Programierpraktikum Übung 4

Alexander Steding
10028034
Gottfried Wilhelm Leibniz Universität
19. September 2022

Inhaltsverzeichnis

1 Einleitung

Dieser Bericht stellt als Abschlussbericht die Ergebnisse und Erkenntnisse des Programmierpraktikums Schadstoffausbreitung zusammen. Als Programmiersprache wurde über alle Aufgaben hinweg Julia verwendet und als Graphisches Backend PlotlyJS.

2 Aufgabe 1

Ziel dieser Aufgabe ist es ein Gauß-Modell für eine kontinuierliche Linienquelle zu Programmieren und anschließend die Maximalkonzentration am Erdboden zu bestimmen.

In Aufgabenteil b wird ein Monte-Carlo-Modell für eine kontinuirliche Linienquelle programmiert und mit dem Gauß-Modell aus Aufgabenteil a verglichen.

2.1 Aufgabenteil a

In Abbildung 1. ist die zu erwartende Konzentrationsverteilung des Gauß-Modells als X-Z-Schnitt visualisiert.

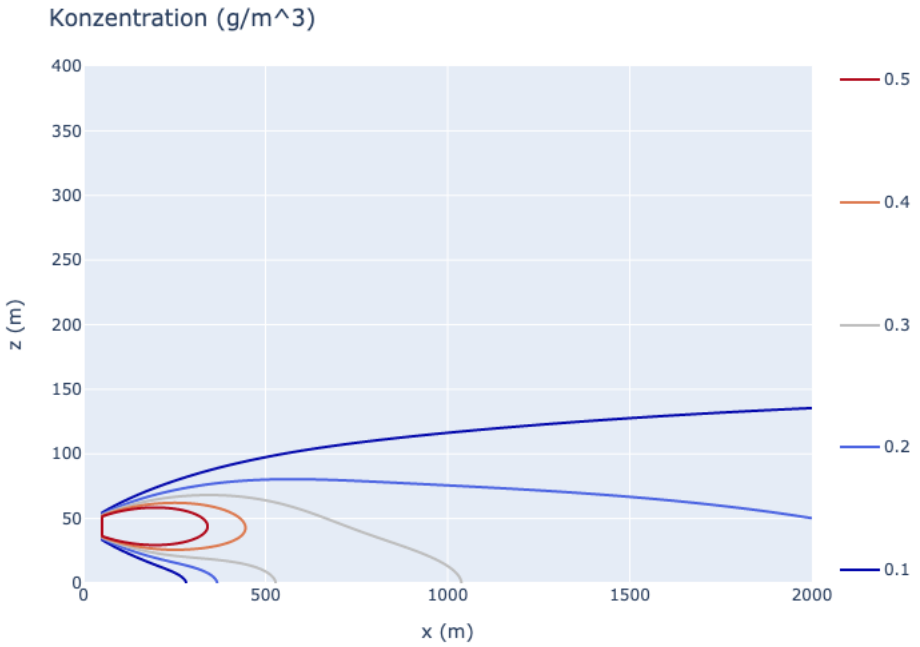

Abb. 1: Konzentrationsverteilung

Für eine feinere grapische Analyse wurde in Abbildung 2. die Konzentrationsverteilung am Erdboden, also für $z = \mathrm{m}$, visualisiert.

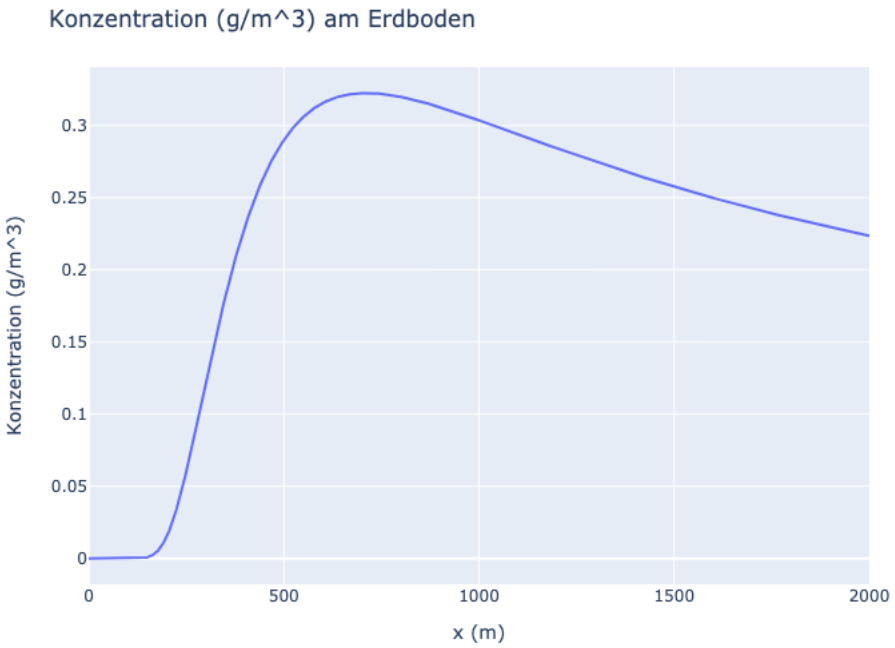

Abb. 2: Konzentrationsverteilung am Erdboden

Die maximale Konzentration wurde final rechnerisch mittels Julia bestimmt als

$$c[0,711] = 0,32263 \frac{g}{m^3} \tag{1}$$

2.2 Aufgabenteil b

Für den Vergleich zwischen Monte-Carlo-Modell und Gauß-Modell wurden beide Modelle in einem Contour-Plot visualisiert. Für die optimale Visualisierung wurden verschiedene Partikelanzahlen visualisiert.

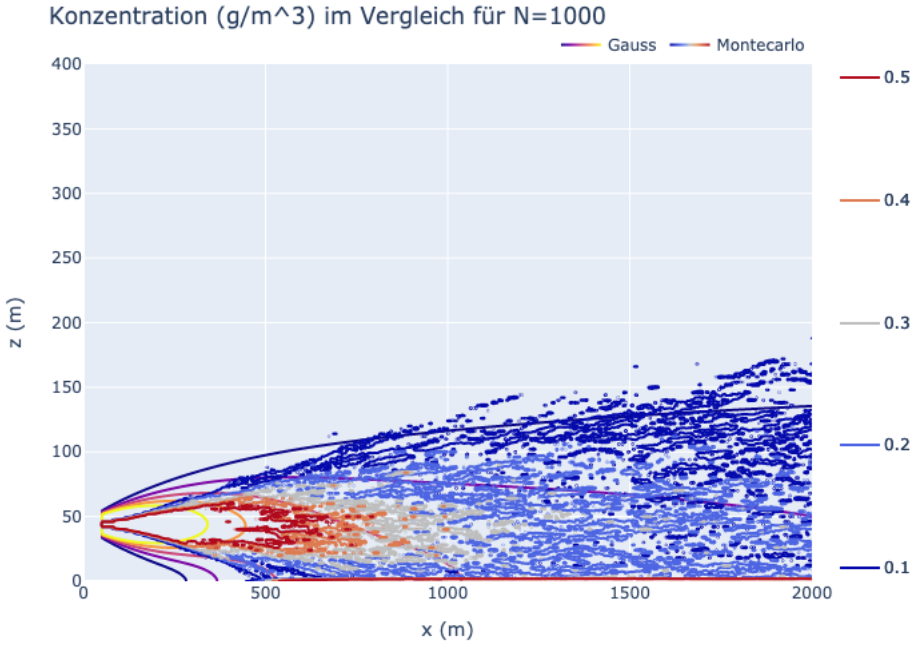

Abb. 3: Vergleich für N=1000

In Abb. 4 ist leider keine gute Annäherung des Montecarlo Modells an das Gaußmodell zu erkennen. Die Anzahl der Teilchen hat beim Montecarlo Modell einen hohen Einfluss auf die Güte des Modells. Bei einer geringeren Anzahl wie

$$N = 1000 \tag{2}$$

sind extrem große Abweichungen zum Gaußmodell zu erkennen.

