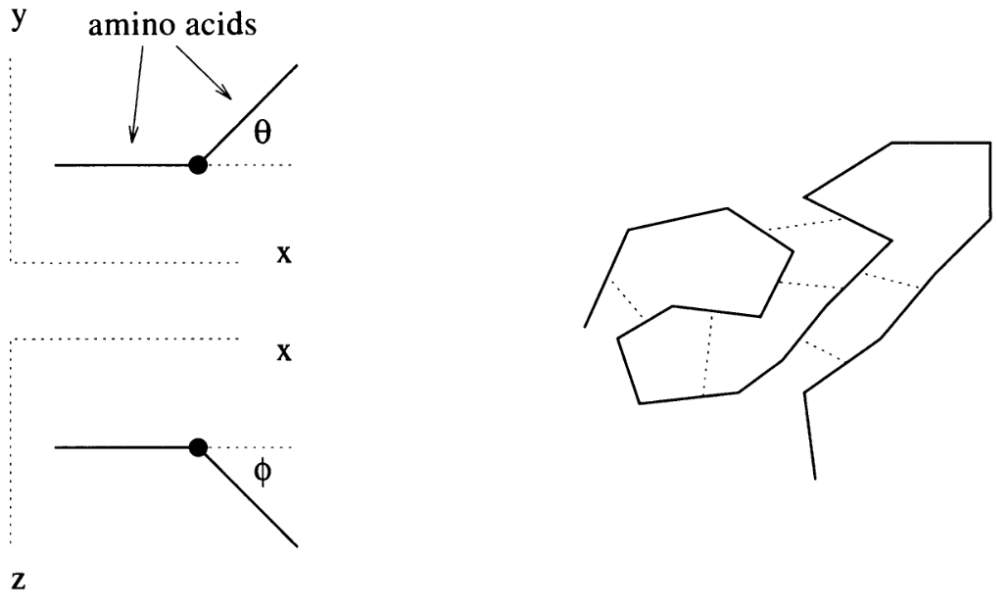


Figure E.II: Left: angle between two amino acids when viewed in a projection in the *x*-*y* plane (top) and *x*-*z* plane (bottom). For simplicity each amino acid is drawn simply as a solid straight line (the side chains are not shown), and the covalent bond between them is represented by the filled circular dot. Right: schematic protein in a folded state. The dotted lines indicate interactions between parts of the chain that are not connected by strong covalent bonds.

While the chemical bonds between adjacent amino acids in a protein chain are not completely rigid, they are also not completely flexible. The relative orientation of two adjacent amino acids in a chain is determined by the covalent chemical bond between the two and can typically assume several different values. This is illustrated in Figure E.II, which shows a hypothetical pair of monomers that are bonded so as to have a relative angle of θ in the *xy* plane and ϕ in the *xz* plane. In most cases the chemical bond permits either or both of these angles to be reversed (that is, $\theta \rightarrow -\theta$) without affecting the bonding energy, yielding four possible choices for the relative orientation of the two amino acids.

When we combine this with four more choices when the next monomer is added to the chain, etc., for a long chain, we can see that there are a large number of different tertiary structures that have the same energy as far as the direct, covalent bonds between the amino acids are concerned. While this picture is a bit simplified, it does illustrate (correctly!) the origin of a protein's flexibility and why it can assume many different particular shapes. If we assume four different relative orientations for each link in a chain of N amino acids, there will be $\sim 4^N$ different possible structures (ignoring self-avoidance constraints). For $N = 300$ this yields $\sim 10^{180}$ distinct tertiary structures, all arising from a single primary structure.

Even though a particular protein can take on a large number of possible structures, the molecule "knows" which particular tertiary structure is the proper one. Otherwise it could not carry out its intended biological functions. Under certain conditions a protein can be induced to unfold, that is, be made to assume the stretched-out shape shown on the right side of Figure E.I. For example, this can often be accomplished by adding the appropriate chemicals to its solution. Such an unfolded structure might also be found when a protein is synthesized initially. Experiments have shown that if an unfolded protein is put back into its biologically "natural" environment, it will fold back into its original structure.

This is remarkable for two reasons. First, the protein manages to find precisely the proper final structure, even though there are an astronomical number of possible alternatives. Second, the time it takes for the protein to refold is typically of the order of seconds. You might have thought that the protein would sample a significant fraction of its $\sim 4^N$ possible states on its way to a final folded structure. It was first pointed out by Levinthal that even if a protein spends only on the order of 10^{-13} s in each of these intermediate states, the folding process would still take longer than the age of the universe! This conundrum is known as Levinthal's paradox.

