Jacob Dennis

Physics 411

11 April 2016

Prof. Gull

Homework 8

**Site Percolation on the 2D Lattice**

*Code 1 – Determining the Coefficients of the RG Transformation Equation*

import numpy as np

import itertools

def CalculateNumberofFilledSpots(seq):

num = 0

for i in seq:

if i == 1:

num += 1

return num

def GetCoefficients(numFilled):

SetOfGrids = []

for seq in itertools.product(range(2), repeat = 9):

SetOfGrids.append(np.array(seq))

numPercolated = 0

for seq in SetOfGrids:

if CalculateNumberofFilledSpots(seq) == numFilled:

if np.all(seq[::3]):

numPercolated += 1

elif np.all(seq[1::3]):

numPercolated += 1

elif np.all(seq[2::3]):

numPercolated += 1

elif np.all(seq[2:7]):

numPercolated += 1

elif np.all(np.concatenate((np.array([seq[0]]), seq[3:6], np.array([seq[8]])))): #3 - length Tetris shape 1:

# x00

# xxx

# 00x

numPercolated += 1

elif np.all(np.concatenate((np.array([seq[2]]), seq[3:6], np.array([seq[6]])))): #3 - length Tetris shape 1:

# 00x

# xxx

# x00

numPercolated += 1

elif np.all(np.concatenate((np.array([seq[1]]), seq[3:5], np.array([seq[6]])))): #2 - length Tetris shape 1:

# 0x0

# xx0

# x00

numPercolated += 1

elif np.all(np.concatenate((np.array([seq[0]]), seq[3:5], np.array([seq[7]])))): #2 - length Tetris shape 1:

# x00

# xx0

# 0x0

numPercolated += 1

elif np.all(np.concatenate((np.array([seq[2]]), seq[4:6], np.array([seq[7]])))): #2 - length Tetris shape 1:

# 00x

# 0xx

# 0x0

numPercolated += 1

elif np.all(np.concatenate((np.array([seq[1]]), seq[4:6], np.array([seq[8]])))): #2 - length Tetris shape 1:

# 0x0

# 0xx

# 00x

numPercolated += 1

print 'Number of Percolated Clusters:', numPercolated

return None

def main():

GetCoefficients(9)

GetCoefficients(8)

GetCoefficients(7)

GetCoefficients(6)

GetCoefficients(5)

GetCoefficients(4)

GetCoefficients(3)

if \_\_name\_\_ == '\_\_main\_\_':

main()

*Results 1*

Number of Percolated Clusters: 1

Number of Percolated Clusters: 9

Number of Percolated Clusters: 36

Number of Percolated Clusters: 67

Number of Percolated Clusters: 59

Number of Percolated Clusters: 22

Number of Percolated Clusters: 3

*Code 2 – Fixed Points and the Critical Exponent*

import numpy as np

import numpy.random as rand

import matplotlib.pyplot as plt

def R(p):

factors = np.array([1.0, 9.0, 36.0, 67.0, 59.0, 22.0, 3.0, 0.0, 0.0, 0.0])

R = 0.0

for i in range(9):

R += factors[i] \* p\*\*(9.0 - i) \* (1. - p)\*\*i

return R

def RminusP(p):

return R(p) - p

def Rprime(p):

return (1.0 - p)\*\*2 \* p\*\*2 \* (9. \* p\*\*4 - 30. \* p\*\*3 + 24. \* p\*\*2 + 34. \* p + 9)

def BisectionMethod(a, b, f, tol):

iterationnum = 0

while (abs(a - b) > tol) & (iterationnum < 10000):

xGuess = (b - a) / 2 + a

if f(xGuess) == 0.:

return xGuess

elif np.sign(f(xGuess)) == np.sign(f(a)):

a = xGuess

elif np.sign(f(xGuess)) == np.sign(f(b)):

b = xGuess

iterationnum += 1

return xGuess

def main():

pRange = np.linspace(0.0, 1.0, 100)

fixed1 = BisectionMethod(-0.1, 0.1, RminusP, .001)

fixed2 = BisectionMethod(0.9, 1.1, RminusP, .001)

pc = BisectionMethod(0.1, 0.9, RminusP, .001)

print 'Fixed Point 1: p =', fixed1

print 'Fixed Point 2: p =', fixed2

print 'Critical probability: p =', pc

print 'nu = ', np.log(3.0) / np.log(Rprime(pc))

if \_\_name\_\_ == '\_\_main\_\_':

main()

