Theoretical Biophysics
A Computational Approach
Concepts, Models, Methods and Algorithms
Pattern and Structure Formation

Dieter W. Heermann

April 7, 2020

Heidelberg University

Table of Contents

Introduction

Introduction I

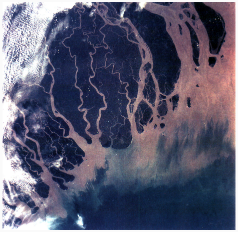
Pine cone

Source:http://jadecrompton.blogspot.de/ 2012/02/fibonacci-in-nature.html

Snowflake

Source: Wikipedia

https://en.wikipedia.org/wiki/Snowflake



River delta

Source: Wikipedia

https://en.wikipedia.org/wiki/River_delta

Activator-Inhibitor systems

Activator-Inhibitor systems I

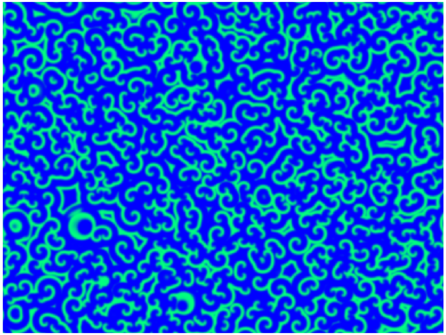


Figure 1: Snapshot of the Oregonator model.

Activator-Inhibitor systems II

Consider the following reactions scheme (**Belousov-Zhabotinsky reaction**) among the A, B, P, Q, X, Y, Z, where the concentrations of species A and B are kept constant and the products P and Q are constantly removed. Thus, only three variables need to be taken into account. F in reaction (V) is a further parameter.

$$\begin{aligned}
I &\quad A + Y && \longrightarrow X && (1)\\
II &\quad X + Y && \longrightarrow P && (2)\\
III &\quad B + X && \longrightarrow 2X + Z && (3)\\
IV &\quad 2X && \longrightarrow Q && (4)\\
V &\quad Z && \longrightarrow FY && (5)\\
& && && (6)
\end{aligned}$$

This leads to the following differential equation system

Activator-Inhibitor systems III

$$\frac{d[X]}{dt} = k_1[A][Y] + k_3[B][X] - k_4[X]^2 \tag{7}$$

$$\frac{d[Y]}{dt} = -k_1[A][Y] - k_3[X][Y] - fk_5[Z] \tag{8}$$

$$\frac{d[Z]}{dt} = k_3[B][X] - k_5[Z] \tag{9}$$

defining the **Oregonator model**. It turns out, that this shows an oscillating chemical reaction.

Activator-Inhibitor systems

```
#https://scipython.com/blog/simulating-the-belousov-zhabotinsky-
    reaction/


import numpy as np
from scipy.signal import convolve2d
import matplotlib.pyplot as plt
from matplotlib import animation

# Width, height of the image.
nx, ny = 600, 450
# Reaction parameters.
alpha, beta, gamma = 1, 1, 1

def update(p,arr):
    """Update arr[p] to arr[q] by evolving in time."""

    # Count the average amount of each species in the 9 cells
    around each cell
    # by convolution with the 3x3 array m.
    q = (p+1) % 2
    s = np.zeros((3, ny,nx))
```

./progs/oregonator–2.py

Activator-Inhibitor systems

```
    m = np.ones((3,3)) / 9
    for k in range(3):
        s[k] = convolve2d(arr[p,k], m, mode='same', boundary='
    wrap')
    # Apply the reaction equations
    arr[q,0] = s[0] + s[0]*(alpha*s[1] - gamma*s[2])
    arr[q,1] = s[1] + s[1]*(beta*s[2] - alpha*s[0])
    arr[q,2] = s[2] + s[2]*(gamma*s[0] - beta*s[1])
    # Ensure the species concentrations are kept within [0,1].
    np.clip(arr[q], 0, 1, arr[q])
    return arr

# Initialize the array with random amounts of A, B and C.
arr = np.random.random(size=(2, 3, ny, nx))

# Set up the image
fig, ax = plt.subplots()
im = ax.imshow(arr[0,0], cmap=plt.cm.winter)
ax.axis('off')

def animate(i, arr):
```

./progs/oregonator–2.py

Activator-Inhibitor systems

```
    """Update the image for iteration i of the Matplotlib
     animation."""

    arr = update(i % 2, arr)
    im.set_array(arr[i % 2, 0])
    return [im]

anim = animation.FuncAnimation(fig, animate, frames=200, interval
    =5,
                               blit=False, fargs=(arr,))

# To view the animation, uncomment this line
plt.show()

# To save the animation as an MP4 movie, uncomment this line
#anim.save(filename='bz.mp4', fps=30)
```

./progs/oregonator–2.py

Fractals

Fractals I

If every point in a set S has arbitrarily small neighbourhoods whose boundaries do not intersect S, then S has a topological dimension of 0.

The **topological dimension** of a subset of $S \subseteq \mathbb{R}^n$ is the least non-negative interger k such that each point of S has arbitrarily small neighbourhoods whose boundaries meet S in a set of dimension $k-1$.

A set S is **self-similar** (deterministically), if it can be divided into N congruent subsets, each of which when magnified by a constant factor M yields the entire set S.

The *fractal dimension* of a self-similar set S is

$$d_f = \frac{\log(N)}{\log(M)} \tag{10}$$

A **fractal** is a set whose fractal dimension exceeds it topological dimension. Clearly, there are many sets that are self-similar but not fractal. Since natural object are not the union of exact reduced copies of the entire set we need to extend the definition to include variations, i.e. **statistical self-similarity** .

Fractals: Koch Curve

The *Koch Curve* starts with a closed unit interval. At the first stage remove the middle third of the interval an replace it with two line segments of length 1/3 to form a virtual triangle. Repeat the procedure on every line segment.

The following function implements the Koch algorithm recursively.

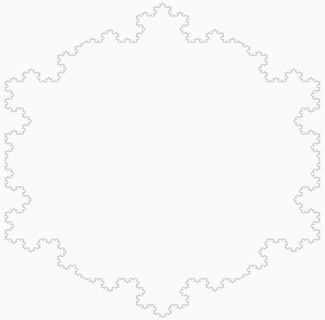
```
def koch(length, depth):
    if length == 0:
        forward(depth)
    else:
        koch(t, length-1, depth/3)
        go_left(60)
        koch(length-1, depth/3)
        go_right(120)
        koch(length-1, depth/3)
        go_left(60)
        koch(length-1, depth/3)
```

Fractals: Koch Curve

Another possibility to implement the Koch algorithm is to use pattern matching and replacement of certain pattern in a resulting string. This is shown in the following listing

```python
for i in xrange(steps):
    path = path.replace("F", "FLFRFLF")
```

Starting with a series of line segments that is closed we can generate constructs that have the appearance of snow flakes (c.f. Figure 2).



Fractals (continued)

the relationship between scale and detail

Felix Hausdorff (1868-1942) and Abram Besicovitch(1891-1970)

The Hausdorff-Besicovitch Dimension

A natural idea to calculate the fractal dimension is to cover the object. Thus we cover an object with boxes and then calculate how many boxes N are needed to cover the object entirely. This depends on the linear dimension of the box b. Plot the results on a log-log plot and determine the slope. The resulting value of the slope is the box dimension

$$d_B \approx d_f \tag{11}$$

Algorithm 1 Box Counting Algorithm

```
1: Generate a non-overlapping regular lattice with lattice constant $l$
2: for boxes of the lattice do
3:     Compute the number of objects within the box
4: end for
```

Fractals: Box Counting Algorithm

In the box counting algorithm (box-covering method) for networks [1] we calculate the minimum number of boxes N_B of linear dimension l that is needed to cover the object

$$N_B(l) \sim l^{d_B} \tag{12}$$

where d_B is the fractal dimension.

In the cluster-growing method on counts the average number objects M_c that are within a range l

$$< M_c(l) > \sim l^{d_B} \tag{13}$$

if $N \sim N_B(l) < M_c(l) >$, then both calculated the same fractal dimension. In scale free networks we have

$$< M_c(l) > \sim e^{l/l_0} \tag{14}$$

with some constant l_0 [1].

Fractals: Box Counting Algorithm (continued)

Algorithm 2 Random Sequential Box Covering Algorithm

```
 1: while all not all vertices are burned and assigned to their respective boxes do
 2:     Label all vertices as not burned $NB$
 3:     Select a vertex randomly at each step; this vertex serves as a seed.
 4:     Search the network by a distance $l$ from the seed
 5:     Burn all $NB$ vertices that are within the distance $l$ from the seed
 6:     Assign newly burned vertices to the new box.
 7:     if no newly burned vertex is found then
 8:         the box is discarded
 9:     end if
10: end while
```

Iterated Function Systems

Assume that we are given x_0 as a starting point and that we define two functions

$$F_0 = \frac{1}{3}x \tag{15}$$

$$F_1 = \frac{1}{3}(x-1)+1 \tag{16}$$

Consider the orbit of the initial value x_0 under the system of functions $\{F_0, F_1\}$ where at each step we choose to apply either F_0 or F_1 randomly with equal probability. The application of the system of functions is repeated for a fixed number of iterations. The end point of this iteration is then one point in a generated set of initial conditions. This basic algorithm is shown in Algorithm 3.

Iterated Function Systems

Algorithm 3 Iterated Function System

```
for $n\_points$ do
    Choose initial condition $x = x_0$
    for $n\_cycles$ do
        $r$ iid from $\{0, 1, ...n - 1\}$
        if $r == 0$ then
            $x = F_0(x)$
        else if $r == 1$ then
            $x = F_1(x)$
        else if $r == 2$ then
            ...
        else if $r == n - 1$ then
            $x = F_{n-1}(x)$
        end if
    end for
    Add new fixed point $x$ to set
end for
```

Iterated Function Systems

The above example can be easily generalized to higher dimensions.

As a further example we generate a fern. For this we we apply the following linear transformation

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} \tag{17}$$

following and using the parameters Bradley [2]:

$$F_0 \quad : \quad \begin{pmatrix} +0.00 & +0.00 \\ +0.00 & +0.16 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad 1\% \tag{18}$$

$$F_1 \quad : \quad \begin{pmatrix} +0.85 & +0.04 \\ -0.04 & +0.85 \end{pmatrix} \begin{pmatrix} 0 \\ 1.6 \end{pmatrix} \qquad 85\% \tag{19}$$

$$F_2 \quad : \quad \begin{pmatrix} +0.20 & -0.26 \\ +0.23 & +0.22 \end{pmatrix} \begin{pmatrix} 0 \\ 1.6 \end{pmatrix} \qquad 7\% \tag{20}$$

$$F_3 \quad : \quad \begin{pmatrix} -0.15 & +0.28 \\ +0.26 & +0.24 \end{pmatrix} \begin{pmatrix} 0 \\ 0.44 \end{pmatrix} \qquad 7\% \tag{21}$$

Iterated Function Systems

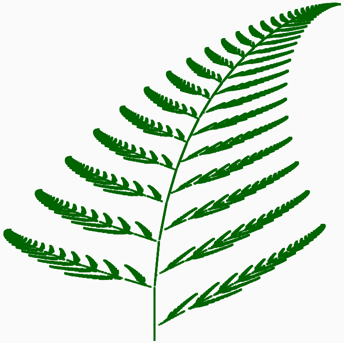


Figure 3: Fern fractal produced by an iterated function system. The parameters are taken from *http://www.stsci.edu/ lbradley/seminar/ifs.html* with 500 iteration steps. Shown is the result for 100000.

Percolation I

Example: [3]

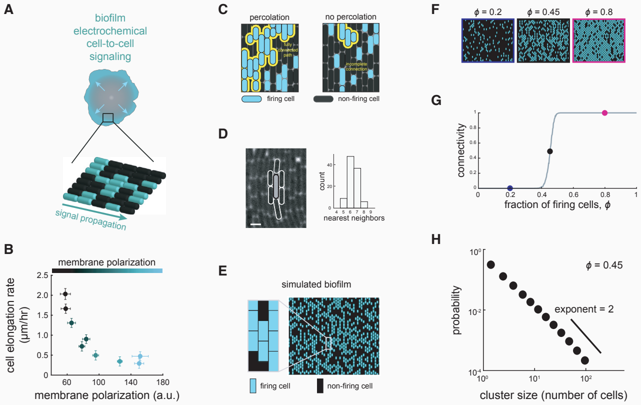


Figure 4: Image taken from Larkin et. al. [3] https://doi.org/10.1016/j.cels.2018.06.005

Percolation II

1 D percolation

Consider a lattice, which we take, for simplicity, as a two-dimensional square lattice. Each lattice site can be either occupied or unoccupied. A site is occupied with a probability $p \in [0, 1]$ and unoccupied with a probability $1 - p$. If p is small then only a tiny fraction of the sites is occupied and only small isolated patches of occupied sites that are close to each other exist. On the other hand, if p is large, then large patches of near lying occupied sites exist. We may at this point speculate that for p less than a certain probability p_c only finite clusters exist on the lattice. We define more precisely what we mean by patches or clusters by saying: A cluster is a collection of occupied sites connected by nearest-neighbour distances (see Figure 5). For p larger than or equal to p_c, there exists a large cluster (for an infinite lattice, i.e., in the thermodynamic limit) such that for an infinite lattice the fraction of sites belonging to the largest cluster is zero below p_c, and non zero above p_c. For $d = 1$ the situation is straight-forward: $p_c = 1$. For $d > 1$ the computation of p_c is non-trivial, and analytic results for the percolation threshold p_c are only available for certain dimensions or special lattices.

Percolation III

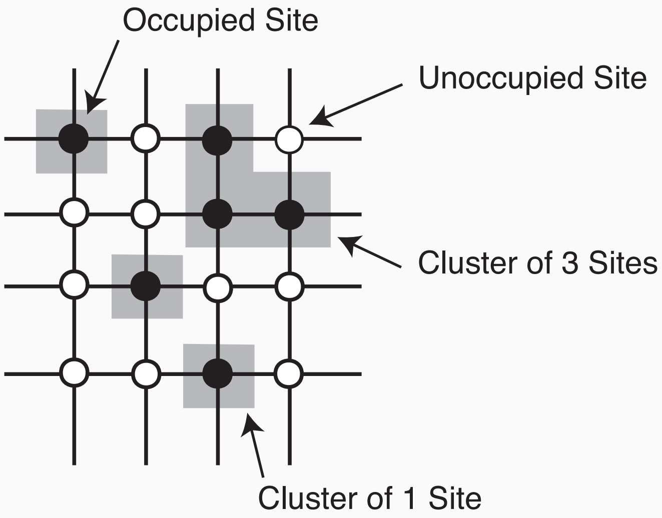
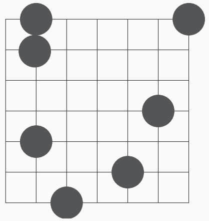
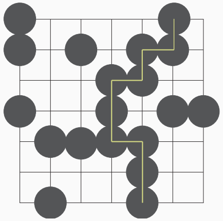
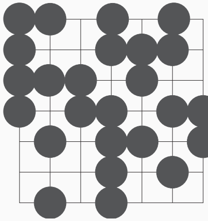


Figure 5: Definition of the percolation problem

Percolation IV


Occupation probability
$p = 0.1$


Occupation probability
$p = 0.59$


Occupation probability
$p > 0.59$

$$n_s(p, L) \tag{22}$$

$$p = \sum_{s<\infty} s n_s(p, L) \tag{23}$$

$$< s >= \sum_{s>0} s^2 n_s(p, L) / \sum_{s>0} s n_s(p, L) \tag{24}$$

Percolation V

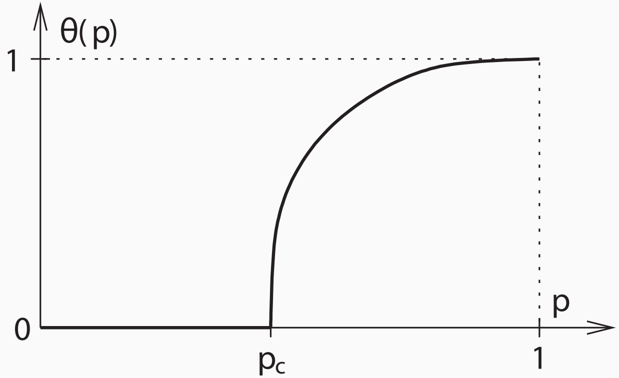


Figure 6: percolation probability

Percolation VI

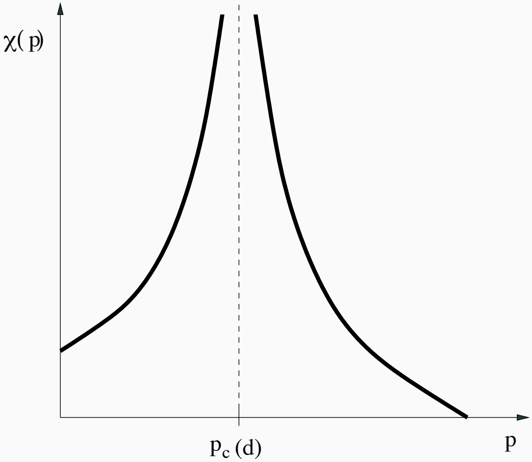


Figure 7: percolation susceptibility

Percolation VII

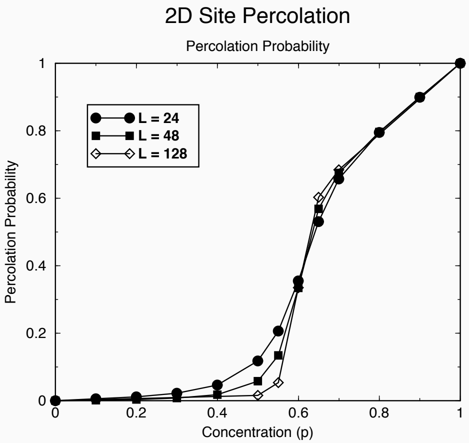


Figure 8: percolation probability

Percolation VIII

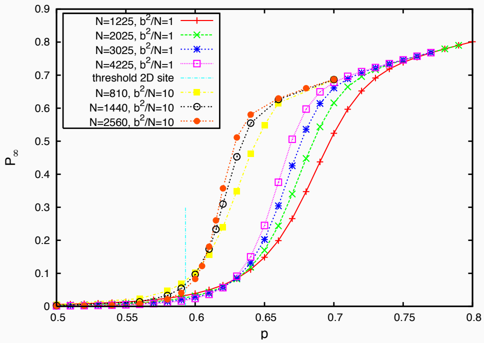


Figure 9: spanning probability

Percolation IX

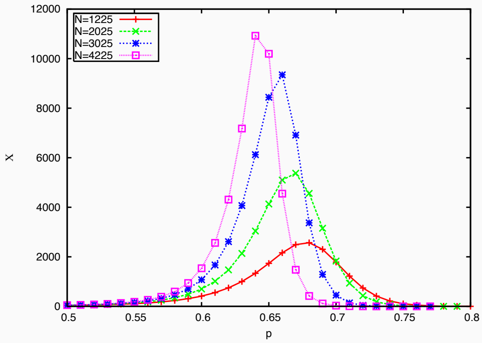


Figure 10: spanning susceptibility

Percolation X

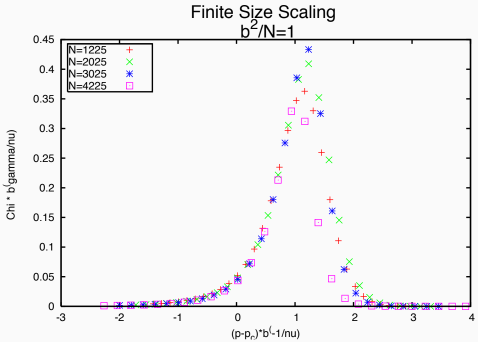


Figure 11: Scaling

Percolation I

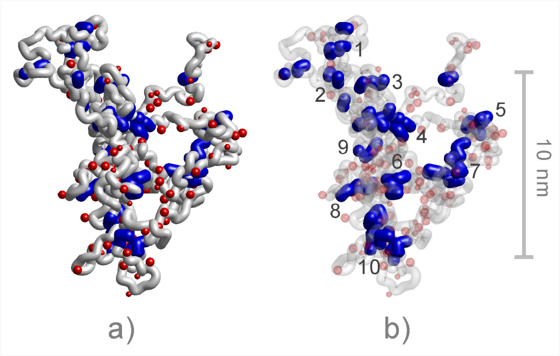


Figure 12: NPC cluster definition

Percolation II

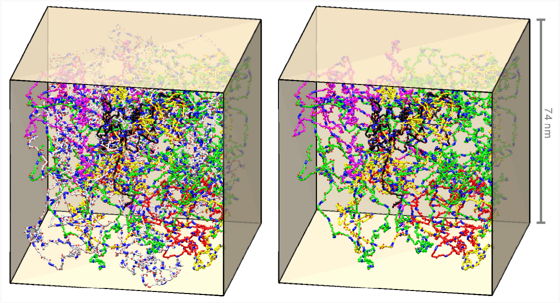


Figure 13: NPC cluster

Percolation I

Union Find Algorithm

Percolation

```
def union(x,y,labels):
    if (x == 0):
        labels[y] += 1
        return y
    if (y == 0):
        labels[x] += 1
        return x
    if (x < y):
        count = labels[y] + 1
        labels[y] = -x
        labels[x] += count
        return x
    elif (x == y):
        labels[x] += 1
        return x
    else:
        count = labels[x] + 1
        labels[x] = -y
        labels[y] += count
        return y

    return x
```

Code 1: union.py

Percolation

```python
def find(x,labels):
   pointer = labels[x]
   if (pointer < 0):
      while( pointer < 0 ):
         la = -pointer
         pointer = labels[la]
      labels[x] = -la
      x         = la
   return x
```

Code 2: union.py

Percolation

```python
# Find the first non-zero site
i = 0
found = False
while (i<L and not found):
   j = 0
   while (j<L and not found):
      if (lattice[i][j]==1):
         index = (i,j)
         found = True
      j += 1
   i += 1

if (not found):
   # All sites are empty: case p=0 or p very small
   return
# create the first label
i = index[0]
j = index[1]
lattice[i][j] = create(labels)
```

Code 3: union.py

Percolation

```
def find_clusters(lattice,L,labels):

   #continue from there on, but first row and column are specific
   while (i<L):
      while (j<L):
         left = j-1
         if (left < 0):
            left = L-1
         top = i-1
         if (top < 0):
            top = L-1
         if (lattice[i][j] == 1):
            if (i == 0):
               top_label = 0
            else:
               top_label  = find(lattice[top][j],labels)
            if (j == 0):
               left_label = 0
            else:
               left_label  = find(lattice[i][left],labels)
            if ( top_label == 0 and left_label == 0):
               # isolated site
               lattice[i][j] = create(labels)
            else:
               lattice[i][j] = union(left_label,top_label,labels)
         j += 1
      j = 0
      i += 1

   return
```

Percolation I

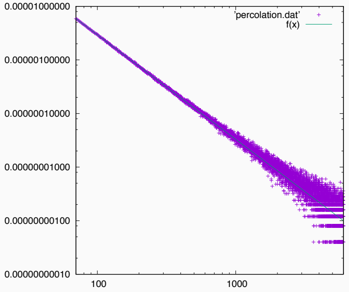


Figure 14: Number of clusters percolation: seed = 47115 max_steps = 10000 L = 500 p = 0.5927460

Fractal Flow I

The global objective of project B-2 arises from the question whether, and if yes what, statistical physics can contribute to a better understanding of porous materials. It is well known that important paradigms of statistical physics are closely related to phenomena arising during hydrodynamic flow in porous materials. When displacing a high viscosity fluid (such as oil) by a low viscosity fluid (such as water) in a porous medium one observes percolation clusters (more precisely invasion percolation) when capillary forces dominate. One observes diffusion limited aggregation clusters when viscous forces are dominant (see e.g. R. Lenormand and G. Daccord in "Random fluctuations and pattern growth" H.E. Stanley and N. Ostrowsky (eds.) NATO ASI Series E vol 157, Kluwer, Dordrecht 1988). A common feature of these observations is selfsimilarity. Self- similar or fractal phenomena have recently attracted much attention from statistical physicists (see e.g. "Fractals in Physics", L. Pietronero and E. Tosatti (eds.), Elsevier Amsterdam 1986), and much progress has been made in their study, as well as in the study of percolation and diffusion limited aggregation. The application of these results to structure and dynamics of porous media promises therefore to be of great interest.

Lung: Gas Diffusion through the Fractal Landscape of the Lung

Fractal Flow II

- Diffusion on Fractals: Three-dimensional percolation threshold.
  Particles are assumed to diffuse randomly in a porous medium. This medium is a simple cubic lattice consisting of pores (empty sites) and rock (occupied sites). Diffusion can take place only in the empty space, and empty and occupied sites are distributed randomly. Increasing the fraction of empty sites, we change from an impermeable to a permeable medium at a percolation threshold of 31 percent of empty sites. Right at this threshold, the pore space is fractal, and the diffusion becomes anomalous, i.e. the squared distance increases weaker than linearly with time.
  - M.J.Velgakis, Physica A 159, 167 (1989)
- Hydrodynamical Cellular automata
  The cellular automata approximation of hydrodynamics (Frisch, Hasslacher, and Pomeau, 1986) is presently restricted to two dimensions. Basically, molecules fly with unit velocity along the lattice directions of a triangular lattice, scattering on each other according to the law of momentum conservation. Brosa's implementation of this algorithm on the HLRZ Cray supercomputer is an order of magnitude faster than the diffusion simulation mentioned above. Duarte and Brosa simulated the flow around a cylinder between two plates and found good agreement with old experimental data. The same algorithm is presently applied to many such obstacles as a model for a porous medium. 2000 * 666 lattices can

Fractal Flow III

be simulated within a minute on a Cray YMP; we average over 8 such lattices with the same porous structure but a different initial velocity distribution.

- U.Brosa and D.Stauffer, J.Statistical Physics 57, 399 (1989)
- J.A.M.S.Duarte+U.Brosa, J.Statistical Physics 59, 501 (1990)
- D.Stauffer, invited talk A4a, EPS Condensed Matter Conference, Lisbon, April 1990

Excercises

Excercises I

Exercise 1: **Van der Pol Oscillator**

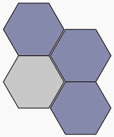
Consider the Van der Pol Oscillator describing an oscillating system with non-linear damping and self regulation. The aim is to control the oscillation such that the system stays in a mean position.

$$\frac{d^2x}{dt^2} - u(1-x^2)\frac{dx}{dt} + x = 0 \tag{25}$$

where u is the control parameter. Produce phase portraits for a set of values of u.

Exercise 2: **Snowflake Cellular Automaton**

Construct a crystallization model for a snowflake [5]. For this, consider a hexagonal unit on which other hexagonal units can crystalize



Excercises II

Exercise 3: **Snowflake with a Multiple Reduction Copy Machine Algorithm**
Use the Multiple Reduction Copy Machine Algorithms to construct a snowflake.

Exercise 4: **Percolation Threshold**
Confirm the site-percolation $p_c = 0.59274601$ threshold [4] for the square lattice.

Exercise 5: **Kinetic Percolation**
For some problems we are interested not so much in all of the clusters in a given percolation configuration as in just the largest. For example, right at the percolation threshold p_c the largest cluster is a fractal and we would like to know more about its properties. We can generate a large percolation cluster by a sort of kinetic growth process from which we can also learn the growth law, i.e., how the radius of gyration grows with time. The algorithm is as follows. Start with a single site and make a list of the nearest neighbors. Occupy each of the sites in this list with probability p. Make a new nearest-neighbor list (be careful, the ones you visited before but decided not to occupy are marked as unoccupied). Program the algorithm and calculate the radius of gyration as a function of p and time.

Excercises III

Exercise 6: **Continuum percolation**

There is also a continuum version of the percolation problem. Imagine a square. You have disks at your disposal which you throw randomly onto the square. For this you generate an x coordinate and a y coordinate using a random number generator. Place the center of the disk at this coordinate with probability p. If you can cross from one side of the square to the opposite site via overlapping disks then a spanning cluster is found. Print out your trials and decide visually if there is a spanning cluster or not.

Bibliography

Bibliography I

References

[1] J. S. Kim, K.-I. Goh, B. Kahng, and D. Kim. A box-covering algorithm for fractal scaling in scale-free networks. *Chaos*, 17(2), 2007.

[2] www.stsci.edu/ lbradley/seminar/ifs.html, 2010.

[3] Joseph W. Larkin , Xiaoling Zhai , Kaito Kikuchi , Samuel E. Redford , Arthur Prindle , Jintao Liu , Sacha Greenfield , Aleksandra M. Walczak , Jordi Garcia-Ojalvo , Andrew Mugler , Gürol M. Süel. Signal percolation within a bacterial community. *Cell Systems https://doi.org/10.1016/j.cels.2018.06.005*, (II):137–145, 2018.

[4] Jesper Lykke Jacobsen. High-precision percolation thresholds and Potts-model critical manifolds from graph polynomials. *Journal of Physics A Mathematical General*, 47(13):135001, April 2014.

[5] S. Wolfram. *Nature*, 311, 1984.

Index

Index I