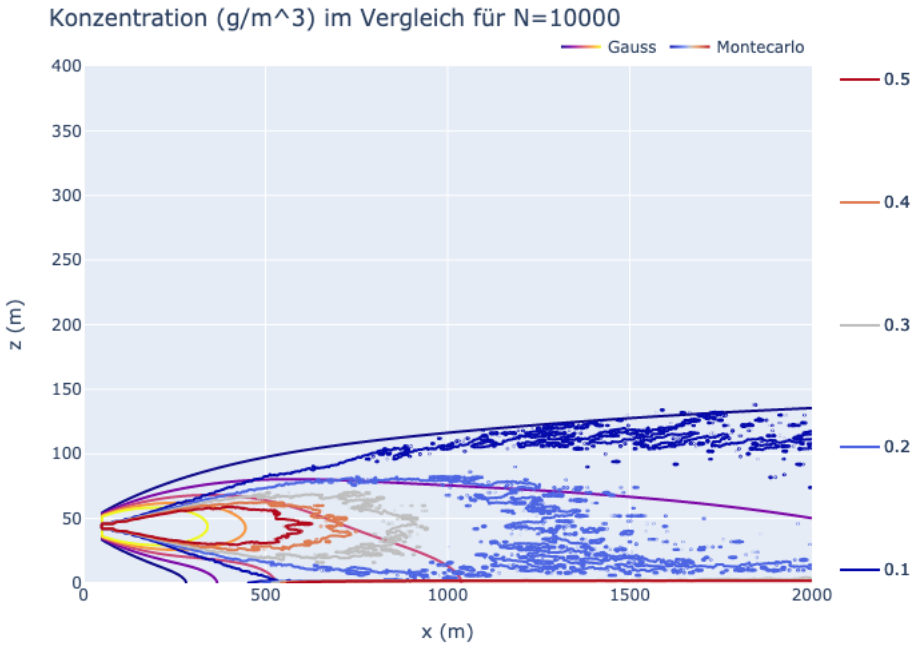

Abb. 4: Vergleich für N=10000

Bei einer mittleren Anzahl wie

$$N = 10000 \tag{3}$$

gleicht sich das MC Modell bedeutend besser an das Gaußmodell an.

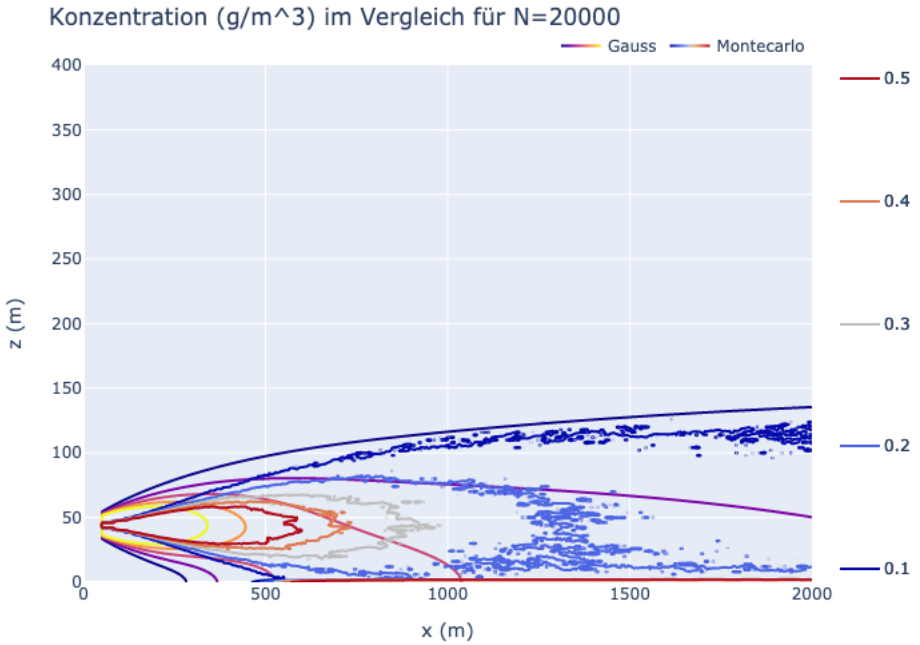

Abb. 5: Vergleich für N=20000

Bei einer hohen Anzahl wie

$$N = 20000 \tag{4}$$

gleicht sich das MC Modell sehr gut an das Gaußmodell an.

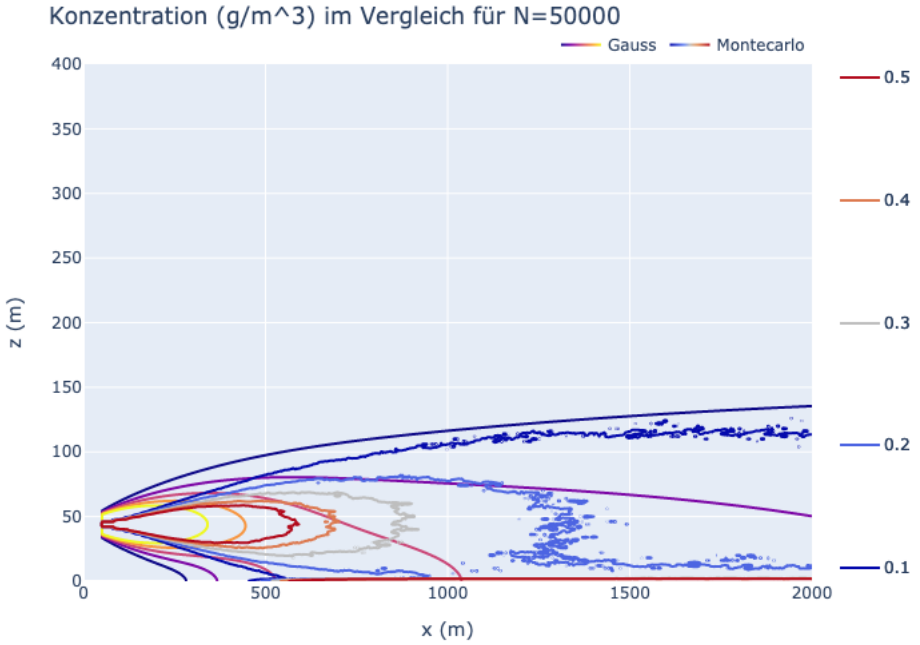

Abb. 6: Vergleich für N=50000

Abbildung 5 zeigt das mit meiner Hardware maximal mögliche Ergebnis. Jenseits von dieser Anzahl sind zwar noch leichte Verbesserungen zu erwarten, nichts destotrotz zeigt sich bei

$$N = 50000 \tag{5}$$

eine extrem gute Annäherung an das Gauß-Modell. Für eine Ausreichende Statistik reichen aber bereits

$$N = 20000 \tag{6}$$

.