*Results 2*

Fixed Point 1: p = 0.0

Fixed Point 2: p = 1.0

Critical probability: p = 0.61953125

nu = 1.77109520864

**Monte Carlo Integration**

*Code*

import numpy as np

import numpy.random as rand

import sympy as sym

def GenerateRandomPoints(numPoints, sizePoints):

Points = rand.random((numPoints, sizePoints))

return Points

def f(size10array):

return (np.sum(size10array) \*\* 2)

def MonteCarloError(f, N):

return np.var(f) / np.sqrt(N)

def main():

x1, x2, x3, x4, x5, x6, x7, x8, x9, x10 = sym.symbols('x1 x2 x3 x4 x5 x6 x7 x8 x9 x10')

integrand = (x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10)\*\*2

print 'int((x1 + x2 + ... + x10)\*\*2), d(x1, x2, ..., x10), 0, 1) is exactly', sym.integrate(

integrand, (x1, 0, 1), (x2, 0, 1), (x3, 0, 1), (x4, 0, 1), (x5, 0, 1),

(x6, 0, 1), (x7, 0, 1), (x8, 0, 1), (x9, 0, 1), (x10, 0, 1))

randomPoints = GenerateRandomPoints(1000, 10)

I = 0.0

fAtPoints = []

for point in randomPoints:

I += 1.0 / 1000 \* f(point)

fAtPoints.append(1.0 / 1000 \* f(point))

print 'Monte Carlo Integration Results:', I, '+/-', MonteCarloError(fAtPoints, 1000)

print 'Expected Results:', 155.0/6.0

if \_\_name\_\_ == '\_\_main\_\_':

main()

*Results*

int((x1 + x2 + ... + x10)\*\*2), d(x1, x2, ..., x10), 0, 1) is exactly 155/6

Monte Carlo Integration Results: 26.300402099 +/- 2.65065005672e-06

Expected Results: 25.8333333333

**Monte Carlo Integration II**

*Code*

import numpy as np

import numpy.random as rand

import matplotlib.pyplot as plt

def GenerateRandomWalk(t):

x = 0.0

for i in np.arange(t):

randNum = rand.random\_integers(0, 1)

if randNum == 0:

x += -1.0

elif randNum == 1:

x += 1.0

return x

def GetSetsofWalks(numSets, t):

X = []

for n in np.arange(numSets):

X.append(GenerateRandomWalk(t))

return np.array(X)

def main():

meanXs = []

meanX2s = []

tRange = np.arange(0.0, 10.0)

for t in tRange:

randomWalks = GetSetsofWalks(500, t)

meanXs.append(np.mean(randomWalks))

meanX2s.append(np.mean(randomWalks\*\*2))

plt.figure(0)

plt.clf()

plt.ylim(-1.0, 1.0)

plt.plot(tRange, meanXs)

plt.xlabel('t')

plt.ylabel('<x>')

plt.title('Position of a Random Walk \n (Averaged over 500 Walks)')

plt.savefig('Homework 8 - Average X for a Random Walk.png')

plt.figure(1)

plt.clf()

plt.plot(tRange, meanX2s)

plt.xlabel('t')

plt.ylabel('<x\*\*2>')

plt.title('Position of a Random Walk Squared \n (Averaged over 500 Walks)')

plt.savefig('Homework 8 - Average X\*\*2 for a Random Walk.png')

if \_\_name\_\_ == '\_\_main\_\_':

main()

*Results*