Since proteins are, in fact, able to fold rapidly into their proper tertiary structure, it would appear that a protein is somehow able to locate the correct structure without searching through all of the possibilities. Precisely how it accomplishes this feat and how it "knows" what the proper tertiary structure should be, is not understood. This is the protein-folding problem.

In order to construct a sensible model of the folding process, we need to consider the different forces and energies in the problem. The largest energy scale is that of the direct covalent bond between adjacent amino acids. This is by far the strongest bonding in the protein, but since different tertiary structures have the same collection of covalent bonds, this energy does not play any role in differentiating between different folded (or unfolded) structures. This differentiation is the result of several other, much weaker forces. One of these is the Van der Waals force between amino acids that are not covalently bonded. While it will tend to bring the monomers together, it will only be important when they are not too far apart. There will also be hydrogen bonds between nearby amino acids; these will lead to attractive forces and thus appear to prefer a folded state for the protein. The Van der Waals and hydrogen bond forces are shown schematically by the dotted lines in Figure E.II.

In addition, we must consider the effect of the water molecules and other chemicals in solution. It turns out that water is attracted to some amino acids and repelled from others. Monomers that are strongly attracted to water molecules will prefer an unfolded structure, since this would allow them to be close to more water molecules. Hence, there will be a competition between the various forces, as some prefer a folded state while others favor an unfolded structure.

The relative importance of these forces will be a complicated function of the particular amino acids in the protein chain and how they are arranged. Obtaining a quantitative understanding of the forces is itself an extremely formidable problem and is certainly more than we want to tackle here.

While a great deal is known about these matters, this remains an area of much research. The key point for us is that folding is a competition between forces. Moreover, these forces are all relatively weak; they turn out to have a magnitude of order $k_B T$ (where T is temperature) per protein. Hence, we have a very delicate interplay between these forces and the disordering effect of temperature.

This has been an extremely brief introduction to proteins, and we have obviously omitted a lot of interesting points. Nevertheless, we have described all of the key features that are needed to construct a model for the folding process. This will be an extremely simple model, and in some ways it will have only a faint resemblance to a real protein. Our goal is to understand how a protein folds and what factors affect the folding process. We begin with a simple model that we believe contains the essential physics. Only after understanding the behavior of this model could we have any hope in attacking a more realistic situation.

We will consider a model protein that sits on a two-dimensional lattice as shown in Figure E.III. We assume a particular primary structure, that is, a sequence of N amino acids, which we denote as $A(1), A(2), \ldots, A(N)$. These are just numbers in the range 1-20 (with no rule against repetitions) corresponding to the 20 different amino acids. Each link in this chain of amino acids is assumed to be located on a site of the lattice, with adjacent, covalently bonded links situated on nearest-neighbor sites.

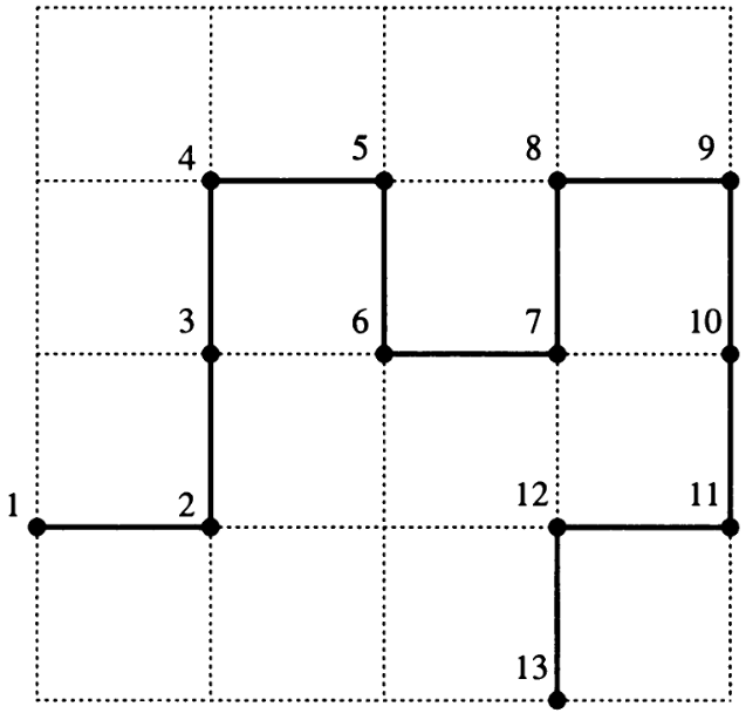


Figure E.III: Lattice model for a protein. The filled circles represent amino acids and the solid lines are covalent bonds between them.

