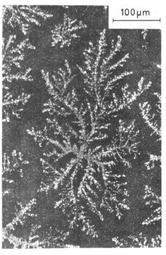
Computerphysik Programmiertutorial 12

Prof. Dr. Matteo Rizzi und Dr. Markus Schmitt – Institut für Theoretische Physik, Universität zu Köln

Github: https://github.com/markusschmitt/comphys2022

Inhalt dieses Notebooks: Zufälliges Clusterwachstum, Herangehensweise für numerische Experimente

Zufälliges Clusterwachstum



Ablagerung von Gold aufgedampft auf einer Glasoberfläche. Bild aus [J. Phys. Chem. 1995, 99, 15, 5639–5644]

Modell:

- Teilchen bewegen sich auf einem Quadratgitter.
- Es gibt ein erstes Teilchen, an dem der Cluster wächst -- der "Seed"
- Teilchenbewegung entspricht einem Random walk.
- Wenn eine der nächsten benachbarten Zellen eines Teilchen zum Cluster gehört, wird das Teilchen mit Wahrscheinlichkeit p_{nn} dem Cluster hinzugefügt und bewegt sich nicht weiter
- Wenn eine der übernächsten benachbarten Zellen eines Teilchen zum Cluster gehört, wird das Teilchen mit Wahrscheinlichkeit p_{nnn} dem Cluster hinzugefügt und bewegt sich nicht weiter

In [1]:
```
using Plots, Random, LinearAlgebra
```

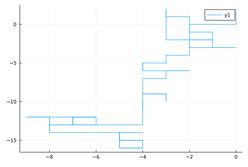
Random walk

In [2]:
```
function walk!(x, directions=[1,2,3,4])

    d = rand(directions)

    if d == 1 # links
        x[1] -= 1
    end
    if d == 2 # rechts
        x[1] += 1
    end
    if d == 3 # oben
        x[2] -= 1
    end
    if d == 4 # unten
        x[2] += 1
    end

end
```

Out[2]:
```
walk! (generic function with 2 methods)
```

In [9]:
```
# Test

x = [0,0]
res = zeros(2,100)
for i in 1:100
    res[:,i] = copy(x)
    walk!(x)
end

plot(res[1,:], res[2,:])
```

Out[9]:



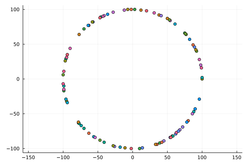
Initialisierung

In [10]:
```
# Erzeuge zufällige Startpositionen auf einem Kreis mit Radius R
function init_particle(R)

    # Zufälligen Winkel generieren
    phi = 2 * pi * rand()

    # Teilchenposition berechnen
    x = zeros(Int, 2)
    x[1] = Int(round(R * cos(phi)))
    x[2] = Int(round(R * sin(phi)))

    return x
end
```

Out[10]:
```
init_particle (generic function with 1 method)
```

In [11]:
```
# Test

plot()
for i in 1:100
    x = init_particle(100)
    scatter!([x[1]],[x[2]], aspect_ratio=:equal, legend=:none)
end
plot!()
```

Out[11]:



