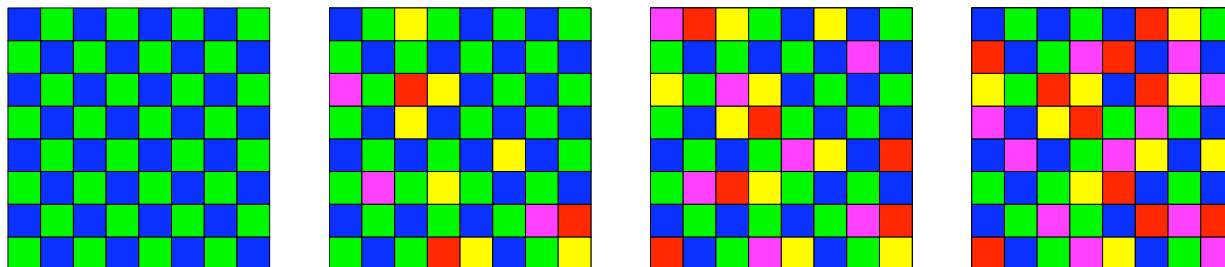
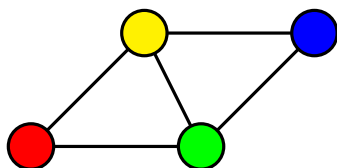


Math 391 – Mixing Times for Markov Chains  
 Summary of Computer Lab #7  
 Monday, Mar. 31

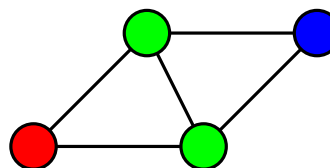
**Goal:** Simulate a sample from an approximately uniform random coloring for the discrete torus.



**Proper coloring:** A proper coloring of a graph  $G = (V, E)$  by  $k$  colors  $c_1, \dots, c_k$  is an assignment of one of the colors to each vertex of the graph, such that no two vertices bounding an edge have the same color. It is simpler to consider the “colors” to be represented just by the numbers  $\{1, \dots, k\}$ . Then we have a function  $f : V \rightarrow \{1, \dots, k\}$  such that for any  $x, y$  with  $\{x, y\} \in E$ , we have  $f(x) \neq f(y)$ .



Proper coloring

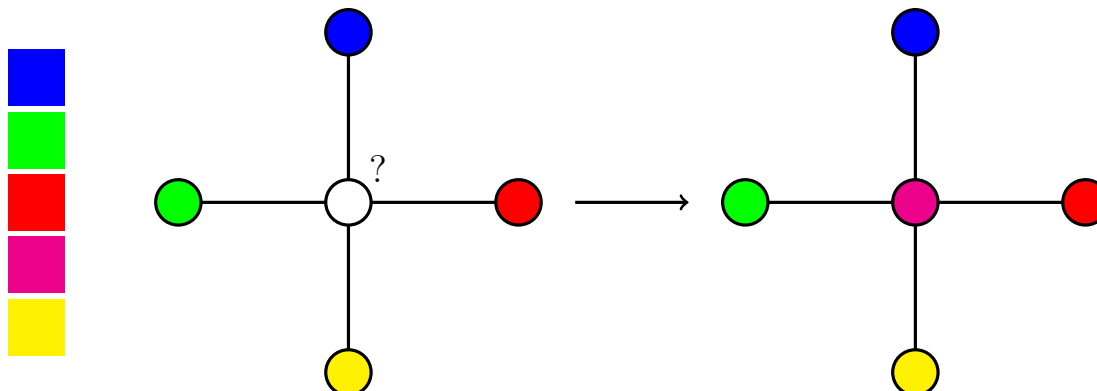


Not a proper coloring

**Fact:** Recall that  $\deg(x) = \#\{y \in V : \{x, y\} \in E\}$  is the degree of  $x$ , the number of neighboring vertices in the graph. If  $\max_{x \in V} \deg(x) = k$ , then you can always make a proper coloring of the graph with  $k + 1$  colors.

Algorithm: (Random greedy algorithm)  
 Chose vertices one at a time from  $V$ .  
 For each vertex, assuming it has not been chosen before, look at the neighbors which have already been colored.  
 Choose a random color among those that have not already been used for any neighboring vertices, and color the vertex that color.  
 Repeat this process until all vertices have been colored.

The reason that this algorithm works is that, even if you get unlucky and you completely surround a vertex by  $k$  different colors, you still have a  $k + 1$ st color to choose from (because you gave yourself  $k + 1$  colors). This is the worst case scenario because the max degree is at most  $k$ .