3 Aufgabe 2

In dieser Aufgabe sollte das Monte-Carlo-Modell mit der Prandtl-Schicht optimiert und die Ergebnisse durch das Prairie-Grass-Experiment validiert werden.

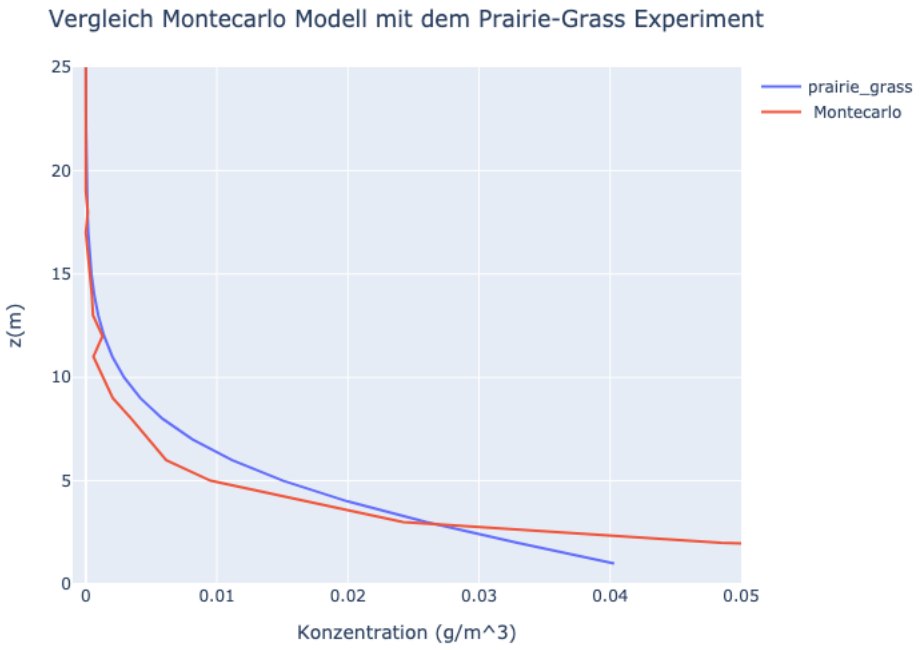

Abb. 7: Vergleich Prairie-Grass

4 Aufgabe 3

Ziel dieser Aufgabe ist die Anwendung der bisher verwendeten Monte-Carlo-Methode in einem praxisnahen Beispiels. Dabei wurden die für eine Straßenschlucht gemessenen Windgeschwindigkeiten sowie deren Standartabweichung importiert und ein Contourprofil aus der berechneten Konzentration gebildet. Aufgrund der besseren Optimierung wurde für die komplexeren Berechnungen statt PlotlyJS Makie verwendet.