Buchhaltung: Nachbarn zählen

In [13]:
```
# Zähle die besetzen benachbarten Gitterplätze
# und erzeuge ein Array mit den unbesetzten Nachbarplätzen (=möglichen Bewegungsrichtungen).
function find_neighbors(x, C)
    L = size(C)[1]

    free_neighbors = []  # Liste der freien benachbarten Gitterplätze

    # nächste Nachbarn
    nearest = 0  # Zahlvariable für besetzt nächste Nachbarn
    # links
    if x[1] > 1
        if C[x[1]-1,x[2]] == 1
            nearest += 1
        else
            push!(free_neighbors, 1)
        end
    end
    # rechts
    if x[1] < L
        if C[x[1]+1,x[2]] == 1
            nearest += 1
        else
            push!(free_neighbors, 2)
        end
    end
    # oben
    if x[2] > 1
        if C[x[1],x[2]-1] == 1
            nearest += 1
        else
            push!(free_neighbors, 3)
        end
    end
    # unten
    if x[2] < L
        if C[x[1],x[2]+1] == 1
            nearest += 1
        else
            push!(free_neighbors, 4)
        end
    end

    # übernächste Nachbarn
    next_nearest = 0 # Zahlvariable für besetzt übernächste Nachbarn
    # links oben
    if x[1] > 1 && x[2] > 1
        if C[x[1]-1,x[2]-1] == 1
            next_nearest += 1
        end
    end
    # rechts oben
    if x[1] < L && x[2] > 1
        if C[x[1]+1,x[2]-1] == 1
            next_nearest += 1
        end
    end
    # links unten
    if x[1] > 1 && x[2] < L
        if C[x[1]-1,x[2]+1] == 1
            next_nearest += 1
        end
    end
    # rechts unten
    if x[1] < L && x[2] < L
        if C[x[1]+1,x[2]+1] == 1
            next_nearest += 1
        end
    end

    return (free_neighbors, nearest, next_nearest)
end
```

Out[13]:
```
find_neighbors (generic function with 1 method)
```

Simulationsschritt

In [33]:
```
function grow!(C, Rmax, p_nn, p_nnn)

    L = size(C)[1] # Gittergröße
    x0 = [div(L,2), div(L,2)] # Position des Seeds

    # Teilchen initialisieren
    x = init_particle(1.5*Rmax) + x0

    # Verwerfe das Teilchen, wenn es sich zu weit vom Cluster entfernt
    while norm(x-x0) < min(3*Rmax, div(L,2))
        # Finde Nachbarn
        free_neighbors, nearest_neighbors, next_nearest_neighbors = find_neighbors(x,C)

        # Teilchen abentscheiden
        for _ in 1:nearest_neighbors
            if rand() < p_nn
                C[x[1], x[2]] = 1
            end
        end
        for _ in 1:next_nearest_neighbors
            if rand() < p_nnn
                C[x[1], x[2]] = 1
            end
        end
        if C[x[1], x[2]] == 1
            Rmax = max(Rmax, norm(x-x0))
            return Rmax
        end

        # Teilchen bewegen
        walk!(x, free_neighbors)

    end

    return Rmax
end
```

Out[33]:
```
grow! (generic function with 1 method)
```

Simulation

In [14]:
```
Random.seed!(4321)

L = 400 # Systemgröße
C = zeros(Int8, L, L) # Cluster array
C[div(L,2),div(L,2)] = 1  # Seed Teilchen

# Wahrscheinlichkeiten
p_nn = 0.8
p_nnn = 0.05

# Anhaltspunkt für R_max
R_max = 10

anim = @animate for i in 1:200
    global R_max
    for i in 1:20
        # Simulation step
        R_max = grow!(C, R_max, p_nn, p_nnn)
    end

    # Plot
    heatmap(C, xlims=(180,300), ylims=(100,300), axis=(off), xticks=nothing, yticks=nothing,
            legend=:none, aspect_ratio=:equal, color=cgrad(:grays,rev=true))
end

gif(anim, "cluster_growth.gif")
```

```
[ Info: Saved animation to
    fn = /Users/markus/Cloud/synology/Teaching/Computerphysik/2022/comphys2022/tutorials/cluster_growth.gif
┌ Plots /Users/markus/.julia/packages/Plots/5B9gd/src/animation.jl:114
```

Out[14]:

Problemlösungsalgorithmus für numerische Experimente

Numerische Experimente werden schnell komplex. Daher sollte man beim Lösen des Problems schrittweise vorgehen um den Überblick zu behalten und Fehler frühzeitig zu identifizieren:

1. Analysieren des Problems: Gleichungen verstehen und numerische Aufgaben identifizieren.
2. Zerlegen des Problems in Teilprobleme.
3. Lösungen der Teilprobleme implementieren (separate Funktionen).
4. Lösungen mit einfachen Beispielen testen.
5. Daten produzieren.
6. Daten analysieren. Das Ergebnis auf Plausibilität prüfen.