In this case one would say that the decision problem, of deciding whether or not there is at least one coloring, is trivial. First of all, abstractly we know that there is one. Second of all, we can construct one using the algorithm above, which only takes on the order of  $|V|$  operations, since for each vertex we simply choose the color greedily, in other words, without having to worry about future steps.

**Q1:** Consider a finite block of the square lattice, or consider the discrete torus with an even number  $L$  of vertices on each side. What is the minimum number of colors needed to color it?

**A1:** Two. We can color the graph in a checkerboard fashion. In the first sequence of pictures (showing outcomes of a Markov chain for generating a random coloring) we have colored the  $8 \times 8$  discrete torus using just two colors: red and blue. In other words, the graph is bipartite. (This is also the reason that we need to make the random walk on such a graph lazy: otherwise the random walk will be periodic with period two. Otherwise, a walker who starts on a blue square would always be on another blue square at all even times, and on a green square at all odd times.)

**Task 1:** Generate an  $L \times L$  matrix with 1's and 2's alternating in a checkerboard fashion. For definiteness, let us take  $L = 8$ . Then, suppose we can generate a vector

$$a = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \\ \vdots \\ -1 \\ 1 \end{bmatrix}.$$

So  $a_k = (-1)^k$ . Then taking  $A = aa^T$  (where  $a^T$  represents the transpose) we have that  $a$  is a  $L \times 1$  matrix and  $a^T$  is a  $1 \times L$  matrix. So  $A$  is an  $L \times L$  matrix, and

$$A_{jk} = (aa^T)_{jk} = a_{j,1}(a^T)_{1,k} = a_{j,1}a_{k,1} = a_j a_k = (-1)^j (-1)^k = (-1)^{j+k}.$$

Now, if we take  $B_{jk} = \frac{3}{2} + \frac{1}{2}A_{jk}$  we have  $B_{jk} = \frac{3}{2} + \frac{1}{2}$  or  $B_{jk} = \frac{3}{2} - \frac{1}{2}$ , depending on whether  $j + k$  is even or odd. In other words,  $B_{jk} = 2$  if  $j + k$  is even, and  $B_{jk} = 1$  if  $j + k$  is odd.

For reasons which will be clear later, we would like to take  $N = L + 2$  and make an  $N \times N$  matrix. In other words, we are going to embed the  $L \times L$  coloring matrix into a bigger matrix with a border of width one around the original.

For Matlab, start by typing the following:

```
>> L = 8
L =
     8
>> N = L+2
N =
    10
>> a = (-1).^(1:N)
a =
    -1     1    -1     1    -1     1    -1     1    -1     1
>> A = a'*a
A =
     1    -1     1    -1     1    -1     1    -1     1    -1
    -1     1    -1     1    -1     1    -1     1    -1     1
     1    -1     1    -1     1    -1     1    -1     1    -1
    -1     1    -1     1    -1     1    -1     1    -1     1
     1    -1     1    -1     1    -1     1    -1     1    -1
    -1     1    -1     1    -1     1    -1     1    -1     1
     1    -1     1    -1     1    -1     1    -1     1    -1
    -1     1    -1     1    -1     1    -1     1    -1     1
     1    -1     1    -1     1    -1     1    -1     1    -1
    -1     1    -1     1    -1     1    -1     1    -1     1
    -1     1    -1     1    -1     1    -1     1    -1     1

>> B = (3/2) + (1/2)*A
B =
     2     1     2     1     2     1     2     1     2     1
     1     2     1     2     1     2     1     2     1     2
     2     1     2     1     2     1     2     1     2     1
     1     2     1     2     1     2     1     2     1     2
     2     1     2     1     2     1     2     1     2     1
     1     2     1     2     1     2     1     2     1     2
     2     1     2     1     2     1     2     1     2     1
     1     2     1     2     1     2     1     2     1     2
```

Let us construct a way to visualize this coloring. We will let 1 stand for blue and 2 stand for green. For reasons we will explain, albeit heuristically, momentarily, we will want to have five colors. So we let 3 stand for red, 4 stand for magenta and 5 stand for yellow.

Then we make a script file to graph the coloring we have generated of an  $L \times L$  square (with periodic boundary conditions).