4.1 Aufgabenteil a

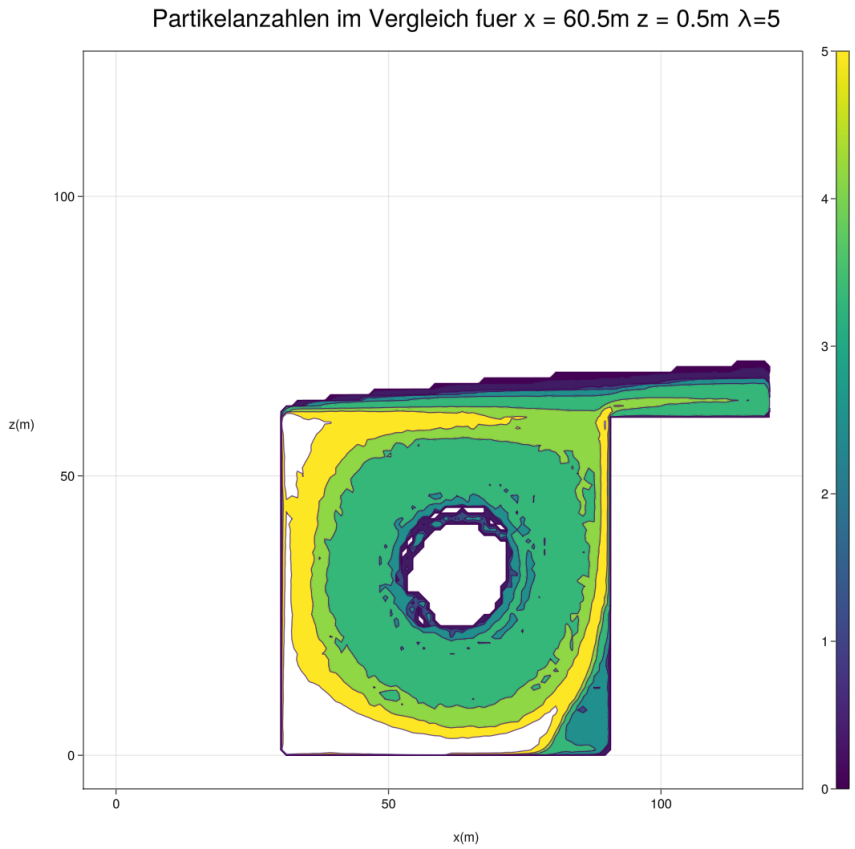

Abb. 8: Konzentrationsverteilung am Erdboden

4.1.1 Vergleich der Teilchenanzahl

4.2 Aufgabenteil b

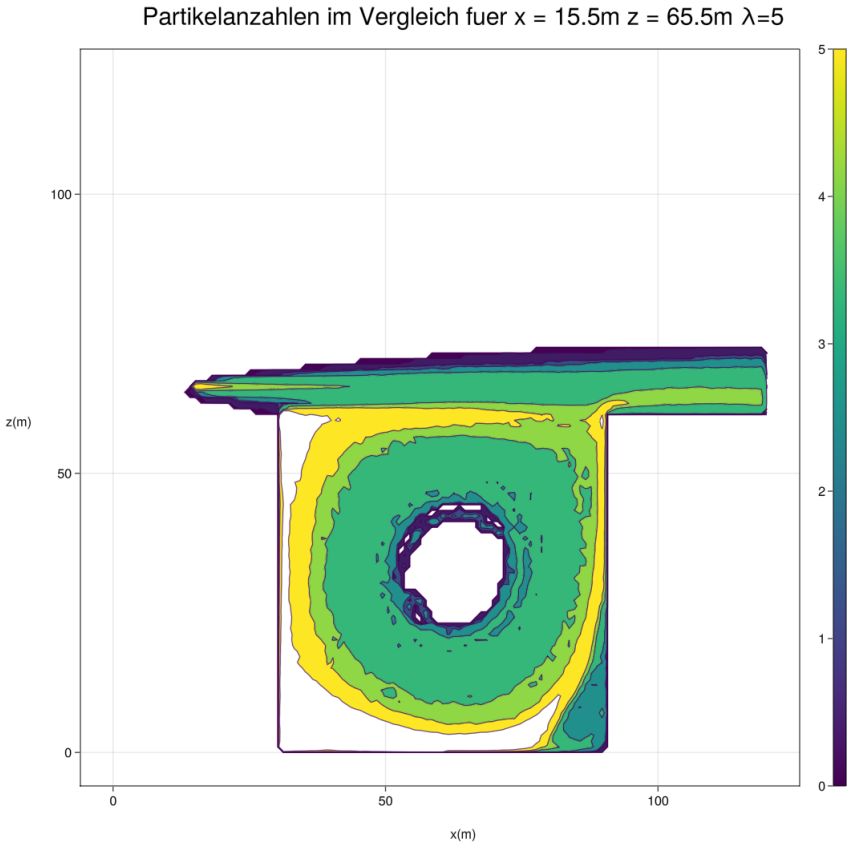

Abb. 9: Konzentrationsverteilung am Erdboden

4.3 Einfluss der Teilchenanzahl

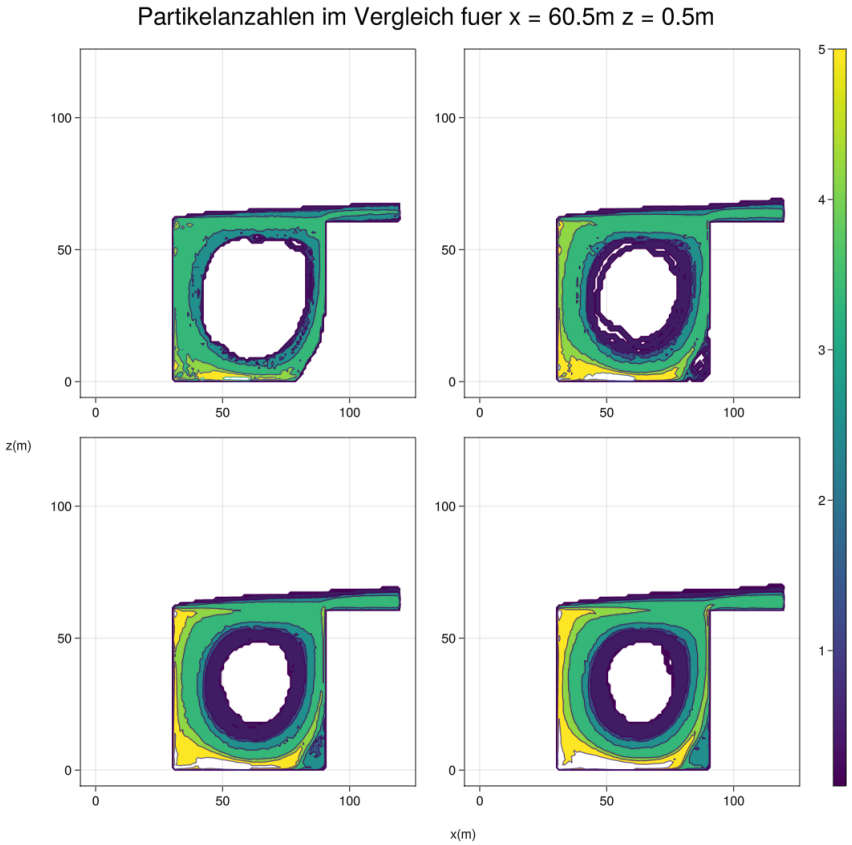

Abb. 10: Verschiedene Konzentrationsverteilung im Vergleich. $N \in [10^2, 10^3, 5 * 10^3, 10^4]$

Für die Situation a) bringt eine Erhöhung der Teilchenzhahl von $N = 100$ dargestellt in Abblidung 10 linksoben auf $N = 1000$ einen enormen Sprung in der Qualität der Konzentration. Dadurch können so zum Beispiel auch die feineren Wirbelstrukturen im Zentrum der Häuserschlucht erfasst werden. Eine