Spezialisierungsbericht

Computersimulation aktiver Random-Walks auf perkolierendem Cluster

Sebastian Steinhäuser

Betreut durch Thomas Voigtmann und Julian Reichert

Contents

1 Perkolation

1.1 Was ist Perkolation?

In diesem Bericht geht es um Perkolation und vor allem um den Random-Walk (mathematisches Modell für eine Bewegung, bei der die einzelnen Schritte zufällig erfolgen) auf einem perkolierenden Cluster in einem Quadratgitter in zwei Raumdimensionen. In diesem Fall gibt es zwei geläufige Perkolationsmodelle, Knotenperkolation (engl. 'site-percolation') und Kantenperkolation (engl. 'bond-percolation'), hier wird immer über Knotenperkolation gesprochen. In folgender Abbildung wird der Unterschied zwischen Knoten- und Kantenperkolation bildlich dargestellt [1].

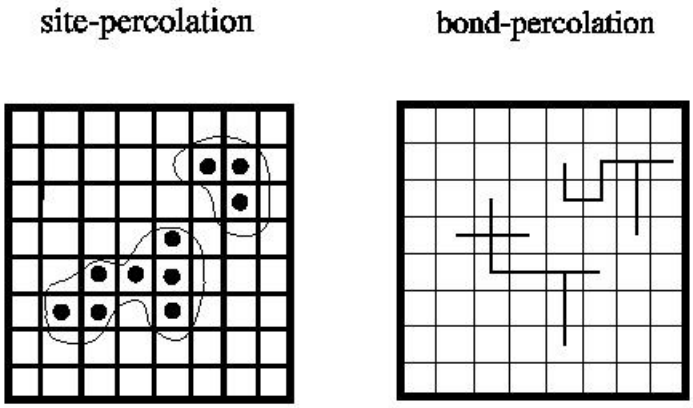

Figure 1: Knoten- und Kantenperkolation

Das Quadratgitter wird zufällig mit einer Wahrscheinlichkeit $1 - p$ geblockt. Es ist Konvention, die Wahrscheinlichkeit, dass ein Platz frei ist, mit p zu bezeichnen, diese Konvention möchte ich hier erhalten. Geblockte Plätze können nicht besetzt werden. Man kann sich geblockte Plätze wie Wände eines Labyrinths vorstellen, durch die der Random-Walker später nicht hindurch kommt.
Nun definiert man einen Cluster als eine Gruppe benachbarter Quadrate die begehbar sind, also nicht geblockt. Man nennt einen Cluster perkolierend, wenn sich der Cluster von einer Kante zur gegenüberliegenden Kante erstreckt, so dass zum Beispiel Wasser durch das Gitter fließen könnte, ähnlich wie es durch eine Kaffeemaschiene (engl. 'percolator') perkoliert/sickert. Vereinfacht gesagt kann ein Random-Walker auf dem perkolierenden Cluster sich unendlich ausbreiten und ist nicht 'eingesperrt'. Man kann sich nun leicht Vorstellen, dass wenn kaum Felder geblockt sind (also p nahe 1) sich auf dem (unendlich großen) Quadratgitter stets ein perkolierender Cluster finden lässt und wenn kaum Felder begehbar sind (also p nahe 0) sich nie ein perkolierender Cluster finden lässt. Es gibt dazwischen eine sogenannten kritischen Wert $p = p_c \approx 0.5928$ [2]

[1] https://de.wikipedia.org/wiki/Perkolationstheorie, 21.10.2019
[2] D. Stauffer, Perkolationstheorie (Taylor & Francis, London, 1994)

bis zu welchem das unendliche Quadratgitter perkoliert, auch Perkolationsschwelle oder englisch percolation-threshold genannt. Historisch geht die Perkolationstheorie (engl. 'percolation theory') auf Paul Flory und Walter H. Stockmayer zurück, die sie während des Zweiten Weltkriegs entwickelten, um Polymerisationsprozesse zu beschreiben. Der Polymerisationsprozess kommt durch das Aneinanderreihen von Molekülen zustande, die dadurch Makromoleküle bilden. Der Verbund solcher Makromoleküle führt zu einem Netzwerk von Verbindungen, die sich durch das ganze System ziehen können.[1]
Üblicherweise wird der Beginn der Perkolationstheorie mit einer Publikation von Broadbent und Hammersley aus dem Jahre 1957 in Verbindung gebracht, in welcher der Name eingeführt wurde, und in welcher sie mit den oben erläuterten geometrischen und wahrscheinlichkeitstheoretischen Konzepten mathematischer behandelt wurde. Hammersley erwähnt in seiner persönlichen Geschichte der Perkolation in 'Percolation Structures and Processes', dass die (damals) neuen Computer, die für die Wissenschaftler dieser Zeit zugänglich wurden, einer der Gründe für die Entwicklung der Perkolationstheorie waren: Hier handelte es sich um Probleme, bei denen die Computer nützlich werden konnten.[2]

1.2 Perkolation auf dem Computer

Es ist selbstverständlich nicht möglich ein unendlich großes Quadratgitter mit zufällig geblockten und freien Gitterplätzen auf dem Computer zu erzeugen, daher wird hier stets ein endliches Quadratgitter(auf dem Computer in Form einer Matrix/Array gespeichert) und mit periodischen Randbedingungen ausgestattet. Um perkolierende Cluster dieser zufällig erzeugten Matrix zu finden nutzt man den sogenannten Hoshen-Kopelmann-Algorithmus.[3] Dieser Algorithmus lässt sich am besten an einem Beipiel erklären. Im nachfolgenden Bild sind freie Felder grau und geblockte Felder weiß, graue Felder die an einer Kante verbunden sind (wo der Random-Walker also von einem Feld ins andere kann) werden mit dem selben Label versehen.

[1] Zitiert aus: https://de.wikipedia.org/wiki/Perkolationstheorie, 21.10.2019
[2] Zitiert aus: D. Stauffer, Perkolationstheorie
[3] https://www.ocf.berkeley.edu/ fricke/projects/hoshenkopelman/hoshenkopelman.html, 21.10.2019