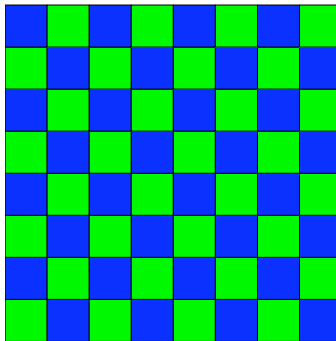
```
>> edit colorgrapher.m
```

As usual, it will ask you if you are sure that you want to make a new script file. Click “Yes.”

```
color = 'bgrmy';
figure
hold on
for m=1:L
    for n=1:L
        rectangle('position',[m,n,1,1],'facecolor',color(B(m+1,n+1)))
    end
end
axis equal
axis off
```

If you then (save the script file `colorgrapher.m` and) return to the command window and type the following, then you will get the first picture from the sequence of four pictures at the beginning:

```
>> colorgrapher
```



It is quite important to understand what is going on, just to get a working knowledge of Matlab’s language. You should type `help rectangle` to get some idea.

**Command:** `if ...then ...end`

An important operation in every programming language is the “if-then” statement. For Matlab there are some important things to know. The way to negate the equality is with a tilde. So  $x \sim= y$  is read as the statement  $x \neq y$ . Another important question is how to concatenate statements. It is with two `&` symbols. So  $(x \sim= y) \&\& (z = y)$  means  $(x \neq y) \wedge (z = y)$ . We have used the  $\wedge$ -notation from set theory. But you can think of this another way. If we take the set of all  $(x, y, z) \in \Omega$  in some sample space  $\Omega$ , such that they satisfy the concatenated statement above, then the resulting set is  $\{x \neq y\} \cap \{z = y\}$ . Notice the similarity between the  $\wedge$  and the  $\cap$  symbol.

Now that we have explained this symbol a little bit, let us write a new script for Matlab. We will break it down and analyze it after we write it. Type:

```
>> edit colorMCstep
```

Of course, as usual, Matlab will ask if you really want to edit the file. But answer, “Yes.” In the editor window type the following:

```
m = ceil(L*rand);
n = ceil(L*rand);
c = ceil(5*rand);
if (B(m,n+1) ~ = c) && (B(m+1,n) ~ = c) && (B(m+2,n+1) ~ = c) && (B(m+1,n+2) ~ = c)
    B(m+1,n+1) = c;
    B(1,:) = B(L+1,:);
    B(L+2,:) = B(2,:);
    B(:,1) = B(:,L+1);
    B(:,L+2) = B(:,2);
end;
```

**Commenting:** (for the Matlab script):

A good program should be commented: that is, you should write comments in your program code to explain what

you are doing with each of the pieces of the program, so that if you go back to look at your code later, you can quickly and easily remember what you are doing. These are notes. At the present time I want to do this for the script above, but also with the goal of explaining the Matlab language as well (for students who do not know it yet).

Recall from Lab 3 that the command `ceil(n*rand)` generates a random integer between 1 and  $n$ . Therefore `n=ceil(L*rand)` and `m=ceil(L*rand)` generate two random integers,  $m$  and  $n$ , in the set  $\{1, \dots, L\}$ . We are going to attempt to update the color of the vertex in the grid at the position  $(m, n)$ . So the first step is to choose a random vertex, and then to update that one.

We also do a similar thing with `c=ceil(5*rand)`, but now  $c$  is only a random integer in  $\{1, \dots, 5\}$ . Why is this? That is because  $c$  controls the color, not the position. More specifically, we are going to try to update the color at the random position  $(m, n)$  to the random color  $c$  (or rather the color, blue, green, red, magenta or yellow depending on whether  $c$  is 1, 2, 3, 4, or 5). Now in order to change to color  $c$  we have to check if this is allowed. To remain a proper coloring, we need to check that none of the neighbors of the vertex at  $(m, n)$  have color  $c$ .

Now, because we are on the torus, if  $(n, m)$  happens to be  $(1, 1)$  then one of the neighbors is  $(L, 1)$  on the other side of the torus, and one of them is  $(1, L)$ , on the bottom of the torus. An easy way to deal with this is to take the color matrix for the vertices  $(m, n)$  with  $m, n \in \{1, \dots, L\}$ , and embed it in a bigger matrix, with two extra columns and two extra rows. Rather, if we would think of  $\tilde{B}$ , an  $L \times L$  matrix, as the color matrix for the torus, we make it the middle  $L$  rows and columns of a bigger matrix  $B$ , which is an  $(L + 2) \times (L + 2)$  matrix. This is why we made  $B$  have dimension  $N = L + 2$ , rather than  $L$  to begin with. We make the first row of  $B$  the same as the  $L$ th row of  $\tilde{B}$ , which is the  $L + 1$ st row of  $B$ . We make the last row of  $B$  the same as the 1st row of  $\tilde{B}$ , which is the 2nd row of  $B$ . We also do the same with the columns. Thus, since  $\tilde{B}$  only represents the entries of  $B$  with both indices between 2 and  $L + 1$ , all of its neighbors do have both indices between 1 and  $L + 2$ . So there is no need to treat  $(1, 1)$  or any other entry specially.