weitere erhöhung auf $N = 5000$ ermnöglicht noch eine gering höhere Auflösung; $N = 10000$ führen trotz höherer Rechenzeit nur zu einer marginalen Verbesserung. Aus ökonomischen Gesichtspunkten ist eine Teilchenanzuahl zwischen 1000 und 5000 optimal.

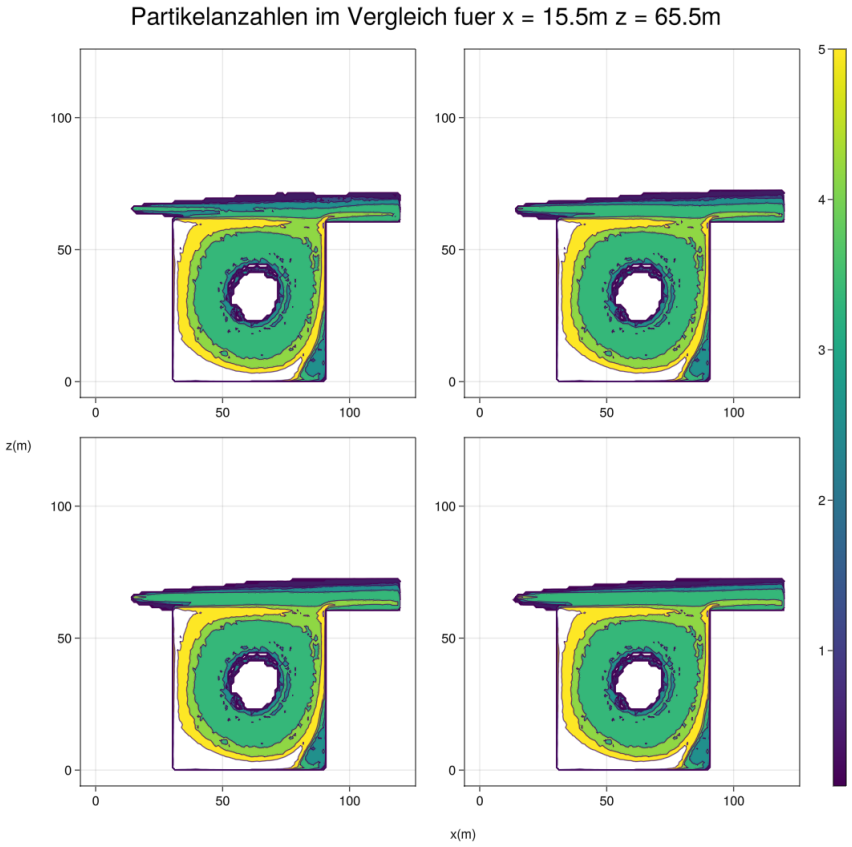

Abb. 11: Verschiedene Konzentrationsverteilung im Vergleich. $N \in [10^2, 10^3, 5 * 10^3, 10^4]$

Für die in Aufgabenteil b) geschilderte Situation zeichnet sich ein ähnlicher Verlauf ab. Zwischen $N = 100$ und $N = 1000$ ist auch hier der Qualitätssprung gut zu erkennen. Im unterschied zur vorausgegangenen Simulation führt eine Steigerung der Partikelanzahl jenseits von $N = 1000$ zu keiner visuellen Verbesserung.