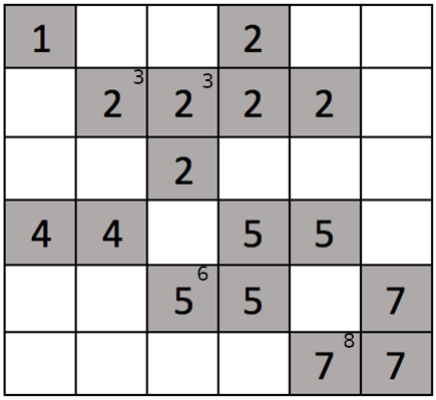

Figure 2: Veranschaulichung des Hoshen-Kopelmann-Algorithmus

1.2.1 Implementierung des Hoshen-Kopelmann-Algorithmus

Die Idee, wie man dieses anschauliche Nummerieren auf den Computer überträgt ist es, die Matrix von (zum Beispiel) oben links beginnend abzugehen. Stößt man auf ein begehbares Feld (also einen Eintrag 1 in der Matrix) vergleicht man mit dem Label über und links von diesem Feld, haben beide das selbe Label, so wird dieses auch in das Feld, welches aktuell bearbeitet wird, eingetragen; haben die beiden Felder ein unterschiedliches Label, so merkt man sich in einer Liste, dass diese beiden Label jetzt verbunden sind durch das aktuell zu bearbeitende Feld und später zusammengefügt werden müssen. Sind die Felder oben und links von dem zu bearbeitenden Feld geblockt, so labelt man das Feld mit dem nächst größeren noch nicht benutzten Label. Ist man in diesem Verfahren durch die Matrix durch, so fügt man die verlinkten Labels zusammen; es gilt als Konvention das kleinst mögliche Label zu verwenden. Zum Beispiel verlinkt das vierte Feld der zweiten Reihe in obiger Abbildung 2 die Labels 2 und 3 oder das vierte Feld der fünften Reihe die Labels 5 und 6. In python ist dieser Algorithmus für das Labeling (welcher auch in der Bildverarbeitung genutzt wird) bereits in der *scipy.ndimage* Bibliothek als *measurements* vorhanden. Nachfolgende Grafik zeigt einen Pseudocode[3] des Hoshen-Kopelmann-Algorithmus, man kann schön die Unterteilung in scannen des Clusters, verlinken von Labeln (union) und finden der Equivalenzklasse (des Labels) (find) sehen.

[3]https://www.ocf.berkeley.edu/ fricke/projects/hoshenkopelman/hoshenkopelman.html

```
Raster Scan and Labeling on the Grid
largest_label = 0;
for x in 0 to n_columns {
    for y in 0 to n_rows {
        if occupied[x,y] then
            left = occupied[x-1,y];
            above = occupied[x,y-1];
            if (left == 0) and (above == 0) then /* Neither a label above nor to the left. */
                largest_label = largest_label + 1; /* Make a new, as-yet-unused cluster label. */
                label[x,y] = largest_label;
            else if (left != 0) and (above == 0) then /* One neighbor, to the left. */
                label[x,y] = find(left);
            else if (left == 0) and (above != 0) then /* One neighbor, above. */
                label[x,y] = find(above);
            else /* Neighbors BOTH to the left and above. */
                union(left,above); /* Link the left and above clusters. */
                label[x,y] = find(left);
    }
}
```

```
Union
void union(int x, int y)  {
    labels[find(x)] = find(y);
}
```

```
Find
int find(int x)  {
    int y = x;
    while (labels[y] != y)
        y = labels[y];
    while (labels[x] != x)  {
        int z = labels[x];
        labels[x] = y;
        x = z;
    }
return y;
}
```

Figure 3: Pseudocode des Hoshen-Kopelmann-Algorithmus

1.2.2 Finden von gerichtet perkolierender Clustern

Geht ein Cluster von einem Rand zum gegenüberliegenden Rand, also tritt an diesen beiden Kanten das selbe Label auf so perkoliert dieser Cluster in der endlichen Matrix. Durch die periodischen Randbedingungen muss das Label am gegenüberliegenden Rand in der gleichen 'Höhe' auftreten. Es wird also einfach der linke Rand der Matrix abgegangen und das jeweilige Label mit dem Label an dem rechten Rand verglichen. Analog verfährt man mit dem oberen Rand. Diese Möglichkeit der Perkolation ist durch die periodischen Randbedingungen allerdings nicht die einzige, wie das nächste Unterunterkapitel zeigt.

1.2.3 Diagonal perkolierende Cluster

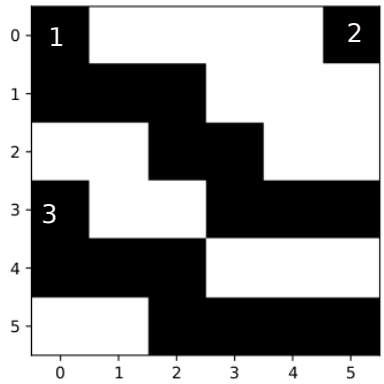

Figure 4: Beispiel eines diagonal perkolierenden Clusters

Neben den in §1.2.2 beschriebenen gerichtet perkolierenden Clustern existieren noch Custer die diagonal über die periodischen Randbedingungen perkolieren. Ein Beispiel ist in Abbildung 4 zu sehen, aus dem ersten Cluster gelangt man über periodische randbedngungen in den dritten Cluster, geht man aus der Position 5,5 nach unten gelangt man in den dritten Cluster in der oberen rechten Ecke, mit einem Schritt nach rechts ist man wieder in Cluster 1.
Das Finden dieser Cluster stellt sich als schwieriger heraus, als das Finden der gerichtet perkolierenden Cluster, da man nicht einfach nur die Labels über die Periodischen Randbedinungen verlinken darf wie das nachfolgende Beispiel zeigt.

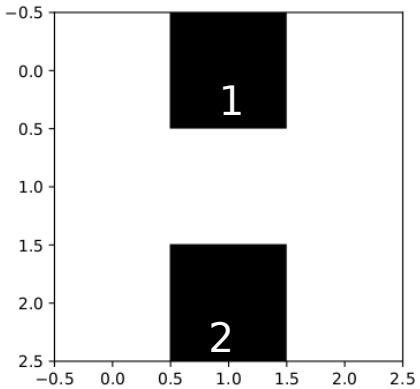

Figure 5: Es existiert offensichtlich kein perkolierender Cluster.

Würde man einfach das Labeln über periodische Randbedingung machen, würden beide Blöcke entsprechend Label 1 erhalten und nach dem Kriterium aus §1.2.2 hätte man einen perkolierenden Cluster gefunden. Dies ist ganz offensichtlich falsch, da es sich um einen Cluster endlicher Größe handelt.
Der (scheinbar) korrekte Algorithmus den perkolierenden Cluster zu finden ist es, Cluster die sich über periodische Randbedinungen Berühren in einer Verlinkungsliste zu Speichern, zu den Labels muss die Richtung gespeichert werden. Man wählt die Konvention: links entspricht $(-1, 0)$ rechts $(+1, 0)$ unten $(0, -1)$ und oben entsprechend $(0, +1)$. Zu dem 'Weg' in der Verlinkungsliste muss die Summe der Richtungen gebildet werden, ist diese ungleich $(0, 0)$ so perkolieren die Cluster die verlinkt sind. Man muss (rekursiv) eine Suchfunktion implementieren die alle zu einem gegebenen Label verlinkten Labels sucht. Auf alle Verlinkten Labels wird erneut die Suchfunktion angewendet, und so weiter, wird das Ausgangslabel gefunden so müssen die Richtungen addiert werden und mit $(0, 0)$ verglichen werden.
Am Beispiel der in Abbildung 4 gezeigten Matrix bedeuted die $1 \to 3$ hat Richtung $(+1, 0)$, $3 \to 2$ bedeutet Richtung $(0, -1)$ und $2 \to 1$ wieder $(+1, 0)$. Insgesamt hat dieser Weg also die Richtung $(+2, -1) \neq (0, 0)$ und wird von dem Algorithmus gefunden.
Im Gegensatz dazu zeigt die nächste Abbildung eine Matrix, mit einem $(0, 0)$ Loop, man "läuft im Kreis", ohne das wirklich eine Box durchschritten wird. Diese Matrix wird auch nicht als perkolierend gefunden, eignet sich aber sehr gut zum debuggen.

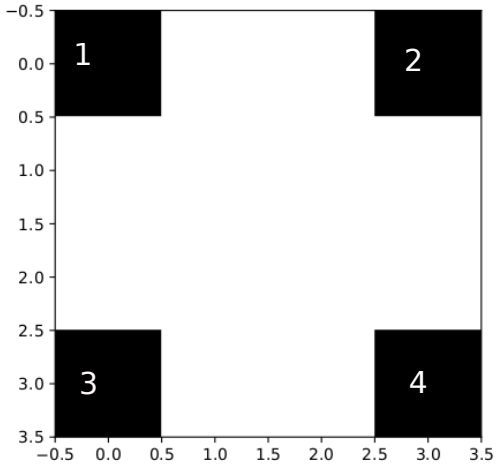

Figure 6: Es existiert offensichtlich kein perkolierender Cluster.

Das einzige Problem ist, dass dieser Algorithmus schlecht mit der Boxgröße skaliert, das die Anzahl der Labels quadratisch mit der Boxgröße skaliert und die Zahl der Knoten nochmals quadratisch mit der Boxgröße. Der Algorithmus skaliert also ungefähr mit L^4, wobei L die Boxlänge bezeichnet. Unten sieht man den oben beschriebenen Teil zur

Suche diagonal perkolierender Cluster als python Code, welcher auch in den nachfolgenden Simulationen genutzt worden ist.

```
# if we did not yet find a percolating cluster inside the box,
# we could have percolation across periodic boundary images
# to find them, go through all cross-boundary links and try
# to find a closed loop among them
# if such a loop exists, *and* this loop circles around such that
# it comes back to its start in a periodic image that is offset
# from the original by at least one in one direction,
# we have a percolating cluster (else, just an isolated loop that
# happens to lie across a box boundary); to figure this out, we
# store with the cross-boundary links also a sense of direction
sadd=lambda xs,ys: tuple(x + y for x, y in zip(xs, ys))
def seekpath (path,pathdir,l,lstart,looplist):
  path = list(path)
  mylooplist = list(looplist)
  path.append(l)
  for n in range(len(mylooplist)):
    (l1,l2,linkdir,inpath)=mylooplist[n]
    if not inpath:
      if l==l1: # and not (len(path)>1 and l2==path[-2]):
        mylooplist[n]=(l1,l2,linkdir,1)
        if l2==lstart:
          #print "found loop: ",path,sadd(pathdir,linkdir)
          if not sadd(pathdir,linkdir)==(0,0):
            return 1
        else:
          return seekpath (path,sadd(pathdir,linkdir),l2,lstart,mylooplist)
  return 0
for l in range(len(loopstarts)):
  if loopstarts[l]:
    percolates[l]=seekpath ([],(0,0),l,l,looplist)
```

Figure 7: Python-Code zur Suche diagonal perkolierender Cluster.

2 Random-Walk in ungeordneten Medien

2.1 "Normeler" Random-Walk

Der (diskrete) Random-Walk auf dem freien Quadratgitter mit Hüpfwahrscheinlichkeit $\frac{1}{d}$ auf jeden der d nächsten Nachbarn, wobei d die Dimension des Gitter bezeichnet, unterliegt bekanntlich normaler Diffusion. Das heißt, dass das mittlere quadratische Verschiebung (engl. mean squared displacement), welches im weiteren mit msd abgekürzt wird, linear mit der Anzahl an Schritten anwächst. Im kontinuierlichen Fall gilt die bekannte Formel:

$$msd(t) \equiv \langle \delta r^2(t) \rangle = 2dDt \ ,$$

wobei (wie oben) d die Dimension und D der Diffusionskoeffizient sind, und hier gilt $D = 1$.

2.2 Random-Walk auf dem Perkolationsgitter

Das Perkolationsgitter bei dem nur ein zufälliger Bruchteil der Gitterplätze begehbar sind und die anderen geblockt sind ist ein besonders einfaches Modell für ein ungeordnetes Medium. Eine spannende Frage - die in den 70er und 80er Jahren des vergangenen Jahrhunderts durch eine Vielzahl von Computersimulationen geklärt wurde - ist, wie sich ein Random-Walker auf so einem unregelmäßigen Perkolationgitter für verschiedene p (= Wahrscheinlichkeit das ein Gitterplatz begehbar ist) verhält. Speziell geht man, motiviert durch die fraktale Struktur der Cluster, von einem Potenzgesetz:

$$msd(t) \sim t^{2\nu}, \quad \nu = 1/d_w$$

aus, wobei ν als Diffusionsexponent bezeichnet wird und d_w als 'walk dimension'. Der im Feld der weichen Materie bekannte französische Physiker Pierre-Gilles de Gennes nannte diese Fragestellung 1976 die 'Ameise im Labyrinth'.[2]
Zuerst muss man die Hüpfwahrscheinlichkeiten festlegen, also ob $\frac{1}{f.n.N.}$ ($f.n.N.$ für freie nächste Nachbarn) oder wie bei dem freien Random-Walk $\frac{1}{d}$ und Züge auf ein geblocktes Feld werden zurückgesetzt. Für lange Zeiten sind beide Variaten äquivalent ('blind' vs 'myopic' ants)[3].
Nun überlegt man sich leicht die beiden Extremfälle p nahe 1 und p nahe 0. Im ersten Fall ($p \approx 1$) erwartet man sofort, dass $\nu \to 1/2$, da es kaum 'Hindernisse' gibt und der Random-Walk nahezu ungestört laufen kann. Anders herum, bei p nahe 0 erwartet man, dass es keine perkolierenden Cluster gibt, somit ist der Random-Walker 'eingesperrt' in sogenannte 'Taschen', damit das msd beschränkt und somit $\nu \to 0$ für lange Zeiten. Besonders spannend ist dieses Problem also in der Nähe der Perkolationsschwelle $p = p_c$, hier findet sich ein Diffusionskoeffizient, der weder 1/2 noch 0 ist.

[2] D. Stauffer, Perkolationstheorie, Kapitel 1.4
[3] S. Havlin and D. Ben-Avraham, Adv. Phys.36, 695 (1987), Appendix A.3

Gefen, Aharony und Alexander nannten diesen Effekt, dass das msd weder normal (linear, $\nu = 1/2$) mit t anwächst noch beschränkt ist, sondern für lange Zeiten asymptotisch einem Potenzgesetz mit $\nu \neq 1/2$ folgt, 'anormale Diffusion'.[6] Diese anormale Diffusion wird, im Falle des unendlich großen Gitters, nur für $p = p_c$ gefunden, den für $p < p_c$ existiert kein perkolierender cluster, das msd ist also beschränkt. Ist $p > p_c$ so gilt für $t \to \infty$ immer $msd \sim t$, aber es gibt ein 'Fenster' mit $msd \sim t^{2\nu}$, wobei $\nu < 1/2$ ist.

2.3 'all-cluster-' und 'percolating-cluster average'

Es gibt im Allgemeinen zwei Varianten wie man ν_{pc} auffassen kann, man mittelt über zufällige Startpunkte auf dem zufällig geblockten Quadratgitter oder man mittelt über Startwerte die nur auf dem perkolierenden Cluster liegen ('all-cluster average' und 'percolating-cluster average'). Es ist zu erwarten, dass der Exponent bei dem 'all-cluster average' unter dem Exponenten des 'percolating-cluster average' liegt, da beim 'all-cluster average' auch in sogenannten Taschen gestartet wird in denen das msd beschränkt ist. Die aktuell bekannten 'Literaturwerte' sind $d_w^{a.c.} \approx 3.036$ und $d_w^{p.c.} \approx 2.878$, woraus sich $\nu_{pc}^{a.c.} \approx 0.329$ und $\nu_{pc}^{p.c.} \approx 0.347$ ergeben.[4] In dieser Spezialisierung wurde versucht diese Werte durch Monte-Carlo-Simulation (MC) zu reproduzieren. Es wurde eine Gitterlänge $L = 1000$ verwendet und die Perkolationsschwelle als $p_c = 0.592\,746$ angenommen. Es wurden zu beiden Mittelungsvarianten über 100 Läufe auf je 100 zufälligen Matrizen gemittelt, wobei nicht auf diagonale Perkolation getestet worden ist (aus Laufzeitgründen). Die Läufe waren je 10^6 MC-Schritte lang und wurden auf einem (10er-) logarithmischen Grid gespeichert mit 48 Speicherstellen.

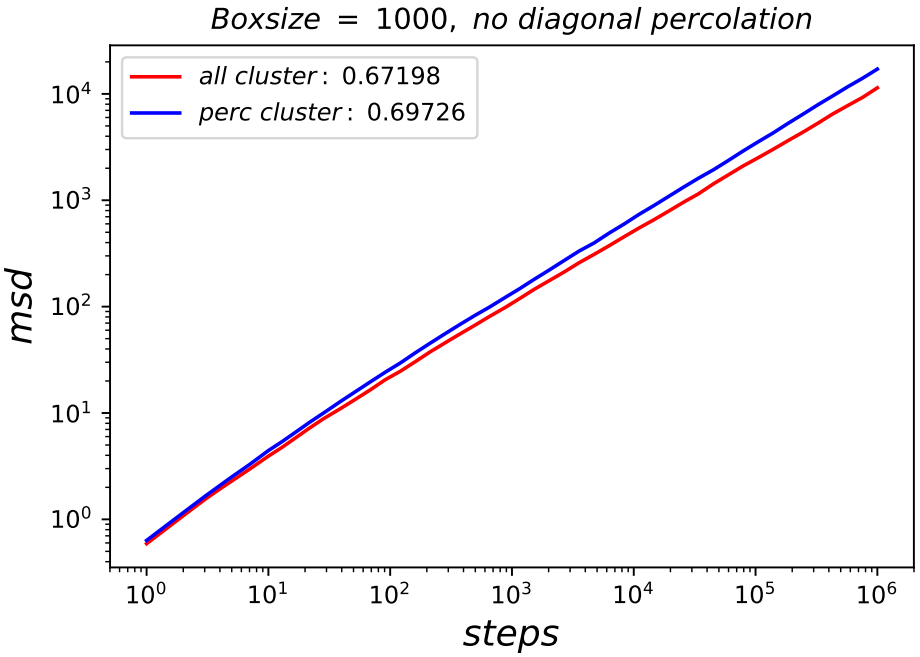

Figure 8: Ergebnisse meiner python3 MC-Simulation, der 'all-cluster average' ist in rot, der 'percolating-cluster average' in blau, es wurden keine diagonal perkolierenden Cluster in die Statistik aufgenommen.

[4] P. Grassberger, Phys. A 262, 251 (1999)

Die Exponenten wurden über einen *scipy.optimize.curve_fit* als $2\nu_{pc}^{a.c.} \approx 0.6712$ und $2\nu_{pc}^{a.c.} \approx 0.6973$ bestimmt. Die Methode den Exponenten durch einen fit zu bestimmen hat sich als deutlich stabiler herausgestellt als das bestimmen der Ableitung durch eine 'central-difference' oder 'forward-difference' Methode, welche unter sehr hohen numerischen Fehler leiden, da zwei sehr große Zahlen von einander abgezogen werden, deren Differenz klein ist. Dieses Problem ist bekannt als 'catastrophic cancellation'. Nimmt man Werte die weiter auseinander liegen (also ein größeres h), so wird die Ableitung ebenfalls sehr ungenau, da der Fehler quadratisch ('central-difference') beziehungsweise linear ('forward-difference') in Abstand h wächst. Ein lokaler Mittelwert hilft etwas, die 'Zacken' in der numerischen Ableitung zu glätten, ist aber durch das logarithmische Grid mit Vorsicht zu genießen, da spätere Zeiten stärker gewichtet werden. Im nachfolgenden Plot ist die numerische Ableitung mit dem Fit-Parameter verglichen und ebenfalls die gefittete Kurve (in schwarz) an das 'all-cluster' mean-squared-displacement angelegt. Man erkennt, dass die numerische Ableitung, welche ähnlich zur 'central difference', aber auf einem logarithmischen Grid ist, und Fit übereinstimmen. Auf oben beschriebenes Glätten der Ableitung wurde verzichtet.

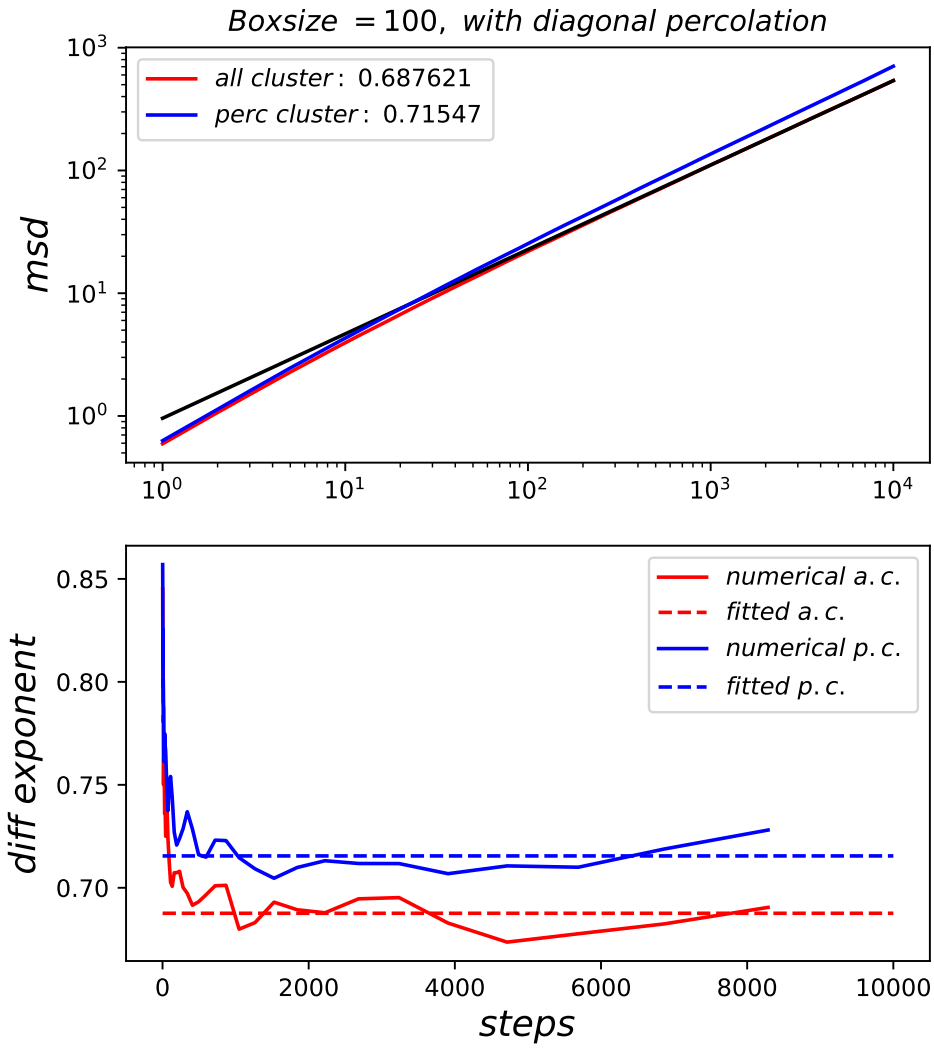

Figure 9: Ergebnisse meiner python3 MC-Simulation, der 'all-cluster average' ist in rot, der 'percolating-cluster average' in blau, es wurden keine diagonal perkolierenden Cluster in die Statistik aufgenommen.

2.4 Vergleich der Statistik mit und ohne diagonale Perkolation

In kleineren Simulationsboxen treten finite-size Effekte auf. Die Diffusionsexponenten sind weiter von den Literaturwerten für eine unendliche Box entfernt; dennoch sind (bei gleicher Boxgröße) wie nachfolgender Plot zeigt die Exponenten mit diagonal perkolierenden Clustern näher am Literaturwerten für eine unendliche Box. Es sind bei einer Boxgröße $L = 100$ ca. $18,6\%$ der perkolierenden Cluster diagonal perkolierend. Größere

Boxen sind aktuell durch zu hohe Laufzeit des Clustersuch-Algorithmus nicht (im Rahmen einer Spezialisierung) realisierbar.

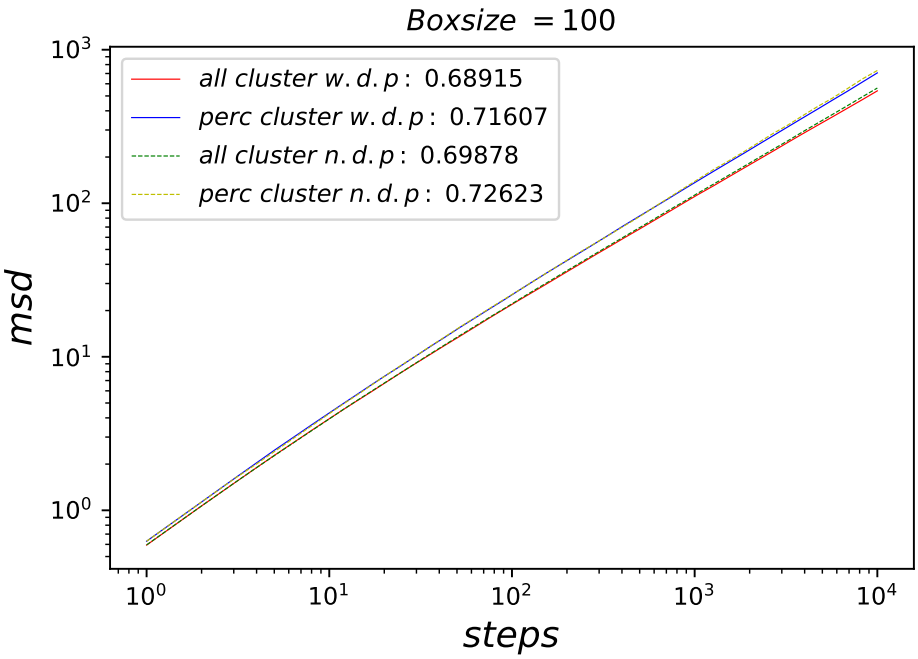

Figure 10: Vergleich von Statistik mit (w.d.p.) und ohne (n.d.p.) diagonal perkolierende Cluster. Gesamplet über 1000 Läufe auf je 500 Matrizen.

Man sieht vorallem an den Werten, das der Literaturwert von $2\nu_{pc}^{p.c.} \approx 0.694$ nun unterschätzt wird (bezogen auf den 'percolating-cluster average'). Zudem ist der Wert bei der Box mit Länge $L = 1000$ ohne diagonal perkolierende Cluster näher am Literaturwert. Man sollte also lieber auf das finden der diagonal perkolierenden Cluster verzichten und größere Boxen wählen, denn der 'finite-size' Fehler scheint größer, als der Fehler, den das Auslassen der diagonal perkolierenden Cluster verursacht.

2.5 Random-Walk auf diagonal perkolierenden Clustern

Um den Einfluss der diagonal perkolierenden Cluster besser zu verstehen habe ich eine Simulation von 1000 Läufen auf je 500 diagonal perkolierenden Clustern angefertigt ('percolating-cluster average'). Die Boxgröße beträgt $L = 100$. Es wurde $\nu_{pc}^{p.c.} \approx 0.681$ gefunden, dieser Wert unterschreitet den Literaturwert von $\nu_{pc}^{p.c.} \approx 0.694$. Dies ist leicht einzusehen, den diagonale Läufe haben ein geringeres msd, da ein Schritt nach (z.B.) rechts gefolgt von einem Schritt nach (z.B.) oben ein msd von 2 ergibt wohingegen zwei Schritte in eine Richtung ein msd von 4 ergeben. Die diagonal perkolierenden Cluster führen also auch dazu, dass das msd geringer wird, so lässt sich erklären, das auch bei der Simulation der $L = 1000$ Box in §2.3 beide Exponenten überschätzt werden.

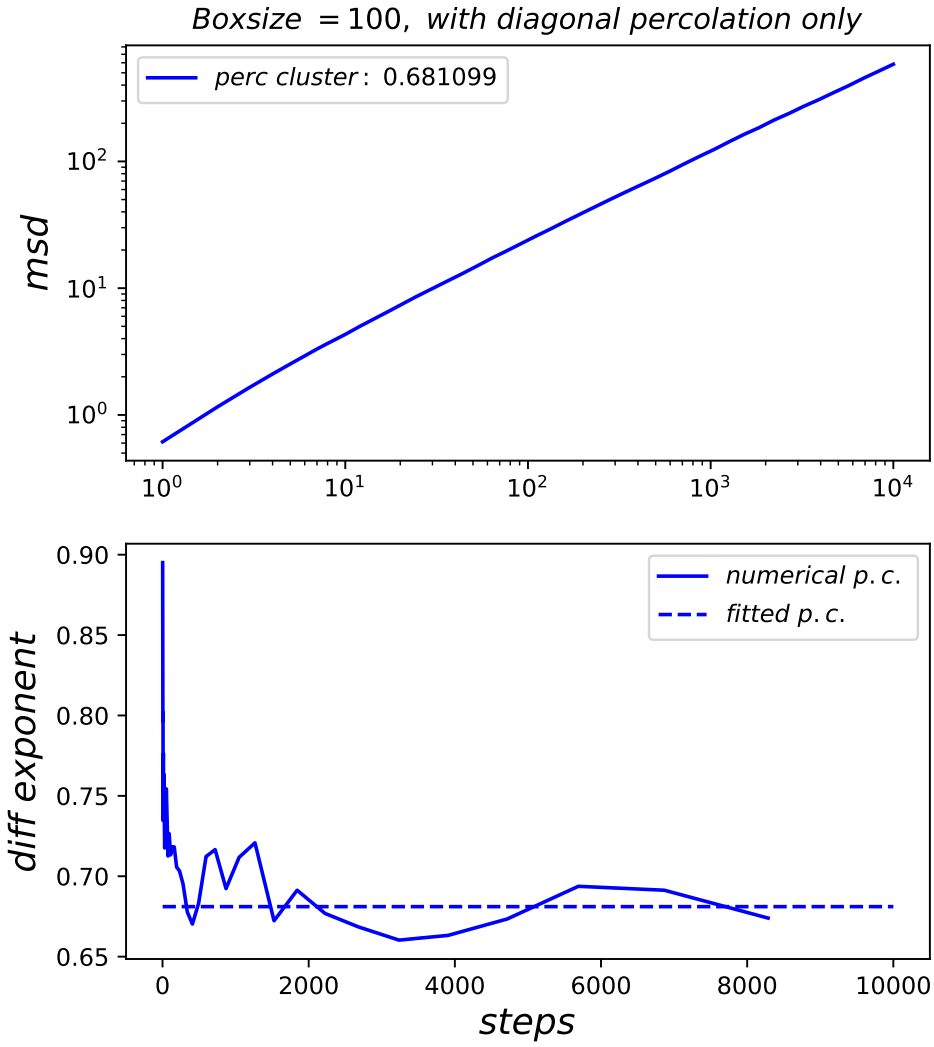

Figure 11: Vergleich von Statistik mit und ohne diagonal perkolierende Cluster. Gesamplet über 1000 Läufe auf je 500 Matrizen.

2.6 Self-Avoiding Walk auf dem Perkolationsgitter

Self-Avoiding Walks (kurz: SAW), also Random-Walks die niemals zu einem zuvor besuchten Gitterplatz zurückkehren sind ein Standardthema in der Physik der weichen Materie, da sie ein Modell für Polymerketten geben. Der Exponent ν^{SAW} des msd beim SAW ist daher von besonderem Interesse, da er eine Verbingung zwischen mittlerer Größe eines Polymers und der Anzahl der Kettenglieder schafft:

$$\tilde{R} \equiv \sqrt{\langle R^2 \rangle} \sim N^{\nu^{SAW}},$$

wobei $\tilde{R}$ der Durchmesser des Polymers ist.
Nun lässt sich auch leicht die Frage formulieren wie sich ein SAW auf dem Perkolationsgitter ausbreitet. Der normale Random-Walk ist langsamer geworden, genauer: $\nu_{pc} \equiv \nu_{pc}^{p.c.} < 1/2$, im Gegensatz dazu wird der SAW auf dem Perkolationsgitter schneller. In anderen Computersimulationen wurde $\nu^{pcSAW} \approx 0.78$ gefunden wohingegen $\nu^{SAW} = 3/4 = 0.75$ ist[5,6]. Die Nachstehende Graphik zeigt, das auch endliche SAW sich auf dem p Perkolationscluster schneller ausbreiten als auf dem freien Gitter[7] Der SAW ist ein wichtiger Vergleich zu dem 'aktiven' Random-Walk.

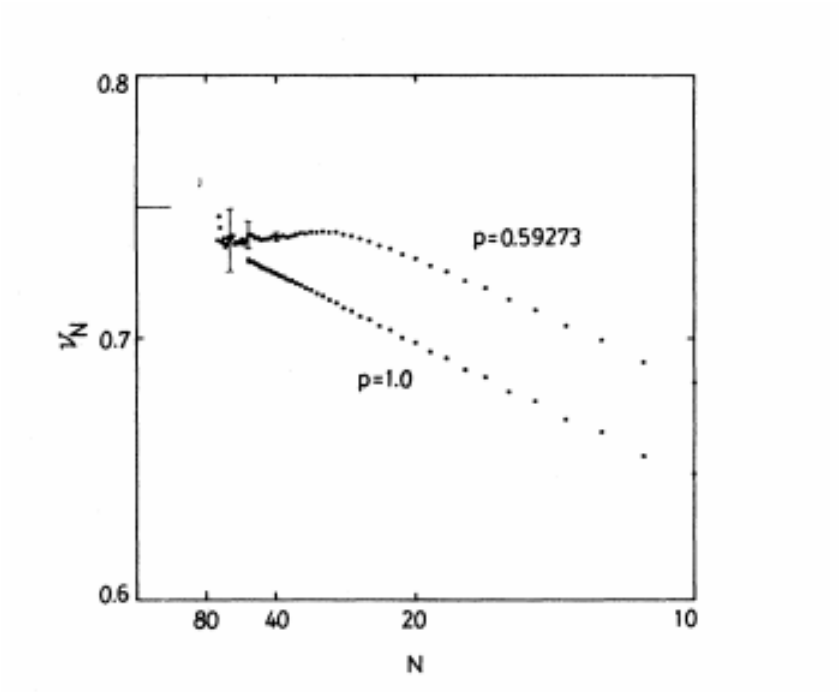

Figure 12: Ergebnisse einer MC-Simulation von Lee, Nakanishi und Kim. N bezeichnet die Länge des SAW's.

2.7 'Aktiver' Random-Walk auf dem Perkolationsgitter

Es wird nun eine Variante des Random-Walk betrachtet, welche zuvor noch nicht besuchte Plätze bevorzugt. Diese Variante eines Random-Walks kann man zum Beispiel für die Modellierung von Chemotaxis von Bakterien verwenden. Das Modell sieht wie folgt aus: man generiert zuerst das Perkolationsgitter, danach werden begehbare Gitterplätze mit 'Nahrung' belegt, welche vom Random-Walker bei dem Besuch des jeweiligen Gitterplatzes vollständig 'gegessen' wird. Der Random-Walk erinnert also an 'Pacman' der durch ein Labyrinth (den perkolierenden Cluster) nach Nahrung absucht.

[5] V. Blavatska und W. Janke, Europhy. Lett. 82, 6606 (2008)
[6] N. Fricke und W. Janke, Europhys. Lett. 99, 56005 (2012)
[7] Lee, Nakanishi und Kim, Phys. Rev. B 39, 13 (1989)

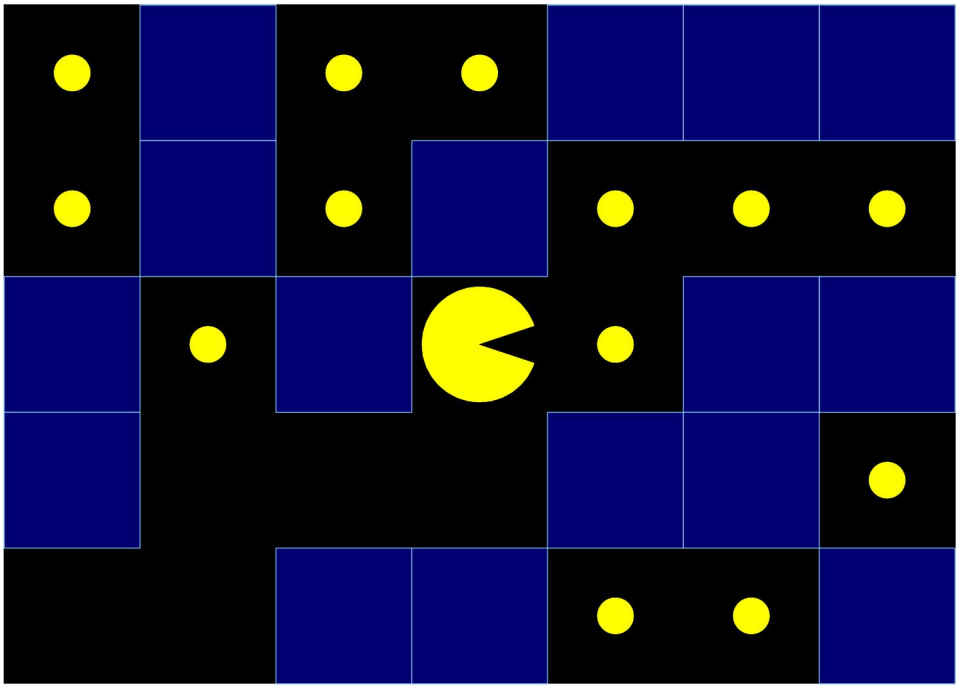

Figure 13: Veranschaulichung des Modells, blaue Quadrate sind geblockte Gitterplätze, kleine gelbe Kreise sind Nahrung und 'Pacman' ist der Random-Walker.[8]

Die Wahrscheinlichkeiten für den nächsten Zug werden gemäß:

$$p_{j \leftarrow i} = \frac{exp(F_j)}{\sum_j exp(F_j)} \tag{1}$$

berechnet, wobei F_j die Menge der Nahrung an Gitterplatz j ist, also F falls Gitterplatz j noch nicht besucht worden ist und 0 falls Gitterplatz j zuvor bereits besucht wurde, i ist dabei der Gitterplatz auf dem der Random-Walker aktuell ist. Dieses Modell scheint, insbesondere für $F \to \infty$, zuerst dem SAW sehr ähnlich zu sein, welcher auf dem Percolationsgitter schneller wird. Monte-Carlo-Simulation dieses 'aktiven' Random-Walkers auf dem Perkolationsgitter zeigt, das der Exponent kleiner ist, als der des normalen Random-Walkers auf dem Perkolationsgitter. Die Nachstehende Abbildung zeigt Ergebnisse einer Monte-Carlo-Simulation mit je 100 Walks auf 100 Perkolationsgittern (Matrizen), wobei nicht auf diagonale perkolation getestet worden ist, mit einer Größe von 1000×1000. Die Nahrung wird in meiner Simulation immer aus der Originalmatrix gelöscht und somit auch aus allen Kopien, welche durch die periodischen Randbedinungen entstehen, diese Methode ist zwar nicht ganz Korrekt (Quelle für finite-size-Effekte), führt aber zu keinem sichtbaren Fehler, solange die Wurzel aus dem msd kleiner als die Boxgröße ist. Der Vorteil ist, dass diese Methode schneller ist und weniger Speicher benötigt. Die Perkolationsschwelle wurde wie zuvor als $p_c = 0.592746$ angenommen.

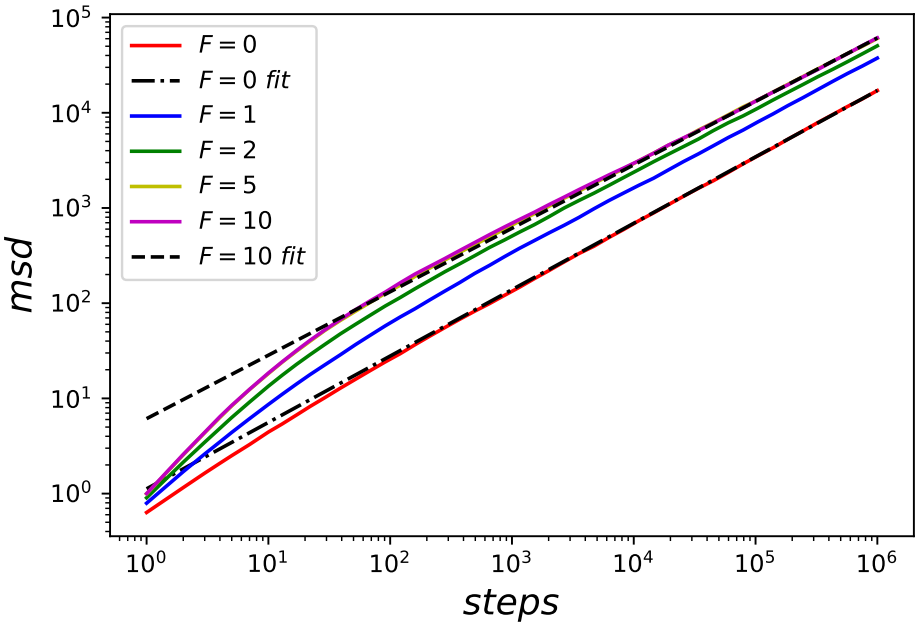

Figure 14: Ergebnisse meiner python3 MC-Simulation des 'aktiven' Random-Walkers auf perkolierendem Cluster ohne diagonale Perkolation. Die gestrichelten Linien sind die mit scipy bestimmten langzeit power-laws.

Es lässt sich erkennen, dass mit steigendem F der Diffusionsexponent ν, also die (halbe) Steigung der Kurve abnimmt, entgegen der naiven Annahme, das je größer F ist, desto ähnlicher wird dieses Modell dem SAW. Es wurden die folgenden Diffusionskoeffizienten gefunden:

F	$2\nu(F)$
0	0.697
1	0.682
2	0.664
5	0.655
10	0.666

Das gleiche Verhalten wurde auch bei einer kleineren Box von $L = 100$ gefunden. Es wurde über 1000 Läufe auf je 100 Perkolationsgittern (inklusive diagonaler Perkolation) gemittelt.

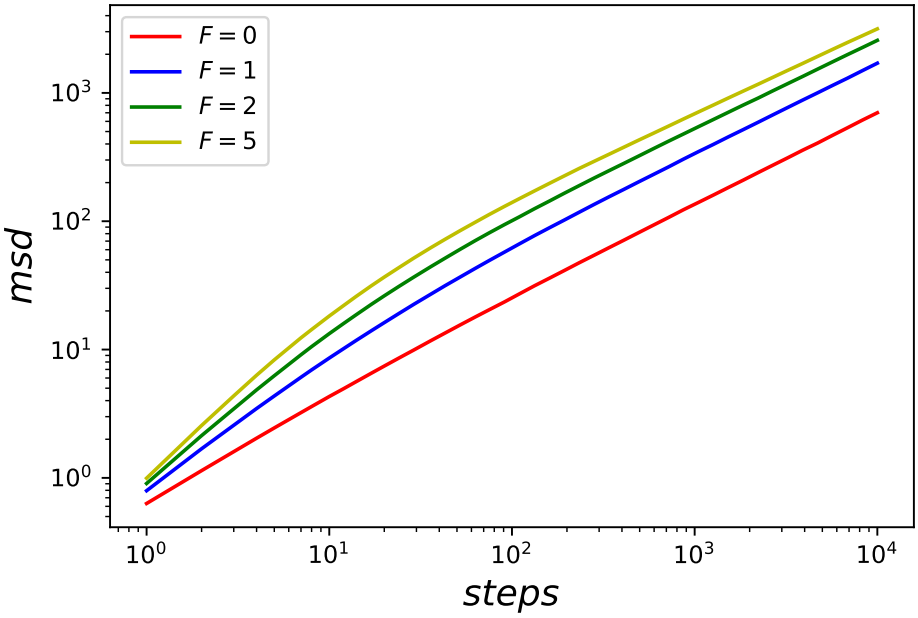

Figure 15: Ergebnisse meiner python3 MC-Simulation des 'aktiven' Random-Walkers auf perkolierendem Cluster mit diagonaler Perkolation.

Erneut wurde die Nahrung die 'Pacman gegessen hat' aus der originalen Matrix und somit auch aus allen Kopien die durch die periodischen Randbedingungen entstehen gelöscht. Die Wurzel aus dem *msd* ist erneut kleiner als die Boxlänge $L = 100$.
Dieses Modell zeigt auch sehr schön den Unterschied zwischen fraktaler und Random-Walk Dimension, da das Perkolationsgitter (also die fraktale Dimension $\approx 1.7 - 1.8$)[9] unabhängig von F ist, die Random-Walk Dimension (also $1/\nu$) aber nicht.

[9]R F Voss, J. Phys. A, 17, 7 (1984)

Monte-Carlo-Simulationen mit einer Boxgröße von $L = 25000$ und bis zu 10^6 Walks auf bis zu 100 000 Matrizen zeigen das gleiche Verhalten. In diesen Simulationen[8] wurden die bereits besuchten Gitterplätze korrekt mit einem 'hash-table' nachgehalten. Die Ergebnisse dieser MC-Simulationen befinden sich in der nachfolgenden Abbildung.

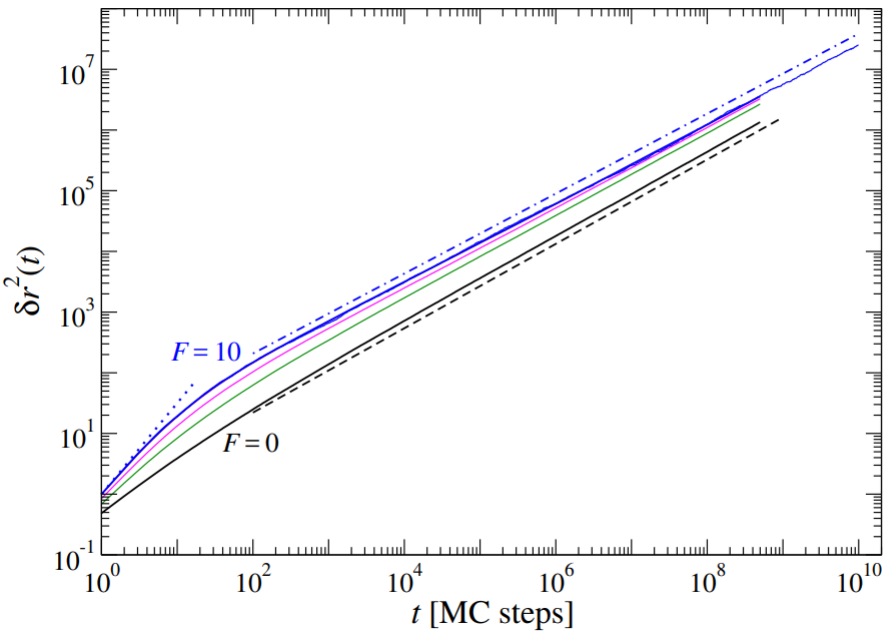

Figure 16: Die blau gepunktete Linie zeigt die power-law des SAW ($\nu^{SAW} = 3/4$), die blau gepunktet/gestrichelte Linie eine power-law zum Exponenten $\nu = 0.33$.

[8] T. Schilling und T. Voigtmann, J. Chem. Phys. 147, 214905 (2017)