Assignment_5_notebook

October 18, 2017

In [1]:
```
import numpy as np
import pylab as py
import matplotlib.pyplot as plt
import math as math
```

We will be simulation percolation. We first will initialize a lattice of ones and zeros and giving a minimal probability p. We do this by looping over a lattice of points and throwing a random number between 0 and 1. If the number thrown at that lattice point is less than p, we initialize the lattice point to 1, zero otherwise.

In [2]:
```
def init_lattice(N,p):

    lat = np.zeros((N,N), float)
    for i in range(0,N):
        for j in range(0,N):
            u = np.random.random()
            if (u < p):
                lat[i,j]=1
            else:
                lat[i,j]=0

    return lat
```

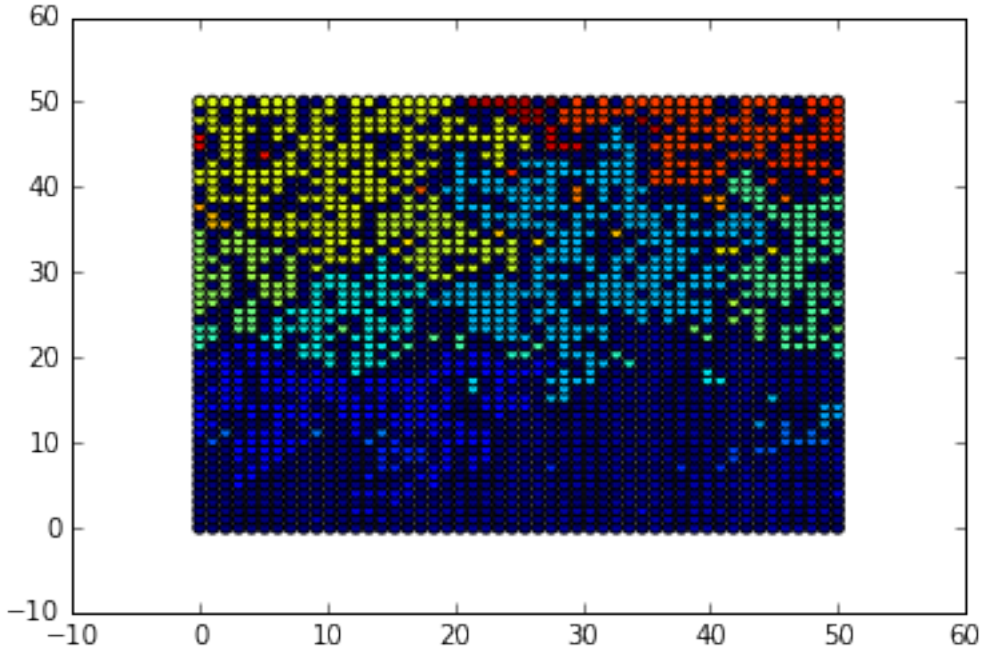
Next we make clusters by using the Hoshen Kopelman algorithm.

In [5]:
```
def make_clusters(lat, N):
    largest_label = 1
    labels = np.zeros((N,N))

    for i in range(0,N):
        for j in range(0,N):

            if (lat[i,j] == 1):
                #print("non zero node")

                if (i!=0):
                    above = lat[i-1,j]
                else:
```

```
                    above = 0
                if (j!=0):
                    left = lat[i, j-1]
                else:
                    left = 0

                if (left == 0 and above == 0):
                    #Assign a new label to this node
                    largest_label = largest_label + 10
                    labels[i,j] = largest_label

                if (left != 0 and above == 0):
                    #if the left
                    left_label = labels[i,j-1]
                    labels[i,j] = left_label

                if (left == 0 and above != 0):
                    above_label = labels[i-1, j]
                    labels[i,j] = above_label

                if (left != 0 and above != 0):
                    labels[i,j] = union(N, labels[i-1,j], labels[i,j-1], labels)

            else:
                labels[i,j] = 0

    return labels

def union(N,label_1, label_2, label_array):
    smaller_label = 0
    larger_label = 0
    if (label_1 > label_2):
        smaller_label = label_2
        larger_label = label_1
    else:
        smaller_label = label_1
        larger_label = label_2

    for i in range(0,N):
        for j in range(0,N):
            if (label_array[i,j] == larger_label):
                label_array[i,j] = smaller_label

    return smaller_label
```