In order to mimic the attractive forces between amino acids that are not covalently bonded, we assume that there is an energy $J_{i,j}$ associated with two amino acids that are nearest neighbors, but that are not covalently bonded, that is, not adjacent in the chain. $J_{i,j}$ thus represents the combined energies of the Van der Waals and hydrogen bonds, as well as the effects of bonding to water molecules. This energy will depend on the specific amino acids that are involved, so i and j are the indices of the two monomers, $i = A(n)$ and $j = A(m)$, where the nth and mth amino acids are situated on nearest-neighbor lattice sites. For example, in the model protein in Figure E.III, amino acids 3 and 6 are nearest neighbors and are not adjacent members of the chain (that is, not directly connected by a covalent bond), so in our model they have an energy of attraction $J_{A(3),A(6)}$. If two amino acids are not nearest neighbors, such as monomers 2 and 6, the energy of attraction is taken to be zero.

The energy of our model protein is then

$$E = \sum_{\langle m,n \rangle} \delta_{m,n} J_{A(n),A(m)} \,, \tag{E.1}$$

where the sum is over all pairs of proteins $\langle m,n \rangle$ in the chain. $\delta_{m,n} = 1$ if amino acids m and n are nearest neighbors that are not connected by direct covalent bonds and is zero

otherwise. Which pairs of amino acids are noncovalently bonded nearest neighbors will depend on the tertiary structure of the protein, so this energy will be a function of the structure. The protein is assumed to be in thermal equilibrium with a heat bath, which is just the solution it is dissolved in, so we can use the rules of statistical mechanics to calculate its properties. We will use the Monte Carlo method in our simulations.

Initial Conditions and Values

I: Choose a primary structure of N links by choosing a sequence of N integers

$A(1), A(2), \ldots, A(N)$ at random from the range 1-20, corresponding to the 20 different possible amino acids. In this project, $N = 30$, unless otherwise specified.

II: The interaction energies $J_{i,j}$ must next be specified. Since there are 20 different possible amino acids [values of $A(m)$], $J_{i,j}$ can be thought of as a 20×20 matrix. In principle, the elements of this matrix could be determined from a quantum mechanical calculation of the Van der Waals forces, hydrogen bonding, etc., but this very formidable task is currently too difficult to tackle in a quantitative way. We will, therefore, take the convenient approach of assuming that the $J_{i,j}$ vary randomly within some specified range. We hope that the important features of the simulation will not depend strongly on how the $J_{i,j}$ are chosen, but that is something we will only know after we have done some work. We shall choose the values of $J_{i,j}$ randomly in the range from -4 to -2 .

III: The final ingredient required for our simulation is the initial tertiary structure. Generate a random structure using self-avoiding random walk.

Monte Carlo Steps

A link in the chain is selected at random by choosing a random integer in the range 1 to N. Let the lattice coordinates of this amino acid be (x_0, y_0). Since we are using a square lattice, this site has four nearest neighbors, and one of these neighbors is next chosen at random; we label it (x_n, y_n). If this neighboring site is not occupied by another amino acid, we then check to see if the monomer at (x_0, y_0) could move to (x_n, y_n) without breaking a covalent bond. This is illustrated in Figure E.IV, which shows a portion of a hypothetical initial structure on the left. Suppose that the amino acid at site 5 is chosen as (x_0, y_0). Site 9 is a nearest neighbor, and the amino acid at site 5 could be moved to this location without breaking (stretching or compressing) either of the bonds to adjacent sites, so this would be an "allowed" move. Site 4 is also a nearest neighbor to site 5, but moving the amino acid to this location would stretch the bonds connecting it to sites 6 and 8, so such a move is prevented by the large energy associated with these covalent bonds. After finding that a move of the amino acid from site 5 to site 9 is permitted by the covalent bond constraints, the Monte Carlo approach is then used to determine if such a move is actually made. The

energy of the chain in both its original structure (on the left in Figure E.IV), and in the potential new structure (on the right) is calculated from Eq. (E.1).

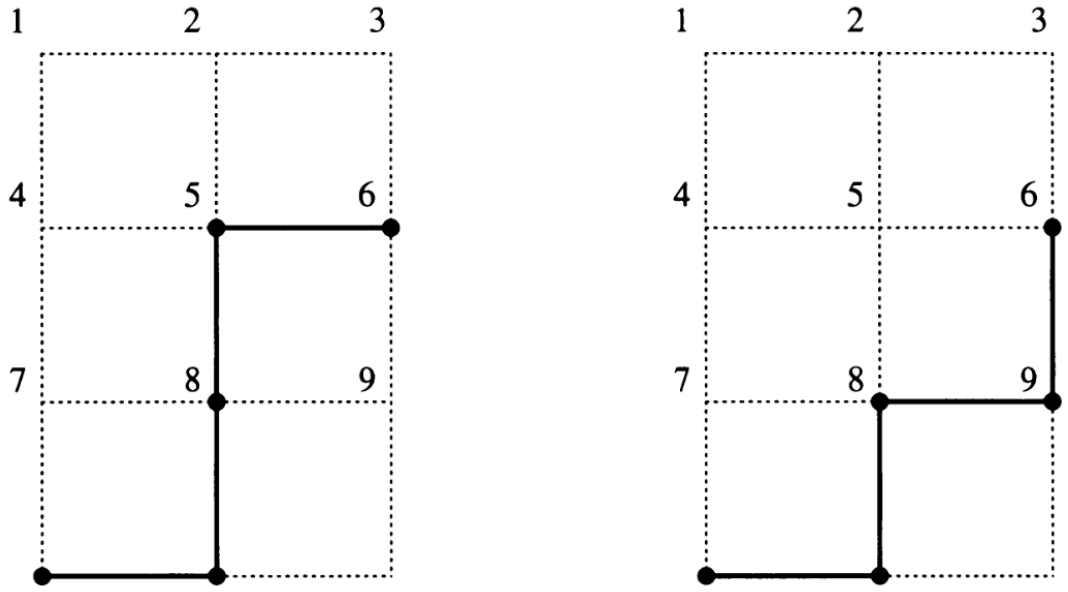


Figure E.IV: Left: hypothetical initial structure of a portion of a protein chain; right: new structure obtained by moving the amino acid initially at lattice site 5 to site 9. Based on the portion of the lattice shown here, these two tertiary structures would have the same energy. However, portions of the chain beyond the right edges of these pictures could cause them to have different energies.

This yields the energy required to make the move, ΔE_{move}. If this energy is negative so that the move would lead to a state with lower energy, the monomer is moved and the structure changes. If ΔE_{move} is positive so that the move would cost energy, the monomer is moved only if the Boltzmann factor $\exp(-\Delta E_{\text{move}}/k_B T)$ is greater than a random number in the range 0-1. Hence, moves that increase the energy are made with a probability given by the Boltzmann factor.

Note: The simulation consists of repeating this procedure a large number of times. Sometimes it results in a new structure, while other times the structure is left unchanged. After each attempted move the system spends one Monte Carlo time step in the resulting state, whether the move is made or not.

Making Movies

Set $k_BT = 1$ and make a movie of the simulation to check whether your code runs correctly. For example, check if the chain is broken and if some acids disappear.

Print out the configuration every 10 steps, for about 1,000 steps.

"Burn-in" Process

The energy of the protein is given by Eq.(E.1), while its "size" Δ is defined to be

$$\Delta = \frac{1}{N}\sum_{i=1}^{N}\left|\mathbf{r}_i - \mathbf{r}_{CM}\right|^2, \qquad \text{(E.2)}$$

where $\mathbf{r}_i$ and $\mathbf{r}_{CM}$ are the position vector of the ith acid and center of mass of the whole chain, respectively.

Plot E and Δ versus number of MC steps. You shall observe that both E and Δ vary until a certain point after which they fluctuate around some constant values. This means the initial state has been "forgotten" and the system attains thermal equilibrium with the heat bath. The steps during this "burn-in" process should be discarded when you calculate the thermal averages in the following steps.

Repeat with $k_BT = 10$ and note any differences.

Thermal Average

After a large number of time steps the protein should reach thermal equilibrium with the heat bath. We can then determine its properties by averaging over the state of the protein during the course of many subsequent Monte Carlo steps.

Set $k_BT = 1$ and plot E and Δ versus number of MC steps after thermal equilibrium has been reached. Show the final configuration of the protein by the movie.

Remember to plot the uncertainties too. If the uncertainties are too large, you should run longer to get better statistics.

Now repeat with an independent run, i.e., with exactly the same protein and same initial configuration, but do the MC steps with a different random number sequence. Plot E versus the MC steps on the same graph as the first run. Show the final configuration of the protein by the movie.

Compare and comment on the result.

Simulated Annealing

Now incorporate simulated annealing into your code. You are suggested to start from $k_BT = 10$ and gradually cool the system down to $k_BT = 1$. Plot E and Δ versus temperature. Show the final configuration of the protein by the movie.

Compare with that without annealing and comment on the result. If you can observe *phase transition* --- most of the change of energy and size occurring inside a certain relatively narrow temperature range, state the transition temperature range.

Now repeat with an independent run, i.e., with exactly the same protein and same initial configuration, but do the MC steps with a different random number sequence. Plot E and Δ versus temperature. Show the final configuration of the protein by the movie.

Compare and comment on the results of the two runs.