4.4 Einfluss von λ

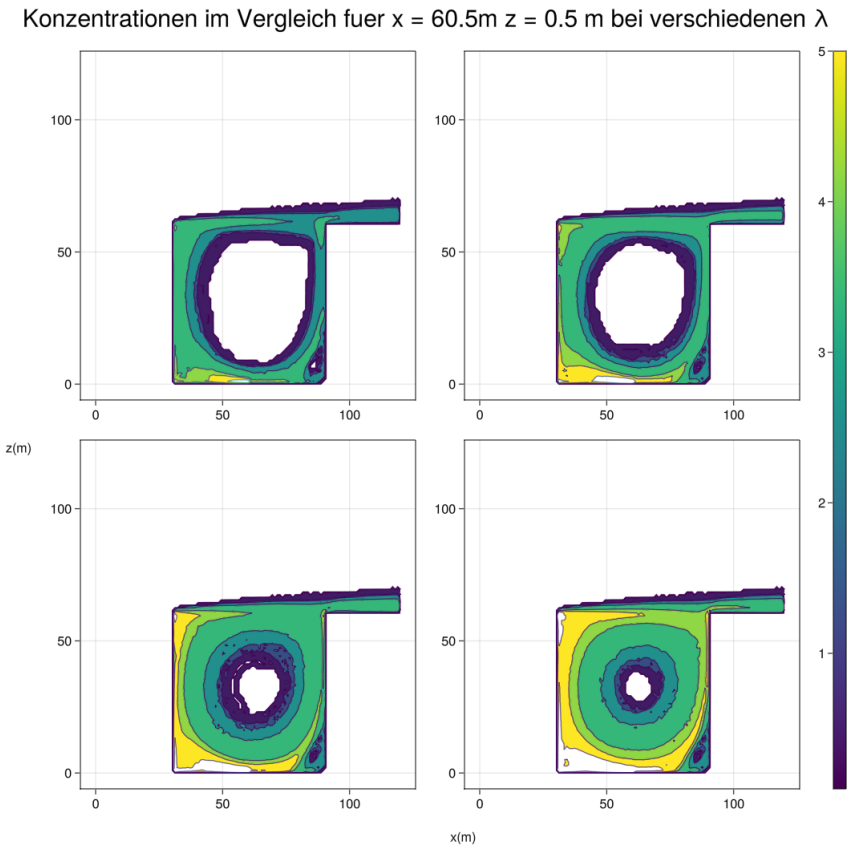

Abb. 12: Verschiedene $\lambda im Vergleich. \lambda \in [10, 5, 2, 1]$

Bilder/3_lambda_x = 15.5_5.png

Abb. 13: Verschiedene $\lambda im Vergleich. \lambda \in [10, 5, 2, 1]$

4.5 Einfluss von σ

Bilder/3_sigma_x = 60.5.png

Abb. 14: Konzentrationsverteilung am Erdboden

Bilder/3_sigma_x = 15.5_5.png

Abb. 15: Konzentrationsverteilung am Erdboden

5 Aufgabe 4

Ideen helfen einen weiter

6 Quellcode

6.1 Aufgabe 2

```
using NetCDF
using Random, Distributions
using ProgressBars
using PlotlyJS

# Definition der globalen Variablen
global n,ubalken,wbalken,zq,xq,xgrenz,zgrenz,tl,nx,ny,nz,dx,dy,dz::Int
global dt,sigu,sigw,ustern,k,znull,q::Float64
global units::String
global gitter,cd,x,cdground::Array

n= 10^3              # Anzahl Partikel
ubalken = 5          # mittlere Windkomponente in u Richtung in m/s
wbalken = 0          # mittlere Windkomponente in w Richtung in m/s
zq = 0               # Quellort z Komponente in m
xq = 0.5             # Quellort x Komponente in m
xgrenz= 110          # Grenze in x Richtung in m
zgrenz=25            # Grenze in z Richtung in m
tl = 100             #s Zeit
dt = 0.4             # Zeitschritt in s
dx = 1               # Gitterweite x Richtung in m
dz = 1               # Gitterweite z Richtung in m
ui=5                 #
ustern = 0.35        #
k = 0.38             # Kappa
znull = 0.008        # Rauhigkeitslaenge
sigu = 2.5 * ustern # Standartabweichung u m/sm/s
sigw = 1.3 * ustern # Standartabweichung w m/s m/s
q= 0.7               # Konzentration fuer das Montecarlo Modell in m/s
rl= exp(- dt/tl)     # Berechnung von rl
units = "g/m^3"      # Einheit fuer Graphen

## Arrays Initialisieren


gitter=zeros(xgrenz,zgrenz)
konk=zeros(xgrenz,zgrenz)


##Funktionen ##

### Geradengleichung ### Fuer die Exakte Gitterauswertung ist es notwendig
function gg(xold, zold, xi, zi, t)
    xg = xold + t * (xi - xold)
    zg = zold + t * (zi - zold)
    if xg>xgrenz
        xg=xgrenz
    end
    if floor(zg) <1
        zg=1
    end
    if floor(xg) <1
        xg=1
```

```
    end
    return convert(Int64, floor(xg)), convert(Int64, floor(zg))
end


### Manager###
function rangecheck(xi, xold, zi, zold,dt)
    rangex = floor(xi - xold)
    rangez = floor(zi - zold)
    if (rangex + rangez) < 2
        gitweis(xi, zi,dt)
    else
        exaktgitter(xi, xold, zi, zold,dt)
    end
end

### Berechnung der Prandtlschicht###
function prandltl(zi,xi)
    if zi < znull
        ubalken = 0
    else
        ubalken = (ustern / k) * log(abs(zi) / znull)
    end

    tl = ((k * ustern) / sigw ^ 2) * abs(zi)
    if (0.1*tl)>((k * ustern) / sigw ^ 2) * abs(2)
        dt = 0.1*tl                             # Normalfall
    else
        dt = ((k * ustern) / sigw ^ 2) * abs(2) #falls dt kleiner als tl in
   2 m Hoehe wird dt auf tl(2m) gesetzt
    end
    return tl,  dt, ui
end

### Exakte Gitterauswertung###
function exaktgitter(xi, xold, zi, zold,dt)
    ti = []
    toks = []
    rangex = convert(Int64,floor(xi - xold))    # Bestimmung der Anazhl an
   Schnittpunkten mit der X-Achse
    rangez = convert(Int64,floor(zi - zold))    # Bestimmung der Anazhl an
   Schnittpunkten mit der Z-Achse
    xsi = ceil(xold)
    zsi = ceil(zold)
    for i in 0:rangex

        if i == 0
            xsi = ceil(xold)
            push!(toks,(xsi - xold) / (xi - xold))
        else
            xsi += 1
            push!(toks,(xsi - xold) / (xi - xold))
        end
    end
    for i in 0:rangez
        if i == 0
            zsi = ceil(zold)
            push!(toks,(zsi - zold) / (zi - zold))
```

```
109         else
110             zsi += 1
111 
112             push!(toks,(zsi - zold) / (zi - zold))
113         end
114     end
115     tku = sort!(toks)
116     for i in 2:length(tku)
117         ti = tku[i]
118         told = tku[i - 1]
119         t = mean([told, ti])
120         posx, posz = gg(xold, zold, xi, zi, t) # Aufrufen der
    Geradengleichung zur Berechnung der Positionen
121         gitter[posx, abs(posz)] += ((tku[i] - tku[i-1] )* dt) # Ein
122         konk[abs(posx), abs(posz)] += ((tku[i] - tku[i-1])* dt*((q * dt)/(n
    * dx * dz))) #Eintragen der Positionen an den Positionen
123     end
124 end
125 
126 ### Berechnung der Positionen###
127 function positionen(xi, wi, zi,tl, ui, dt)
128 
129     rl = exp(- dt / tl)
130     Random.seed!()
131     d = Normal()
132     rr = rand(d, 1)[1]
133 
134     xi = xi + ui * dt
135     wi=rl*wi + sqrt((1 - rl^2))*sigw* rr
136     zi = zi + wi * dt
137 
138 
139 
140 return xi, wi, zi
141 
142 end
143 
144 ### ungefaehre Gitterauswertung ###
145 function gitweis(xi, zi,dt)
146     if floor(zi) <1
147         zi=1
148     end
149     xm = abs(convert(Int64,floor((xi))))
150     zm = abs(convert(Int64,floor((zi))))
151     gitter[xm, zm] = gitter[xm, zm] + 1
152     konk[xm, zm] += 1*((q * dt)/(n * dx * dz))
153     return
154 end
155 
156 
157 function monte()
158     for i in ProgressBar(1:n+1)
159         xi=xq
160         zi=zq
161         ui=ubalken
162         wi=wbalken
163         dt=0
164 
```

```
165
166         while (ceil(xi+ui*dt) < xgrenz)
167             xold=xi
168             zold=zi
169             if zi<1
170                 zi=-zi
171                 wi= -wi
172                 tl, dt,ui = prandltl(zi,xi)
173                 xi,wi,zi =positionen(xi, wi, zi,tl, ui, dt)
174                 rangecheck(xi, xold, zi, zold,dt)
175
176
177             else
178                 tl, dt,ui = prandltl(zi,xi)
179                 xi,wi,zi =positionen(xi, wi, zi,tl, ui, dt)
180                 rangecheck(xi, xold, zi, zold,dt)
181             end
182         end
183
184     end
185     return konk
186 end
187
188 function prairie_grass(konk)
189     pg_mod= []
190     for i in 100:xgrenz
191         for j in 1:zgrenz
192             #print(cd[j,1])
193             push!(pg_mod, konk[i,j])
194
195         end
196
197     end
198     c0 = 4.63E-02
199     gamma = 0.68
200     my = 1.3
201     zs = 3.4
202     z= collect(1:zgrenz)
203     pg = zeros(length(z)+1)
204 for k in 1:zgrenz
205    pg[k] = c0 * exp(-gamma * (z[k]/zs)^my)
206 end
207 print(pg_mod)
208 return pg, pg_mod
209 end
210
211
212 ### Visualisierung ###
213
214  function grafen(pg,pg_mod)
215 savefig(plot([
216
217 scatter(
218 y=collect(1:zgrenz),
219 x=pg,
220 name="prairie_grass",
221 showlegend=true ,),
222
```

```
scatter(
y=collect(1:zgrenz),
x=pg_mod,
name=" Montecarlo",
showlegend=true ,)],

Layout(
    title="Vergleich Montecarlo Modell mit dem Prairie-Grass Experiment",
    xaxis_title="Konzentration (" * units * ")",
    yaxis_title="z(m)",
    xaxis_range=[-0.001, 0.05],
    yaxis_range=[0, 25]
)), "Bericht/Bilder/2.png")
 end



function main()
    konzentrationen=monte()
    pg,pg_mod=prairie_grass(konzentrationen)
    grafen(pg,pg_mod)

end


main()
```

6.2 Aufgabe 3

```
using NetCDF
using Random, Distributions
using ProgressBars
using LinearAlgebra
using PlotlyJS
using GLMakie
using FileIO

xgrenz = 120  # !m
zgrenz = 120
units = "g/m^3"     # Einheit fuer Graphen
#r = symbols("r")
xlist=[]
zlist=[]
dx = 1
dy = 1
dz = 1
ges=[]
k = 0.38
#znull = 0.008
q = 150
gitter=zeros(xgrenz,zgrenz)
konk=zeros(xgrenz,zgrenz)
function gg(xold, zold, xi, zi, t)

    xg = xold + t * (xi - xold)
    zg = zold + t * (zi - zold)
    if xg>xgrenz
```

```
        xg=xgrenz
    end
    if floor(zg) <1
        zg=1
    end
    return convert(Int64, floor(xg)), convert(Int64, floor(zg))
end

function prandltl(zi,xi,lambda,marongus,marongws,k)
    xii=convert(Int64,floor(xi))
    zii=convert(Int64,floor(zi))
    if floor(zi)==0
zii=1
    end
    tl= 0.05*((k*zii)/(1+k*(zii/lambda)))/(0.23*sqrt(marongus[xii+1,zii+1]+
   marongws[xii+1,zii+1]))
    if (0.1*tl)>0.05*((k*2)/(1+k*(2/lambda)))/(0.23*sqrt(marongus[xii+1,zii
   +1]+marongws[xii+1,zii+1])) #falls dt kleiner als tl in 2 m Hoehe
      dt = 0.1*tl
    else
        dt = 0.05*((k*2)/(1+k*(2/lambda)))/(0.23*sqrt(marongus[xii+1,zii+1]+
   marongws[xii+1,zii+1]))
    end
    return tl,  dt
end

function rangecheck(xi, xold, zi, zold,dt,n)
    rangex = floor(xi - xold)
    rangez = floor(zi - zold)
    if (rangex + rangez) < 2
        gitweis(xi, zi,dt,n)
    else
        exaktgitter(xi, xold, zi, zold,dt)
    end
end

function exaktgitter(xi, xold, zi, zold,dt)
    ti = []
    toks = []
    rangex = convert(Int64,floor(xi - xold))
    rangez = convert(Int64,floor(zi - zold))
    xsi = ceil(xold)
    zsi = ceil(zold)
    for i in 0:rangex

        if i == 0
            xsi = ceil(xold)
            push!(toks,(xsi - xold) / (xi - xold))
        else
            xsi += 1
            push!(toks,(xsi - xold) / (xi - xold))
        end
    end
    for i in 0:rangez
        if i == 0
            zsi = ceil(zold)
            push!(toks,(zsi - zold) / (zi - zold))
        else
```

```
            zsi += 1

            push!(toks,(zsi - zold) / (zi - zold))
        end
    end
    tku = sort!(toks)
    for i in 2:length(tku)
        ti = tku[i]
        told = tku[i - 1]
        t = mean([told, ti])
        posx, posz = gg(xold, zold, xi, zi, t)
        gitter[posx, abs(posz)] += ((tku[i] - tku[i-1] )* dt)
        konk[abs(posx), abs(posz)] += ((tku[i] - tku[i-1])* dt*((q * dt)/(n * dx * dz)))
    return

    end
end

function positionen(xi, wi, zi, tl, ui, dt,xold,zold,xolder,zolder,marongus, marongws,marongu,marongw )

ixolder= floor(xolder )+1
izolder=floor(zolder )+1
    ixold= floor(xold )+1
    izold= floor(zold )+1
    izi= floor(zi )+1
    ixi= floor(xi)+1
    rl = exp(- dt / tl)
    Random.seed!()
    d = Normal()
    rr = rand(d, 1)[1]
    if izi == 1
        izi = 2
    end
    if ixi == 91
        ixi=90
    end
    if ixold== 91
        ixold=90
    end
    if izold== 1
        izold=2
    end

    difqu= abs(marongus[abs(convert(Int64,(ixolder))),abs(convert(Int64,(izolder)))]-marongus[abs(convert(Int64,(ixi))),abs(convert(Int64,(izi)))])/ 2* abs(xolder-xi)
    difqw=abs(marongws[abs(convert(Int64,(ixolder))),abs(convert(Int64,(izolder)))]-marongws[abs(convert(Int64,(ixi))),abs(convert(Int64,(izi)))])/ 2*abs(xolder-xi)


    ukack= rl *ui  + sqrt((1 - rl ^ 2)) * sqrt(marongus[abs(convert(Int64,(ixi))),abs(convert(Int64,(izi)))])*rr +(1-rl)*tl*difqu
    ui= marongu[abs(convert(Int64,(ixold))),abs(convert(Int64,(izold)))] + ukack
    xi = xi + ui * dt
```

```
134     wkack = rl * wi + sqrt((1 - rl ^ 2)) *sqrt(marongws[abs(convert(Int64,(
    ixi))),abs(convert(Int64,(izi)))])*rr +(1-rl)*tl*difqw
135     wi=marongw[abs(convert(Int64,(ixold))),abs(convert(Int64,(izold)))] +
    wkack
136     zi = zi + wi * dt
137     while ((xi<=31 && zi<=61)||(xi>=90 && zi <=61)||(30<=xi<=90 && zi<=1)||
    zi<1)
138         if (xi>=90 && zi<=61) # rechte Wand
139             br=1
140             if br==1
141                 xi=xi-2*(abs(90-xi))
142                 ui=-ui
143             end
144         elseif (xi<=31 && zi<=61) && zi>1 #linke Wand
145             br=1
146             if br==1
147             xi=xi+2*(abs(31-xi))
148             ui=-ui
149             end
150 
151         else  #Boden
152             #println("Boden ist aus Lava")
153             wi=-wi
154             zi=zi+2*(abs(1-zi))
155         end
156 
157 end
158 return xi, wi, zi,ui
159 
160 end
161 
162 """
163 function  eckendreck(xi,zi,xold,zold,ui,wi)
164     if xi<=30&& typeof(solve(r*[1,1]+[30,60]-[xold,zold],r)) !=Vector{Any}
    && typeof(solve(r*[1,1]+[30,60]-[xi,zi],r)) !=Vector{Any}
165         ui=-ui
166         wi=-wi
167         xi=xold
168         zi=zold
169         br=2
170 
171     elseif xi<=30&& typeof(solve(r*[1,1]+[30,0]-[xold,zold],r)) !=Vector{Any
    } && typeof(solve(r*[1,1]+[30,0]-[xi,zi],r)) !=Vector{Any}
172             ui=-ui
173             wi=-wi
174             xi=xold
175             zi=zold
176             br=2
177     elseif xi>=90 && typeof(solve(-r*[1,1]+[90,0]-[xold,zold],r))!=Vector{
    Any} && typeof(solve(-r*[1,1]+[90,0]-[xi,zi],r)) !=Vector{Any}
178             ui=-ui
179             wi=-wi
180             xi=xold
181             zi=zold
182             br=2
183     elseif xi>=90&& typeof(solve(-r*[1,1]+[90,60]-[xold,zold],r))!=Vector{
    Any} && typeof(solve(-r*[1,1]+[90,60]-[xi,zi],r)) !=Vector{Any}
184         ui=-ui
```

```
        wi=-wi
        xi=xold
        zi=zold
        br=2
else
    ui=ui
    wi=wi
    br=1
end
return ui,wi, br
end
"""

function gitweis(xi, zi,dt,n)
    if floor(zi) <0
        zi=0
    end
    xm = abs(convert(Int64,floor((xi +1))))
    zm = abs(convert(Int64,floor((zi +1))))
    gitter[xm, zm] = gitter[xm, zm] + 1
    konk[xm, zm] += 1*((q * dt)/(n * dx * dz))
    return
end

function monte(xq,zq,n,lambda,sigma)
    #using SymPy
#n = 10^3# !Anzahl Partikel



marongu =  ncread("Bericht/input_uebung5.nc", "u")
marongw =ncread("Bericht/input_uebung5.nc","w")
marongus = sigma*ncread("Bericht/input_uebung5.nc","u2")
marongws = sigma*ncread("Bericht/input_uebung5.nc","w2")
gurkenlist=findall(x->x==-9999.0,marongu)
gurkenlistw=findall(x->x==-9999.0,marongw)
for i in 1: length(gurkenlist)
marongu[gurkenlist[i]]=NaN
end

for i in 1: length(gurkenlistw)
    marongw[gurkenlist[i]]=NaN
end
for i in ProgressBar(1:n)
    xi = xq
    zi = zq
    dt=0
    ui=0
    wi=0
    xold=xq
    zold=zq
    while (ceil(xi+ui*dt) < xgrenz)
        xolder=xold
        zolder=zold
        xold = xi
        zold = zi
            tl, dt = prandltl(zi,xi,lambda,marongus,marongws,k)
            xi, wi, zi,ui = positionen(xi, wi, zi, tl, ui, dt,xold,zold,
```

```
   xolder,zolder,marongus,marongws,marongu,marongw)

            rangecheck(xi, xold, zi, zold,dt,n)
            push!(xlist, xi)
            push!(zlist, zi)
    end

end
return  konk
end



function vergleichmakie(xq,zq,lambda)
na= 100
nb= 1000
nc =5000
nd= 10000
sigma=1


levels= [0.00001,0.01,1.0,2,10,100,200,500]#-1:0.1:1#
   [0,0.01,0.025,0.05,0.5,0.75,1.0,1.25,1.5,2]#[0.001,0.005,0.01,0.05,0.1,0.5,0.75,1,2

 fig = Figure(resolution=(1080,1080))
 xs=LinRange(0, xgrenz,xgrenz)
 ys=LinRange(0, zgrenz, zgrenz)



a=monte(xq,zq,na,lambda,sigma)
contourf(fig[1, 1],ys, xs, a,levels=levels)
contour!(fig[1, 1],ys, xs,a ,levels=levels )
b=monte(xq,zq,nb,lambda,sigma)
contourf(fig[1, 2],xs, ys, b,levels=levels,title= "N = " *string(nb))
contour!(fig[1, 2],ys, xs,b ,levels=levels )
c=monte(xq,zq,nc,lambda,sigma)
contourf(fig[2, 1],xs, ys,c,levels=levels,title= "N = " *string(nc))
contour!(fig[2, 1],ys, xs,c ,levels=levels )
d=monte(xq,zq,nd,lambda,sigma)
contourf(fig[2, 2],xs, ys,d,levels=levels,title= "N = " *string(nd))
contour!(fig[2, 2],ys, xs,d ,levels=levels )



Colorbar(fig[1:2,3], limits = (0.1, 5), colormap = :viridis,
    flipaxis = false)

Label(fig[3, :], text = "x(m)")
Label(fig[:, 0], text = "z(m)")
Label(fig[0, :], text = "Konzentrationen im Vergleich fuer x = " *string(xq)
   *"m z = " *string(zq)*"m", textsize = 30)
save(
"Bericht/Bilder/3_vergleich_x = "*string(xq)*".png", fig)
end

function einzelmakie(xq,zq,lambda)
n=100
```

```
levels= [0.00001,0.01,1.0,2,10,100,200,500]#-1:0.1:1
fig = Figure(resolution=(1080,1080))
xs=LinRange(0, xgrenz,xgrenz)
ys=LinRange(0, zgrenz, zgrenz)

a=monte(xq,zq,n,lambda,1)
#al=maximum(a)
#println(al)
contourf(fig[1, 1],ys, xs,a ,levels=levels, xlabel = "x label", ylabel = "y label" )
contour!(fig[1, 1],ys, xs,a ,levels=levels )
Colorbar(fig[1,2], limits = (0, 5), colormap = :viridis,
    flipaxis = false)

Label(fig[0, :], text = "Konzentrationen im Vergleich fuer x = " *string(xq) *"m z = " *string(zq)*"m   ="*string(lambda), textsize = 30)
Label(fig[2, :], text = "x(m)")
Label(fig[:, 0], text = "z(m)")
save(
"Bericht/Bilder/3_single_x = "*string(xq)*"_"*string(lambda)*".png", fig)
end

function lambdamakie(xq,zq)
    xgrenz = 120  # !m
    zgrenz = 120
    la= 10
    lb= 5
    lc =2
    ld= 1
    n= 1000
    sigma=1
    levels= [0.00001,0.01,1.0,2,10,100,200,500]#-1:0.1:1#[0,0.01,0.025,0.05,0.5,0.75,1.0,1.25,1.5,2]#[0.001,0.005,0.01,0.05,0.1,0.5,0.75,1,2

     fig = Figure(resolution=(1080,1080))
     xs=LinRange(0, xgrenz,xgrenz)
     ys=LinRange(0, zgrenz, zgrenz)
    a=monte(xq,zq,n,la,sigma)
    contourf(fig[1, 1],ys, xs, a,levels=levels)
    contour!(fig[1, 1],ys, xs,a ,levels=levels )
    b=monte(xq,zq,n,lb,sigma)
    contourf(fig[1, 2],xs, ys, b,levels=levels)
    contour!(fig[1, 2],ys, xs,b ,levels=levels )
    c=monte(xq,zq,n,lc,sigma)
    contourf(fig[2, 1],xs, ys,c,levels=levels)
    contour!(fig[2, 1],ys, xs,c ,levels=levels )
    d=monte(xq,zq,n,ld,sigma)
    contourf(fig[2, 2],xs, ys,d,levels=levels)
    contour!(fig[2, 2],ys, xs,d ,levels=levels )
    Colorbar(fig[1:2,3], limits = (0.1, 5), colormap = :viridis,
        flipaxis = false)
    Label(fig[3, :], text = "x(m)")
    Label(fig[:, 0], text = "z(m)")
    Label(fig[0, :], text = "Konzentrationen im Vergleich fuer x = " *string(xq)*"m z = " *string(zq)*" m bei verschiedenen   ", textsize = 30)
    save(
    "Bericht/Bilder/3_lambda_x = "*string(xq)*".png", fig)
end
```

```
350
351 function sigmamakie(xq,zq)
352     l=10
353     n= 1000
354     siga=1
355     sigb=2
356     sigc=5
357     sigd=10
358     levels= [0.00001,0.01,1.0,2,10,100,200,500]#-1:0.1:1#
        [0,0.01,0.025,0.05,0.5,0.75,1.0,1.25,1.5,2]#[0.001,0.005,0.01,0.05,0.1,0.5,0.75,1,2
359      fig = Figure(resolution=(1080,1080))
360      xs=LinRange(0, xgrenz,xgrenz)
361      ys=LinRange(0, zgrenz, zgrenz)
362     a=monte(xq,zq,n,l,siga)
363     contourf(fig[1, 1],ys, xs, a,levels=levels)
364     contour!(fig[1, 1],ys, xs,a ,levels=levels )
365     b=monte(xq,zq,n,l,sigb)
366     contourf(fig[1, 2],xs, ys, b,levels=levels)
367     contour!(fig[1, 2],ys, xs,b ,levels=levels )
368     c=monte(xq,zq,n,l,sigc)
369     contourf(fig[2, 1],xs, ys,c,levels=levels)
370     contour!(fig[2, 1],ys, xs,c ,levels=levels )
371     d=monte(xq,zq,n,l,sigd)
372     contourf(fig[2, 2],xs, ys,d,levels=levels)
373     contour!(fig[2, 2],ys, xs,d ,levels=levels )
374     Colorbar(fig[1:2,3], limits = (0.1, 5), colormap = :viridis,
375         flipaxis = false)
376     Label(fig[3, :], text = "x(m)")
377     Label(fig[:, 0], text = "z(m)")
378     Label(fig[0, :], text = "Konzentrationen im Vergleich fuer x = " *string
        (xq)*"m z = " *string(zq)*" m bei verschiedenen  u  &  w ", textsize =
        30)
379     save(
380     "Bericht/Bilder/3_sigma_x = "*string(xq)*".png", fig)
381 end
382
383     function main()
384
385         ### Aufgbabe a
386         xq = 60.5  # !m
387         zq = 0.5 # !m
388         #vergleichmakie(xq,zq,5)
389         #lambdamakie(xq,zq)
390         sigmamakie(zq,xq)
391         ### Aufgbabe b
392         xq = 15.5  # !m
393         zq =  65.5 # !m
394         #vergleichmakie(xq,zq,5)
395         lambdamakie(xq,zq)
396         sigmamakie(zq,xq)
397
398
399     end
400     main()
```