One effect of this fix is that we will update  $B(m + 1, n + 1)$ , not  $B(m, n)$ . So the neighboring colors are  $B(m, n + 1)$ ,  $B(m + 1, n)$ ,  $B(m + 2, n + 1)$  and  $B(m + 1, n + 2)$ . We need to check that none of these have the same color as  $c$ , the one we want to update  $B(m + 1, n + 1)$  to. Therefore, we need to run an `if ... then` statement, to check the following condition

$$(B(m,n+1) \sim= c) \ \&\& \ (B(m+1,n) \sim= c) \ \&\& \ (B(m+2,n+1) \sim= c) \ \&\& \ (B(m+1,n+2) \sim= c)$$

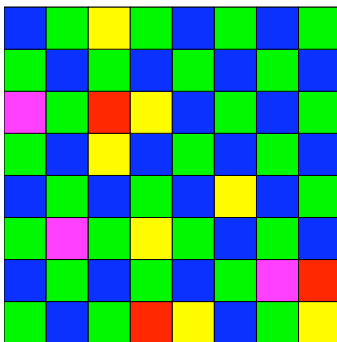
This explains the first line of the `if ... then` statement. We will only change the color  $B(m + 1, n + 1)$  to  $c$  if this condition is verified, since this is the condition necessary to check that the new coloring is a proper coloring. If any part of the condition is violated then the `if ... then` loop terminates without doing anything. In that case the new coloring is the same as the old coloring, because we didn't change anything. But if the condition is satisfied, then the first thing we do is change  $B(m + 1, n + 2)$  to  $c$ , which is the meaning of the command `B(m+1,n+1) = c;`.

Now, since the interior point  $(m + 1, n + 1)$  has been changed, we should, in principle, change the borders. The first and last rows and columns of  $B$  are dependent on what is happening in  $B(j, k)$  for  $j, k \in \{2, \dots, L + 1\}$ . Therefore, since one of these numbers has changed (namely  $B(m + 1, n + 1)$ ) we re-adjust the values of  $B$  on the border. That is the reason for the four command `B(1,:) = B(L+1,:); B(L+2,:) = B(:,:); B(:,1) = B(:,L+1); B(:,L+2) = B(:,2);`. This basically completely explains the Matlab script.

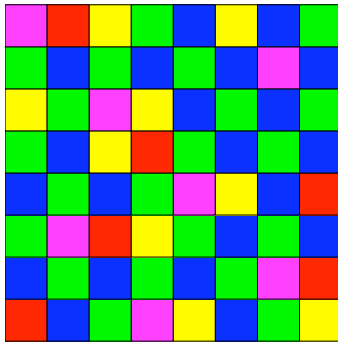
The mathematical analysis of this chain is slightly more involved. In particular, it is easy to see that the chain is lazy. Therefore, if it is irreducible, then it is aperiodic. But it does not seem trivial to check that it is irreducible. On the other hand, irreducibility may not be the most important question for this Markov chain.

We can now run our Matlab scripts to see the evolution of this Markov chain.

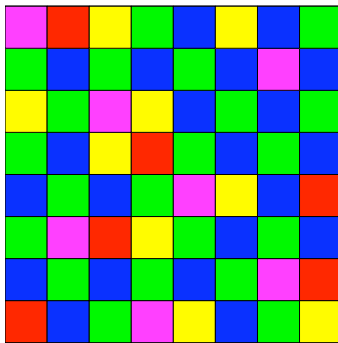
```
>> for t=1:25, colorMCstep, end
>> colorgrapher
```



```
>> for t=26:75, colorMCstep, end
>> colorgrapher
```



```
>> for t=76:175, colorMCstep, end
>> colorgrapher
```



**Assignment:** Repeat the whole procedure with  $L = 10$  and send me the four figures you get at times  $t = 0$ ,  $t = 25$ ,  $t = 75$  and  $t = 175$ . Do you think that  $t = 175$  is enough to mix-up the Markov chain when  $L = 10$ ?