In [8]:
```
N = 50
p = .58
```

```
lat = init_lattice(N, p)
labels = make_clusters(lat, N)
xs = np.linspace(0,N,N)
ys = np.linspace(0,N,N)
X,Y = np.meshgrid(xs,ys)

plt.scatter(X,Y,c=labels, marker = 'o')
plt.show()
```

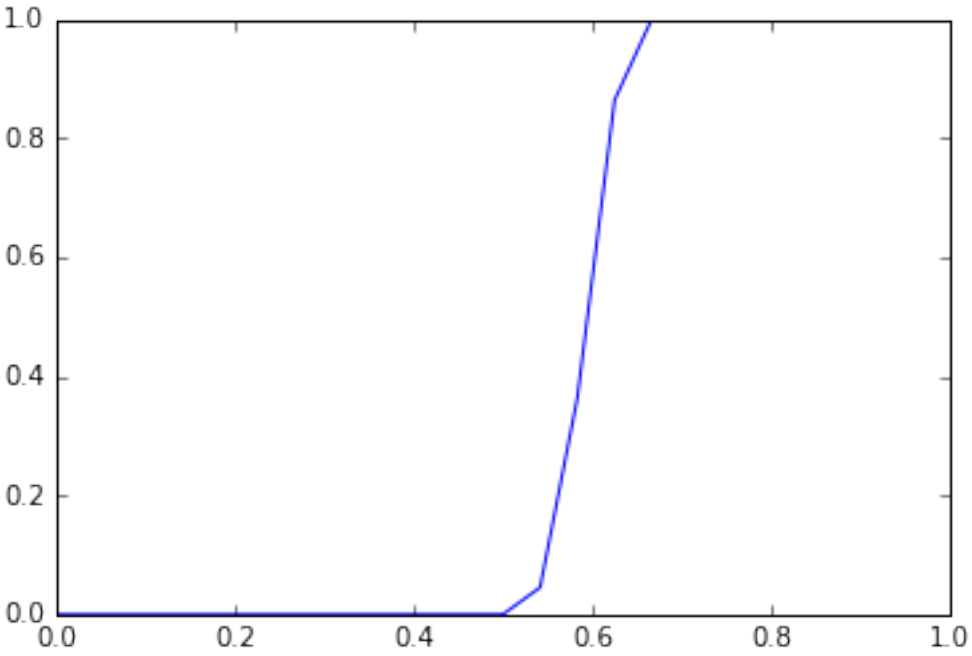


We have labeled clusters with different colors to show how percolation is occuring. Next we compute the probability that for a given p, a complete percolation will occur. We define a complete percolation as a cluster which starts at the top row and continues to the bottom row.

```
In [12]: def check_percolation(labels, N):
             for i in range(0,N):
                 if (labels[0,i] != 0):
                     label = labels[0,i]
                 else:
                     continue
                 for j in range(0,N):
                     if (labels[N-1,j] == label):
                         return 1
             else:
                 return 0

In [13]: N = 50
```

```
        a = np.zeros(25)
        trials = 22
        n = 0

        ps = np.linspace(0,1,25)
        for i in ps:
            print("testing p = ", i)
            for x in range (0,trials):
                lat = init_lattice(N, i)
                labels = make_clusters(lat, N)
                if(check_percolation(labels,N) == 1):
                    a[n] += 1/trials
            n+=1

        plt.plot(ps, a)
        plt.show()
```

```
testing p =  0.0
testing p =  0.0416666666667
testing p =  0.0833333333333
testing p =  0.125
testing p =  0.166666666667
testing p =  0.208333333333
testing p =  0.25
testing p =  0.291666666667
testing p =  0.333333333333
testing p =  0.375
testing p =  0.416666666667
testing p =  0.458333333333
testing p =  0.5
testing p =  0.541666666667
testing p =  0.583333333333
testing p =  0.625
testing p =  0.666666666667
testing p =  0.708333333333
testing p =  0.75
testing p =  0.791666666667
testing p =  0.833333333333
testing p =  0.875
testing p =  0.916666666667
testing p =  0.958333333333
testing p =  1.0
```



As we can see there is a sharp phase transition near p=.58

In [ ]: