Programmierpraktikum

Der Bericht

Alexander Steding
10028034
Gottfried Wilhelm Leibniz Universität
22. September 2022

Inhaltsverzeichnis

1 Einleitung

Dieser Bericht stellt als Abschlussbericht die Ergebnisse und Erkenntnisse des Programmierpraktikums Schadstoffausbreitung zusammen. Angefangen mit dem Vergleich zwischen Gauß-Modell und Monte-Carlo-Modell, weiter über die Validierung des Monte-Carlo-Modells und schließlich zur Anwendung des Modells in einer Straßenschlucht. Abschließend werden Gedanken über die dreidimensionale Schadstoffausbreitung in Stadtgebieten geteilt. Als Programmiersprache wurde über alle Aufgaben hinweg Julia verwendet und als Graphisches Backend PlotlyJS sowie GlMakie.

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 Aufgabenteil a wird zunächst ein Gauß-Modell für die Ausbreitung einer kontinuirliche Linienquelle programmiert mit Vernachlässigung der Prandtlschicht. Hier gilt:

$$x = 51\text{m} \tag{1}$$
$$z = 45\text{m} \tag{2}$$
$$q = 540\frac{\text{kg}}{\text{h}} \tag{3}$$
$$\tau_l = 100\text{s} \tag{4}$$
$$\sigma_w = 0.39\frac{\text{m}}{\text{s}} \tag{5}$$
$$\tag{6}$$

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

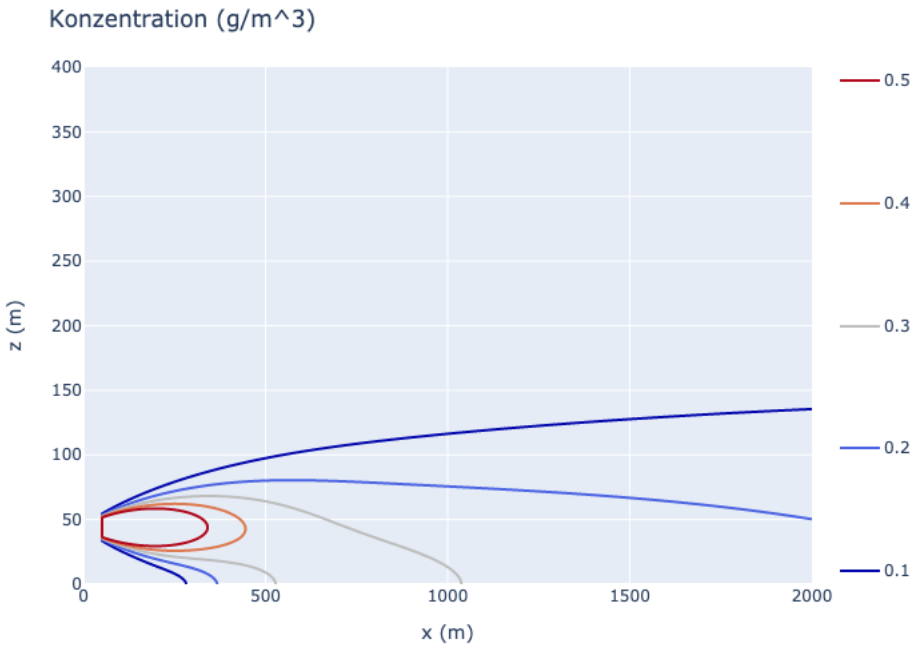

Abb. 1: Konzentrationsverteilung als Contour-Plot. Die Konzentration ist in $\frac{\mathrm{g}}{\mathrm{m}^3}$ dargestellt.

Für eine feinere grapische Analyse wurde in Abbildung 2. die Konzentrationsverteilung am Erdboden, also für $z = 0\mathrm{m}$, visualisiert.

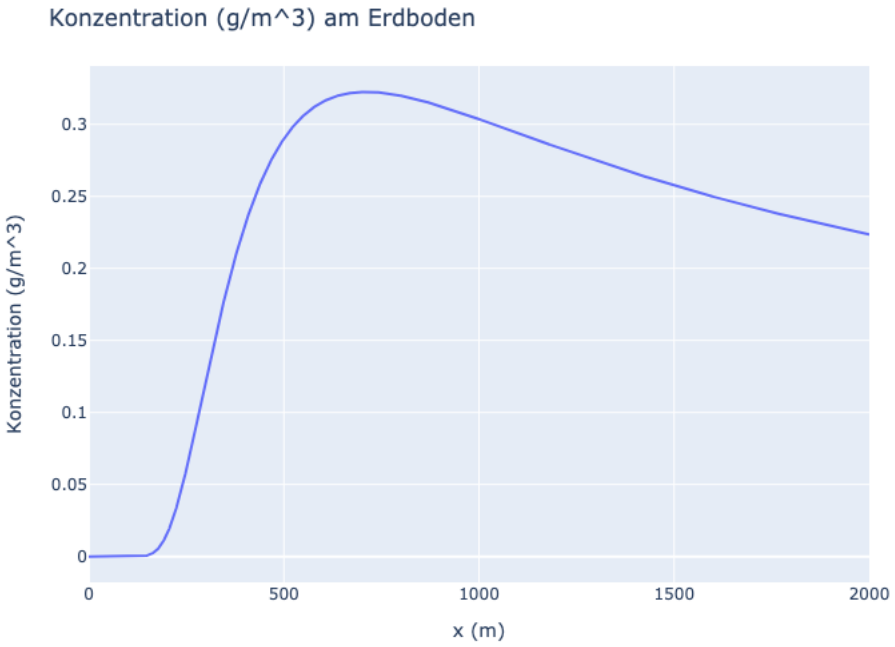

Abb. 2: Konzentrationsverteilung am Erdboden. Die Konzentration ist in $\frac{\mathrm{g}}{\mathrm{m}^3}$ dargestellt.

Die maximale Konzentration wurde final rechnerisch mittels Julia bestimmt als

$$c[0,711] = 0,32263\frac{g}{m^3} \tag{7}$$

.

2.2 Aufgabenteil b

In Aufgabenteil b wird nun das Monte-Carlo-Modell mit einer genährten Gitterauswertung verwendet. 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 simuliert.

Zunächst für $N = 1000$

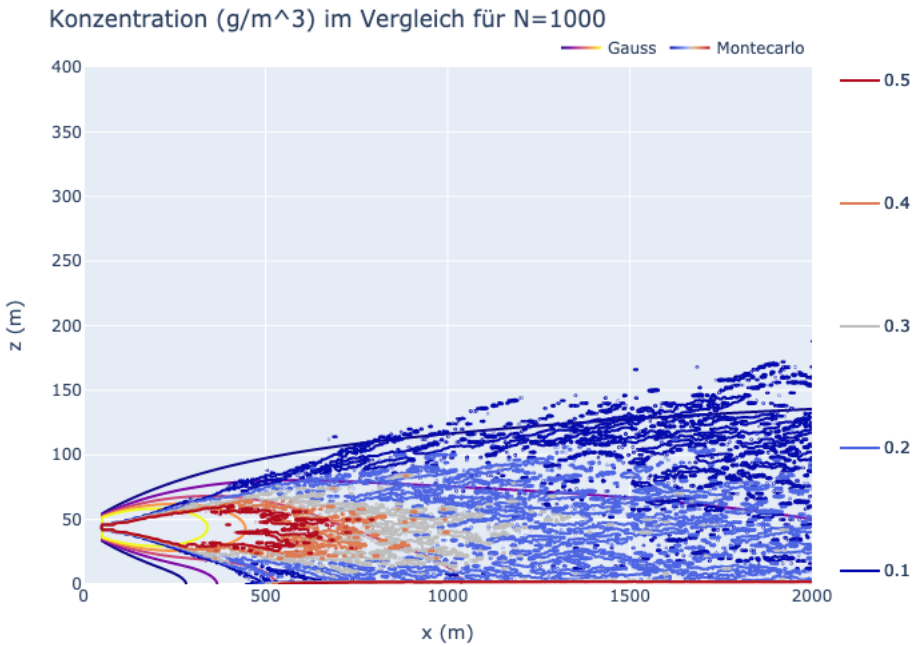

Abb. 3: Vergleich für N=1000. Die Konzentration ist in $\frac{g}{m^3}$ dargestellt.

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

$$N = 1000 \tag{8}$$

sind extrem große Abweichungen zum Gauß-Modell zu erkennen, obwohl auch hier bereits die Grundstruktur erkennbar ist.

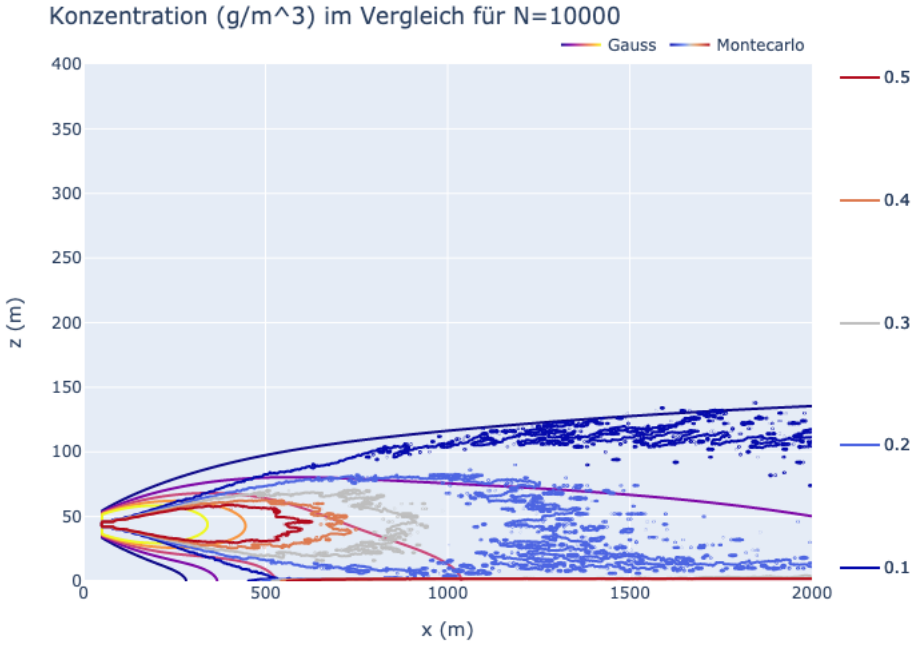

Abb. 4: Vergleich für N=10000. Die Konzentration ist in $\frac{g}{m^3}$ dargestellt.

Bei einer mittleren Anzahl wie

$$N = 10000 \tag{9}$$

gleicht sich das Monte-Carlo-Modell bedeutend besser an das Gauß-Modell an.

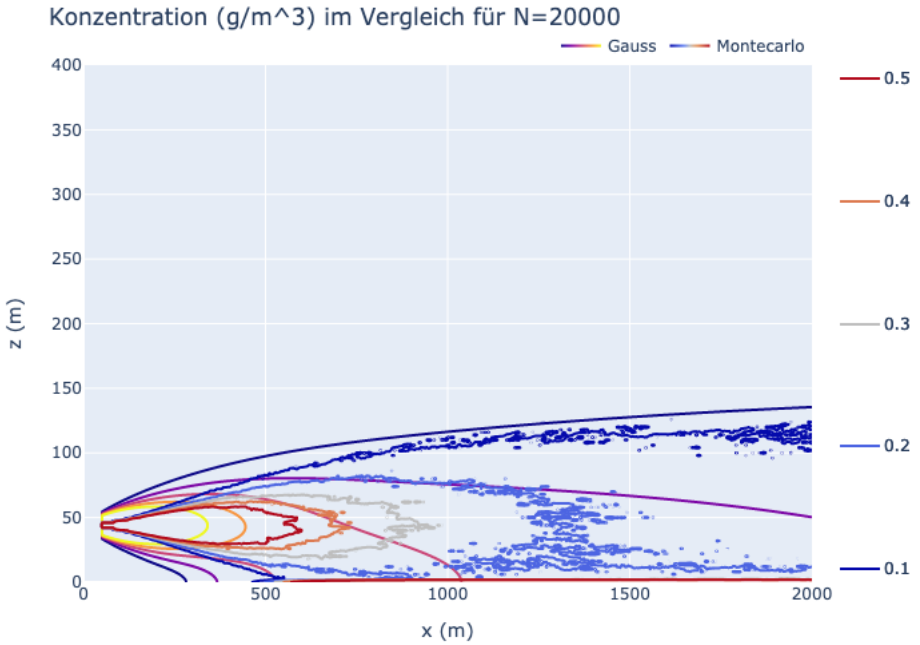

Abb. 5: Vergleich für N=20000. Die Konzentration ist in $\frac{g}{m^3}$ dargestellt.

Bei einer hohen Anzahl wie

$$N = 20000 \tag{10}$$

gleicht sich das MC Modell sehr gut an das Gauß-Modell an.

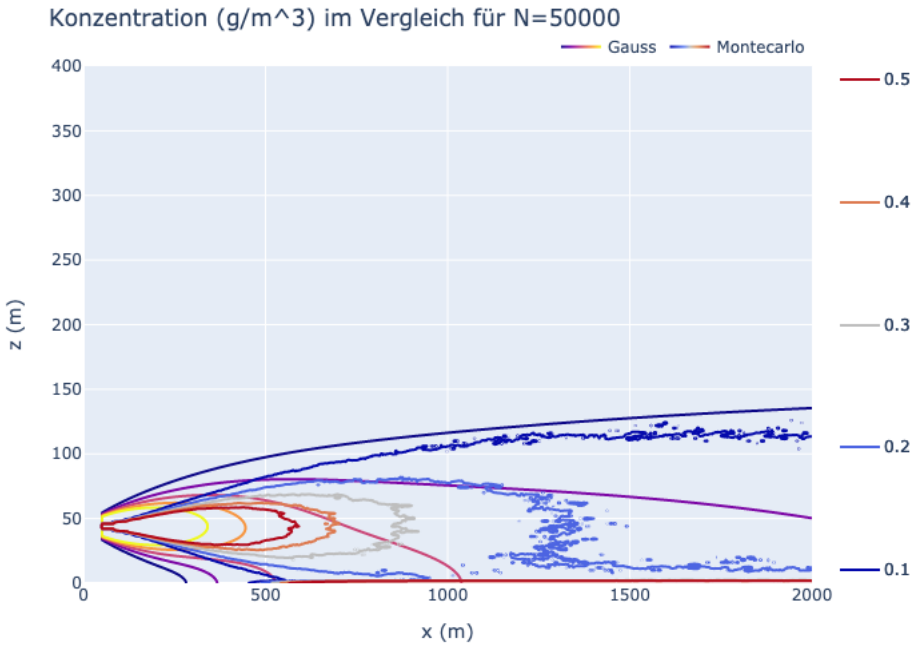

Abb. 6: Vergleich für N=50000. Die Konzentration ist in $\frac{g}{m^3}$ dargestellt.

Abbildung 6 zeigt das mit meiner Hardware maximal mögliche Ergebnis. Jenseits von dieser Anzahl sind zwar noch leichte Verbesserungen zu erwarten, nichts destotrotz zeigt sich bei

$$N = 50000 \tag{11}$$

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

$$N = 20000 \tag{12}$$

.

Die größten Unterschiede zwischen dem Monte-Carlo-Modell und dem Gauß-Modell enstehen bei zunehmender Entfernung von der Quelle, sowie in der Nähe des Quellortes selbst. Auch im Bereich bis 500 m in X Richtung ist die Konzentration beim Monte-Carlo-Modell niedrieger als beim Gauß-Modell, allen voran aber fehlt hier die Variation der Konzentration, die mit dem Gauß-Modell aufgelöst werden kann.

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. Für die Valiedierung wurde ein grafischer Vergleich gewählt. Dazu wurde jeweils die Z-Achse vom Prairie-Grass-Experiment

sowie des Monte-Carlo-Modells mit der entsprechenden Konzentration in einem Plot dargestellt. Als Randbedingungen sollen hier gelten:

$$x = 0\text{m} \tag{13}$$
$$z = 0.5\text{m} \tag{14}$$
$$z_0 = 0.008\text{m} \tag{15}$$
$$u_* = 0.35\frac{\text{m}}{\text{s}} \tag{16}$$
$$\kappa = 0.38 \tag{17}$$

Im folgenden ist die Konzentrationsverteilung bei $x = 100\text{m}$ dargestellt.

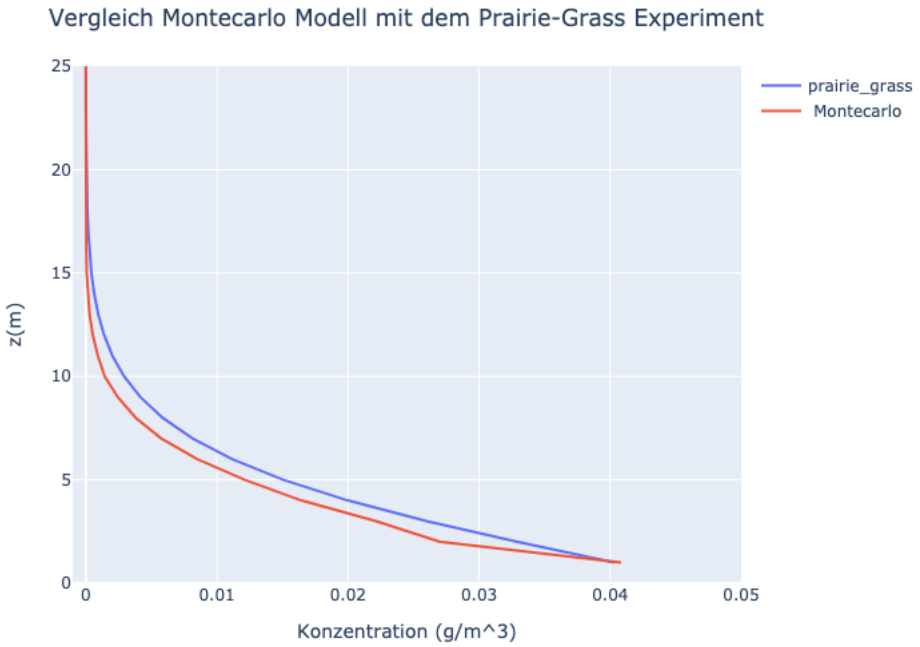

Abb. 7: Vergleich Prairie-Grass, Monte-Carlo. Die Konzentration ist in $\frac{\text{g}}{\text{m}^3}$ dargestellt.

Rein Qualitativ betrachtet ist der Verlauf des Monte-Carlo-Modells dem des Prairie-Grass-Experimentes extrem ähnlich. Insbesondere für $z > 15\text{m}$ ist der Unterschied fast verschwindend gering. Die Größte Abweichung bei einer rein qualitativen Betrachtung liegt in Bodennähe bei $z < 2\text{m}$ zu finden. Trotzdem ist über den gesammten Verlauf der Höhe eine Abweichung von etwa $\Delta c = 0.0025\frac{\text{g}}{\text{m}^3}$ zu beobachten. Zum Erdboden direkt hin konvergieren die beiden Konzentrationen erneut gegeneinader so dass die Unterschiede an dieser Stelle nahezu verschwinden.

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 mit dem Modell PALM simulierten Windgeschwindigkeiten sowie deren Standartabweichung importiert und ein Contourprofil

aus der berechneten Konzentration gebildet. Für die Berechnung von τ_l wurde der Prandtl-Kolmorov-Ansatz verwendet. Aufgrund der besseren Optimierung wurde für die komplexeren Berechnungen in dieser Aufgabe statt PlotlyJS GlMakie als Visualiesierungs-Backend verwendet. Im ersten Szenario wurde ein Schadstoffquelle im Straßenverkehr mit Quellort

$$x = 60.5\text{m} \tag{18}$$
$$z = 0.5\text{m} \tag{19}$$

simuliert.

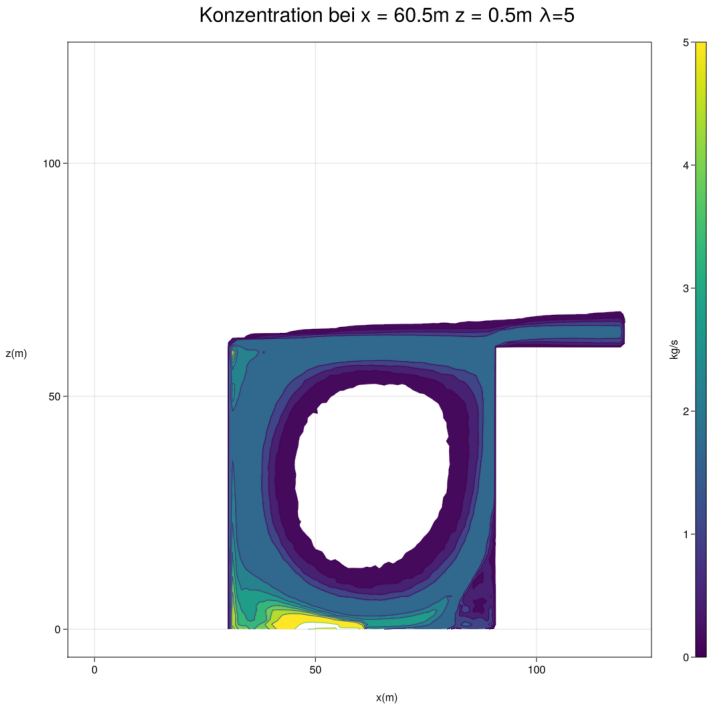

Abb. 8: Konzentrationsverteilung am Erdboden. Die Konzentration ist in $\frac{\text{kg}}{\text{s}}$ dargestellt. N=1000, $\lambda = 5$

In der Straßenschlucht bildet sich durch den Wind über der Schlucht ein Wirbel aus. In diesem sinkt die Konzentration mit abnehmendem Abstand zum Zentrum, sowie tendenziel mit der Höhe. Besonders am Boden der Lee-Seite der Straßenschlucht kommt es zu einer extrem starken Ansammlung der Schadstoffe. Teilweise sind dort um den Faktor 500 höhere Konzentrationen simuliert worden als im Zentrum des Wirbels. Dies kann zu je nach Schadtstoff zu negativen gesundheitlichen Folgen für Fußgänger*innen, Radfahrer*innen, Anwohner*innen, welche sich in der Nähe aufhalten führen.

Im zweiten Szenario wurde als Schadstoffquelle ein Hausbrand mit Quellort

$$x = 15.5\text{m} \tag{20}$$
$$z = 65.5\text{m} \tag{21}$$

simuliert.

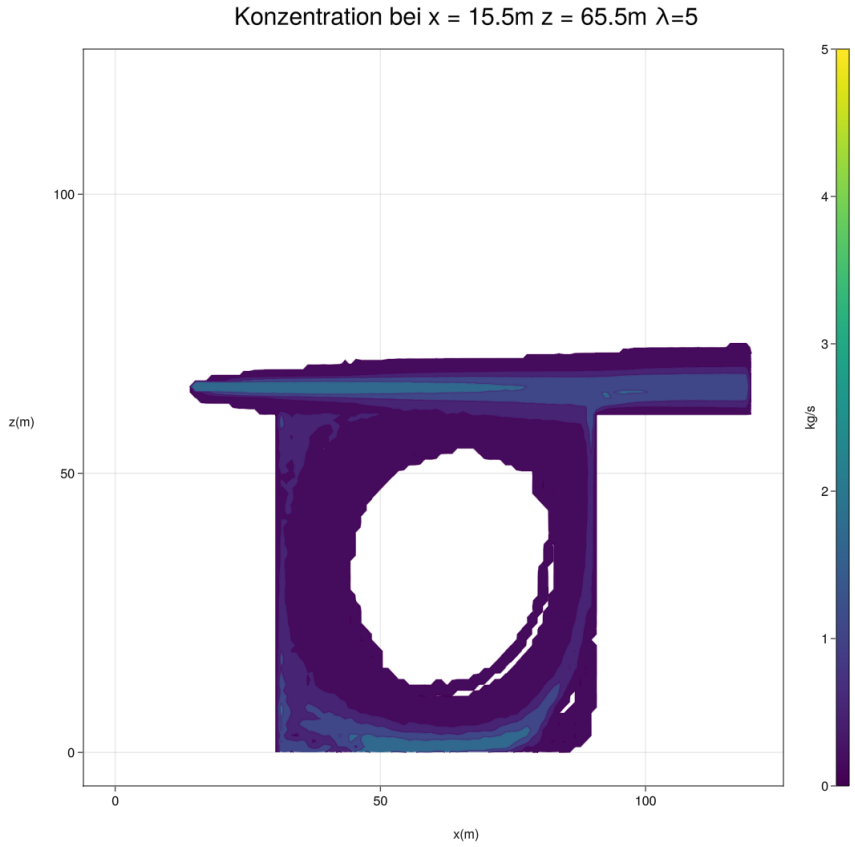

Abb. 9: Konzentrationsverteilung am Erdboden. Die Konzentration ist in $\frac{\text{kg}}{\text{s}}$ dargestellt.N=1000, $\lambda = 5$

Abseits der offensichtlichen Änderung durch den neuen Quellort über den Häusern, zeigt sich ein nahezu ähnliches Bild wie im ersten Szenario. Erneut befindet sich der Ort der größten Konzentration an Schadstoffen in Bodennähe. Ein signifikanter Unterschied ist jedoch die quantitative Konzentrationshöhe am Erdboden. Diese liegt mit $c = 2,5\frac{\text{kg}}{\text{s}}$ nur noch halb so hoch wie im vorherigen Fall. Die allgemein geringere Konzentration lässt sich auf die Quellposition über der Häuserschlucht zurückführen, wodruch nur noch ein Teil der Partikel in den Wirbel gelangen kann.

4.1 Einfluss der Teilchenanzahl

Desweiteren soll nun der Einfluss der Teilchenanzahl analysiert werden. Dazu wurde für jedes Szenario ein Plot, welcher vier Contourplots für unterschiedliche Partikelanzahlen im direkten Vergleich visualisiert gewählt.

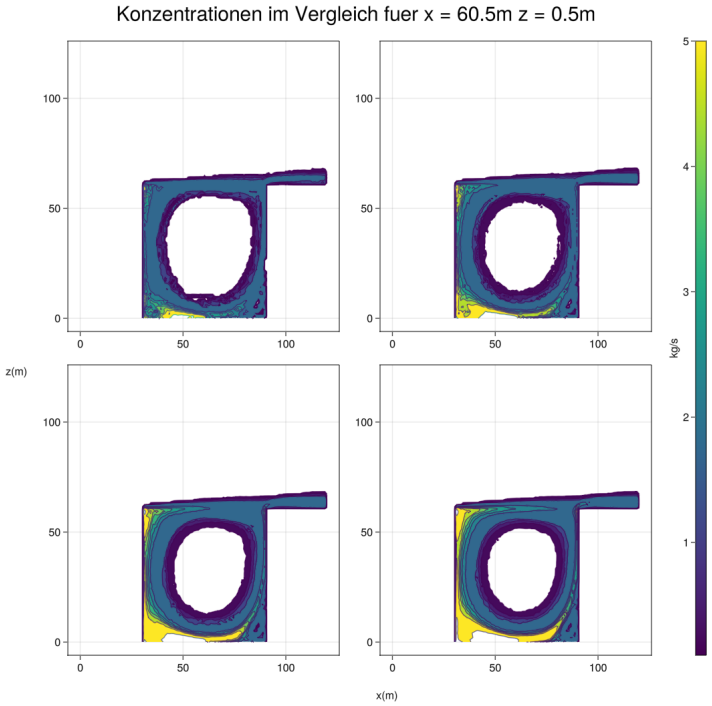

Abb. 10: Verschiedene Konzentrationsverteilung im Vergleich. $N \in [10^2, 10^3, 5 * 10^3, 10^4]$ v.L.n.R. $\lambda = 5$

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.

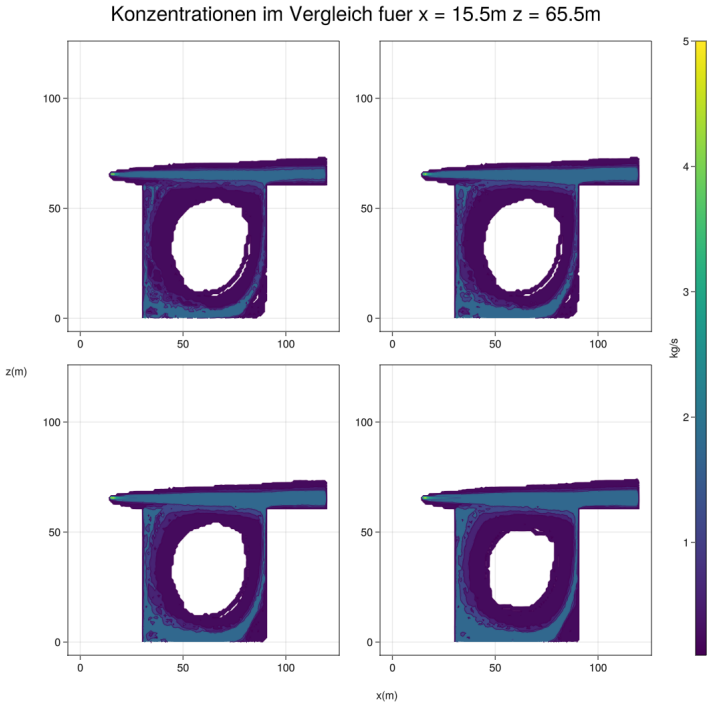

Abb. 11: Verschiedene Konzentrationsverteilung im Vergleich. $N \in [10^2, 10^3, 5 * 10^3, 10^4]$v.L.n.R, $\lambda = 5$

Für die in Aufgabenteil b) geschilderte Situation zeichnet sich ein leicht anderer 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 $N = 1000$ weiterhin zu einer deutlichen Verbesserung. Da aufgrund der Quellposition bei diesem Szenario weniger Partikel in den Wirbel gelangen, muss für eine repräsentative Darstellung eine höhere Teilchenzahl als bei Situtation a) verwendet werden.

4.2 Einfluss von λ

Auch der Einfluss von λ aus dem Prandt-Kolmogorov-Ansatz soll nun analog betrachtet werden.

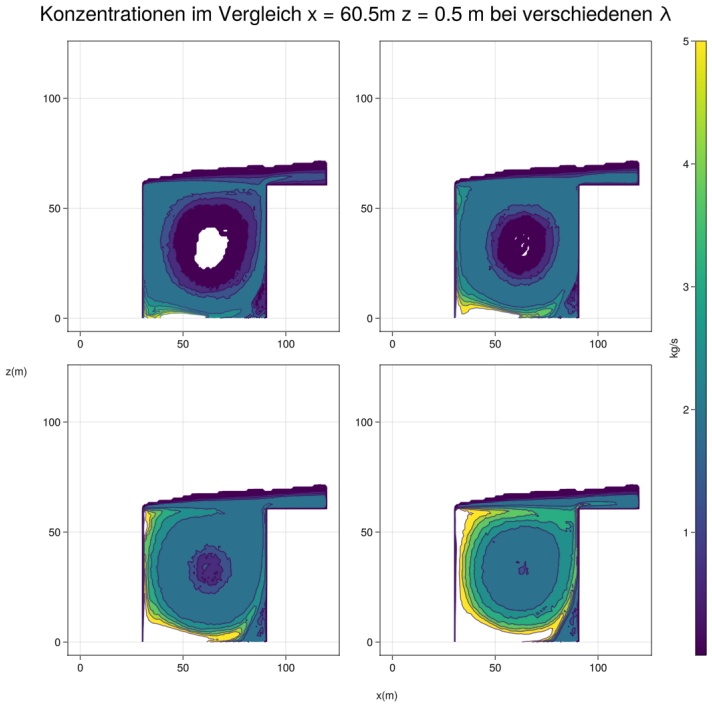

Abb. 12: Verschiedene λ im Vergleich. $\lambda \in [10, 5, 2, 1]$ v.L.n.R, N=1000

Da es sich bei λ um eiine Komponente des Mischungsweg handelt, führt eine Vergrößerung dieses zu einer Vergößerung des Wirbels, et vicae versa. Bei kleinem Mischungsweg kommt es daher in einem deutlich erhöhtem Bereich zu einer Ansammlung gefährlicher Konzentrationen, bei einem großen Mischungsweg zu einer örtlich beschränkteren Ausbreitung.

4.3 Einfluss von σ

Als letztes soll nun der Einfluss von σ_u und $\sigma_w$analog betrachtet werden. Es wurden jeweils beide Standartabweichungen simultan verändert.

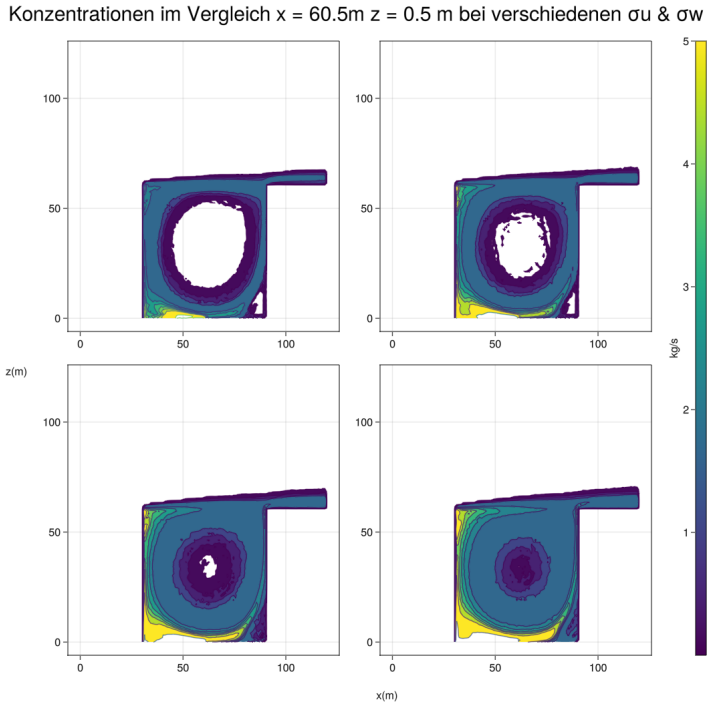

Abb. 13: Verschiedene σ im Vergleich. $\sigma \in [1, 2, 5, 10]$ v.L.n.R N=1000, $\lambda = 5$

Eine Veränderung der Standartabweichung hat in diesem Simulations-Modell direkten Einfluss auf den turbolenten Anteil der Geschwindigkeitskomponente. Eine Zunahme der Standartabweichung führt folglich zu einem stärker ausgeprägtem Wirbel, eine Reduzierung zu einem schwächerem. Analog dazu verhällt sich aufgrund der varierenden Diffusionsstärke jeweils die Konzentration. Wodurch bei einem hohen σ eine erhöhte Konzentration auch an den Häuserwänden zu beobachten ist.

5 Aufgabe 4

Bei einer Ausbreitung in einem dreidmensionalem Stadtgebiet müssten neben der zusätlichen räumlichen Ausbreitungskoordinate, auch die neue Gitterdimension, sowie die Windkomponenten angepasst und erweitert werden. Auch muss überlegt werden ob statt dem Mischungsweg der turbolente Diffusionskoeffitient als multidimensionale Größe verwendet werden kann. Dieser könnte möglicherweise aus dem Strömungsmodell stammen.

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
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
dx = 1               # Gitterweite x Richtung in m
dz = 1               # Gitterweite z Richtung in m
ustern = 0.35        # Schubspannungsgeschwindigkeit in m/s
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


#Array wird in der Gr   e xgrenz*zgrenz aufgespannt und mit 0 gef llt
konk=zeros(xgrenz,zgrenz)



# Montecarlo-Modell #

##Funktionen zur Berechnung des Montecarlo Modell ##

### Berechnung der Prandtlschicht###
function prandltl(zi,xi)
    #### Berechnung der mittleren Windgeschwindigkeit ####
    if zi < znull
        ubalken = 0 #kein mittlerer Wind f r zi < znull
    else
        ubalken = (ustern / k) * log(abs(zi) / znull) # Log. Windprofil
    end
    ##Berechnung des Lagrangeschen Zeitschrittes tl
    tl = ((k * ustern) / sigw ^ 2) * abs(zi)

    #### Berechnung des Zeitschrittes dt
    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, ubalken
end


### Berechnung der Positionen###
function positionen(xi, wi, zi,tl, ui, dt)
    #Lagrangesche Autokorellationsfunktion
    rl = exp(- dt / tl)
    Random.seed!()
    d = Normal()
    rr = rand(d, 1)[1]

    xi = xi + ui * dt
    wi=rl*wi + sqrt((1 - rl^2))*sigw* rr
    zi = zi + wi * dt
return xi, wi, zi

end
### Gitterauswertung ###
function exaktgitter(xi, xold, zi, zold,dt)
    #Initialisierung
    ti = []
    toks = []
    # Berechnung der Anzahl an Gitterschnitpunkten
    rangex = convert(Int64,floor(abs(xi - xold)))
    rangez = convert(Int64,floor(abs(zi - zold)))

    # Fallunterscheidung: Gibt es Schnittpunkte
    if rangex >0 && rangez>0
    #Falls es sie gibt muss die Konzentration pro Anteil Giterpunkt berechnet werden
        # Berechnung von ti und tj
        for i in 0:rangex
            if xold >xi
            xsi=ceil(xold) +i
            else
            xsi=ceil(xold) -i
            end
            push!(toks,(xsi - xold) / abs(xi - xold)) # Einh ngen in die Liste
        end
        for i in 0:rangez
            if zold >zi
                zsi=ceil(zold) +i
                else
                zsi=ceil(zold) -i
                end
            push!(toks,(zsi - zold) / abs(zi - zold)) # Einh ngen in die Liste
        end
        tku = sort!(toks) #Sortieren der Liste
        for i in 2:length(tku)
            ti = tku[i]
            told = tku[i - 1]
```

```
108             t = mean([told, ti])
109             # Der Mittelwert t der beiden Punkte told und ti kann in einer
    Geradengleichung verwendet werden
110             # So k nnen die Betroffenen Gitterpunkte ermittelt werden
111             posx, posz = gg(xold, zold, xi, zi, t) #Ausf hren der
    Geradengleichung
112             #Anschlie end wird die konzentrationen am mit der GG
    berechneten Punkt um den entsprechenden Betrag erh ht
113             konk[abs(posx), abs(posz)] += ((tku[i] - tku[i-1])* dt*((q * dt)
    /(n * dx * dz)))
114         end
115 else
116     #Gitterauswertung falls Partikel in dem Giterpunkt verbleibt
117     konk[convert(Int64,ceil(xi) ),convert(Int64,ceil(zi))] +=  ((q * dt)/(n
    * dx * dz)) #Erh hung um die gesammte Konzentration
118 end
119 end
120
121 ### Geradengleichung zur exakten Gitterauswertung ###
122 function gg(xold, zold, xi, zi, t)
123     xg = xold + t * (xi - xold)
124     zg = zold + t * (zi - zold)
125     if xg>xgrenz
126         xg=xgrenz
127     end
128     #Die Geradengleichung kann negative Werte liefern.
129     #Um einen Abbruch zu verhindern werden zg und xg in diesem Fall auf 1
    gesetzt
130     if floor(zg) <1
131         zg=1
132     end
133     if floor(xg) <1
134         xg=1
135     end
136     return convert(Int64, floor(xg)), convert(Int64, floor(zg)) #
    Zur ckgeben der Werte als Integer
137 end
138
139 ### Berechnung des Montecarlo-Modells ###
140 function monte()
141     for i in ProgressBar(1:n+1) #Loop  ber  alle Partikel
142         #Initialisieren der Parameter f  r jeden Partikel
143         xi=xq
144         zi=zq
145         ui=ubalken
146         wi=wbalken
147         dt=0
148         while (ceil(xi+ui*dt) < xgrenz)
149             xold=xi #Zwischenspeichern der alten Werte f  r x
150             zold=zi #Zwischenspeichern der alten Werte f  r y
151             if zi<znull #Ber cksichtigen der Totalreflexion
152                 zi= zi+ 2*abs(zi-znull) #Reflexion
153                 wi= -wi
154                 tl, dt,ui = prandltl(zi,xi) #Berechnung der Prandtlschicht
155                 xi,wi,zi =positionen(xi, wi, zi,tl, ui, dt) #Berechnung der
    neuen Positionen
156                 exaktgitter(xi, xold, zi, zold,dt) #Exakte Gitterauswertung
157             else
```

```
                tl, dt,ui = prandltl(zi,xi) #Berechnung der Prandtlschicht
                xi,wi,zi =positionen(xi, wi, zi,tl, ui, dt) #Berechnung der neuen Positionen
                exaktgitter(xi, xold, zi, zold,dt) #Exakte Gitterauswertung
            end
        end

    end
    return konk #Zur ckgeben der Konzentration als Feld
end

# Prairie-Grass-Experiments #
function prairie_grass(konk)
    pg_mod= [] #Initialisieren
    #Ermitteln der ben tigten Werte aus dem Montecarlo-Modells
    for i in 100:xgrenz
        for j in 1:zgrenz
            push!(pg_mod, konk[i,j])
        end
    end
    c0 = 4.63E-02
    gamma = 0.68
    my = 1.3
    zs = 3.4
    z= collect(1:zgrenz)
    pg = zeros(length(z)+1)
for k in 1:zgrenz
   pg[k] = c0 * exp(-gamma * (z[k]/zs)^my)
end
return pg, pg_mod
end

### Visualisierung ###

 function grafen(pg,pg_mod)
    #Darstellung mit PlotlyJS
    title="Vergleich Montecarlo Modell mit dem Prairie-Grass Experiment"
    xaxis_title="Konzentration (" * units * ")"
    plot_prairie=scatter(y=collect(1:zgrenz),x=pg,name="prairie_grass", showlegend=true ,)
    plot_monte= scatter(y=collect(1:zgrenz),x=pg_mod,name=" Montecarlo", showlegend=true ,)
    savefig(plot([plot_prairie,plot_monte],Layout(title= title, xaxis_title= xaxis_title, yaxis_title="z(m)",xaxis_range=[-0.001, 0.05], yaxis_range =[0, 25])), "Bericht/Bilder/2.png")
 end



 ### Main ####
function main()
    #Zun chst wird das Montecarlo-Modell berechnet
    konzentrationen=monte() #Die Konzentration wird in einem Array zwischengespeichert
    #Als n chstes wird das prairie_grass-Experiment mit der berechneten Konzentration durchgef hrt
```

```
208     pg,pg_mod=prairie_grass(konzentrationen) #Die Arrays werden
    zwischengespeichert
209     # Als letztes kommt es zur Visualisierung
210     grafen(pg,pg_mod)
211
212 end
213
214
215 main() #Ausf hren der Main Funktion
```

6.2 Aufgabe 3

```
1  using NetCDF
2  using Random, Distributions
3  using ProgressBars
4  using LinearAlgebra
5  using GLMakie
6  using SymPy
7  using FileIO
8
9  # Globale Variablen
10 xgrenz = 120         # Grenze des Modells in x Richtung in m
11 zgrenz = 120         # Grenze des Modells in z Richtung in m
12 r = symbols("r")     # Wird ben tigt f r die Eckenreflexion
13 dx = 1               # Gitterweite in x-Richtung in m
14 dz = 1               # Gitterweite in z-Richtung in m
15 k = 0.38             # Karman-Konstante
16 q = 1                # Anfangskonzentration in kg/s
17
18 ## Arrays Initialisieren
19
20 #Array wird in der Gr   e xgrenz*zgrenz aufgespannt und mit 0 gef llt
21 konk=zeros(xgrenz,zgrenz)
22 # Montecarlo-Modell #
23 ##Funktionen zur Berechnung des Montecarlo Modell ##
24
25 ### Berechnung der Prandtlschicht###
26 function prandltl(zi,xi,lambda,us,ws,k)
27     xii=convert(Int64,floor(xi))
28     zii=convert(Int64,floor(zi))
29     if floor(zi)==0
30     zii=1
31     end
32     #Berechnung des Lagrangeschen Zeitschrittes tl
33     tl= 0.05*((k*zii)/(1+k*(zii/lambda)))/(0.23*sqrt(us[xii+1,zii+1]+ws[xii
    +1,zii+1]))
34
35     #### Berechnung des Zeitschrittes dt
36     if (0.1*tl)>0.05*((k*2)/(1+k*(2/lambda)))/(0.23*sqrt(us[xii+1,zii+1]+ws[
    xii+1,zii+1]))  #falls dt kleiner als tl in 2 m Hoehe
37        dt = 0.1*tl
                                  # Normalfall
38     else
39         dt = 0.05*((k*2)/(1+k*(2/lambda)))/(0.23*sqrt(us[xii+1,zii+1]+ws[xii
    +1,zii+1]))      #falls dt kleiner als tl in 2 m Hoehe wird dt auf tl(2m)
    gesetzt
40     end
41     return tl,  dt
```

```
end

### Gitterauswertung ###
function exaktgitter(xi, xold, zi, zold,dt,n)
    #Initialisierung
    ti = []
    toks = []
    # Berechnung der Anzahl an Gitterschnitpunkten
    rangex = convert(Int64,floor(abs(xi - xold)))
    rangez = convert(Int64,floor(abs(zi - zold)))

    # Fallunterscheidung: Gibt es Schnittpunkte
    if rangex >0 && rangez>0
    #Falls es sie gibt muss die Konzentration pro Anteil Giterpunkt berechnet werden
        # Berechnung von ti und tj
        for i in 0:rangex
            if xold >xi
            xsi=ceil(xold) +i
            else
            xsi=ceil(xold) -i
            end
            push!(toks,(xsi - xold) / abs(xi - xold)) # Einh ngen in die Liste
        end
        for i in 0:rangez
            if zold >zi
                zsi=ceil(zold) +i
                else
                zsi=ceil(zold) -i
                end
            push!(toks,(zsi - zold) / abs(zi - zold)) # Einh ngen in die Liste
        end
        tku = sort!(toks) #Sortieren der Liste
        for i in 2:length(tku)
            ti = tku[i]
            told = tku[i - 1]
            t = mean([told, ti])
            # Der Mittelwert t der beiden Punkte told und ti kann in einer Geradengleichung verwendet werden
            # So k nnen die Betroffenen Gitterpunkte ermittelt werden
            posx, posz = gg(xold, zold, xi, zi, t) #Ausf hren der Geradengleichung
            #Anschlie end wird die konzentrationen am mit der GG berechneten Punkt um den entsprechenden Betrag erh ht
            konk[abs(posx), abs(posz)] += ((tku[i] - tku[i-1])* dt*((q * dt) /(n * dx * dz)))
        end
else
    #Gitterauswertung falls Partikel in dem Giterpunkt verbleibt
    konk[convert(Int64,ceil(xi) ),convert(Int64,ceil(zi))] +=  ((q * dt)/(n * dx * dz)) #Erh hung um die gesammte Konzentration
end
end

### Berechnung der Geradengleichung ###
function gg(xold, zold, xi, zi, t)
```

```
92      xg = xold + t * abs(xi - xold)
93      zg = zold + t * abs(zi - zold)
94
95      #Durch die Geradengleichung k nnen Gitterpunkte jenseits der Grenzen
        berechnet werden
96      #Um einen Abbruch zu verhindern werden in disem Fall die Maximalwerte
        verwendet
97      if xg>xgrenz
98          xg=xgrenz
99      end
100     if zg>zgrenz
101         zg=zgrenz
102     end
103     return convert(Int64, floor(xg)), convert(Int64, floor(zg)) #
        Zur ckgeben der Werte als Int
104 end
105
106 ### Funktion zur Berechnung der Positionen ###
107 function positionen(xi, wi, zi, tl, ui, dt,xold,zold,xolder,zolder,us,ws,u,w
        )
108     #Da Julia mit einer Indexierung ab 1 startet m ssen die Indexe der
        Parameter angepasst werden
109     ixolder= floor(xolder )+1
110     izolder=floor(zolder )+1
111     ixold= floor(xold )+1
112     izold= floor(zold )+1
113     izi= floor(zi )+1
114     ixi= floor(xi)+1
115     #Lagrangesche Autokorellationsfunktion
116     rl = exp(- dt / tl)
117     Random.seed!()
118     d = Normal()
119     rr = rand(d, 1)[1]
120     # Abfangen von m glichen Grenzf llen f r die Indexe
121     if izi == 1
122         izi = 2
123     end
124     if ixi == 91
125         ixi=90
126     end
127     if ixold== 91
128         ixold=90
129     end
130     if izold== 1
131         izold=2
132     end
133
134     # Zentraler Diffentialquotient
135     difqu= abs(us[abs(convert(Int64,(ixolder))),abs(convert(Int64,(izolder))
        )]-us[abs(convert(Int64,(ixi))),abs(convert(Int64,(izi)))])/ 2* abs(
        xolder-xi)
136     difqw=abs(ws[abs(convert(Int64,(ixolder))),abs(convert(Int64,(izolder)))
        ]-ws[abs(convert(Int64,(ixi))),abs(convert(Int64,(izi)))])/ 2*abs(zolder-
        zi)
137
138     ## Turbulenter Anteil
139     uturbo = rl * ui + sqrt((1 - rl ^ 2)) * sqrt(us[abs(convert(Int64,(ixi))
        ),abs(convert(Int64,(izi)))])*rr +(1-rl)*tl*difqu
```

```
    wturbo = rl * wi + sqrt((1 - rl ^ 2)) * sqrt(ws[abs(convert(Int64,(ixi)),abs(convert(Int64,(izi)))])*rr +(1-rl)*tl*difqw

    #Berechnung der Windgeschwindigkeit (Mittelwert + Turbulenz)
    ui = u[abs(convert(Int64,(ixold))),abs(convert(Int64,(izold)))] +uturbo
    wi = w[abs(convert(Int64,(ixold))),abs(convert(Int64,(izold)))] +wturbo

    #Berechnung der neuen Positionen
    xi = xi + ui * dt
    zi = zi + wi * dt

    #Reflexion    berprfen
    while ((xi<=31 && zi<=61)||(xi>=90 && zi <=61)||(30<=xi<=90 && zi<=0))
        if (xi>=90 && zi<=61) # Reflexion an der rechten Wand
            eckenreflexion(xi,zi,xold,zold,ui,wi)
            if br==1 # Falls dies der Fall ist gab es keine Eckenreflexion
                xi= xi-2*(abs(90-xi))
                ui=-ui
            end
        elseif (xi<=31 && zi<=61)  # Reflexion an der linken Wand
            eckenreflexion(xi,zi,xold,zold,ui,wi)
            if br==1 # Falls dies der Fall ist gab es keine Eckenreflexion
                xi=xi+2*(abs(31-xi))
                ui=-ui
            end
        elseif (30<=xi<=90 && zi<=0)  #Reflexion am Boden
            eckenreflexion(xi,zi,xold,zold,ui,wi)
            if br==1 # Falls dies der Fall ist gab es keine Eckenreflexion
            wi=-wi
            zi= -zi
            end
        else
            print("Fehler in der Reflexion")
        end


end
return xi, wi, zi,ui

end


### Funktion zur    berprfung    der Eckenreflexion ###
function  eckenreflexion(xi,zi,xold,zold,ui,wi)
    #Hier wird die Reflexion an den Ecken    berprft   .
    #Falls es eine Eckenreflexion gibt muss eine der folgenden Gleichungen l sbar sein, wodurch sich deren Typ ver ndert
    #Reflexion an der linken oberen Ecke
    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}
        ui=-ui
        wi=-wi
        xi=xold
        zi=zold
        br=2
    #Reflexion an der linken unteren Ecke
    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}
```

```
194             ui=-ui
195             wi=-wi
196             xi=xold
197             zi=zold
198             br=2
199     #Reflexion an der rechten unteren Ecke
200     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}
201             ui=-ui
202             wi=-wi
203             xi=xold
204             zi=zold
205             br=2
206     #Reflexion an der rechten oberen Ecke
207     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}
208         ui=-ui
209         wi=-wi
210         xi=xold
211         zi=zold
212         br=2
213 else
214     ui=ui
215     wi=wi
216     br=1
217 end
218 return ui,wi, br
219 end
220
221 ###   bergreifende   Funktion zur Berechnung des Monte-Carlo-Modells ###
222 function monte(xq,zq,n,lambda,sigma)
223
224 # Einlesen der Parameter aus der externen NC-Datei
225     u =  ncread("Bericht/input_uebung5.nc", "u")
226     w =  ncread("Bericht/input_uebung5.nc","w")
227     us = sigma*ncread("Bericht/input_uebung5.nc","u2")
228     ws = sigma*ncread("Bericht/input_uebung5.nc","w2")
229     #Ersetzen von den Fehlenden Werten mit NaN
230     templist  = findall(x->x==-9999.0,u)
231     templistw = findall(x->x==-9999.0,w)
232     for i in 1: length(templist)
233     u[templist[i]]=NaN
234     end
235     for i in 1: length(templistw)
236         w[templistw[i]]=NaN
237     end
238
239 #Loop  ber  alle Partikel
240     for i in ProgressBar(1:n)
241         #Initialisieren aller Parameter f  r den n chsten Partikel
242         xi = xq
243         zi = zq
244         dt=0
245         ui=0
246         wi=0
247         xold=xq
248         zold=zq
249         while (ceil(xi+ui*dt) < xgrenz)
```

```
            #Abspeichern der alten Werte
            xolder=xold
            zolder=zold
            xold = xi
            zold = zi
            #Berechnung der Prandtlschicht
            tl, dt = prandltl(zi,xi,lambda,us,ws,k)
            #Berechnung der neuen Positionen
            xi, wi, zi,ui = positionen(xi, wi, zi, tl, ui, dt,xold,zold,
    xolder,zolder,us,ws,u,w)
            # Gitterauswertung der Konzentration
            exaktgitter(xi, xold, zi, zold,dt,n)
        end

    end
    return  konk #Zur ckgeben der Konzentration als Feld
end


# Visualisierung #

## Funktionen zur Visualisierung ##

### Partikelanzahlplot ###
function partikelanzahlplot(xq,zq)
    #Initialisierung der Parameter
    lambda=5
    sigma=1
    # Partikelanzahlen
    na= 100
    nb= 1000
    nc = 5000
    nd= 10000

    # Grafeneinstellungen
    levels=  [0.00001,0.01,0.025,0.05,0.5,0.75,1.0,1.25,1.5,2,5]
    fig = Figure(resolution=(1080,1080))
    xs=LinRange(0, xgrenz,xgrenz)
    ys=LinRange(0, zgrenz, zgrenz)

    # Erster Plot
    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)

    #Zweiter Plot
    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 )

    #Dritter Plot
    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 )

    #Vierter Plot
    d=monte(xq,zq,nd,lambda,sigma)
    contourf(fig[2, 2],xs, ys,d,levels=levels,title= "N = " *string(nd))
```

```
307     contour!(fig[2, 2],ys, xs,d ,levels=levels )
308
309     #Farbskala
310     Colorbar(fig[1:2,3], limits = (0.1, 5), colormap = :viridis, flipaxis =
        false, label = "kg/s")
311
312     Label(fig[3, :], text = "x(m)")
313     Label(fig[:, 0], text = "z(m)")
314     Label(fig[0, :], text = "Konzentrationen im Vergleich fuer x = " *string
        (xq)*"m z = " *string(zq)*"m", textsize = 30)
315     save("Bericht/Bilder/3_vergleich_x = "*string(xq)*".png", fig)
316
317     #L schen der Grafik aus dem Zwischenspeicher
318     empty!(fig)
319 end
320
321 ### Einzelplot ###
322 function einzelplot(xq,zq)
323     #Initialisierung der Parameter
324     n=5000
325     lambda=5
326     sigma=1
327
328     # Grafeneinstellungen
329     levels= [0.00001,0.01,0.025,0.05,0.5,0.75,1.0,1.25,1.5,2,5]
330     fig = Figure(resolution=(1080,1080))
331     xs=LinRange(0, xgrenz,xgrenz)
332     ys=LinRange(0, zgrenz, zgrenz)
333
334     #Plot
335     a=monte(xq,zq,n,lambda,1)
336     contourf(fig[1, 1],ys, xs,a ,levels=levels, xlabel = "x label", ylabel =
        "y label" )
337     contour!(fig[1, 1],ys, xs,a ,levels=levels )
338
339     #Label
340     Colorbar(fig[1,2], limits = (0, 5), colormap = :viridis,flipaxis = false
        ,  label = "kg/s")
341     Label(fig[0, :], text = "Konzentration bei x = " *string(xq)*"m z = " *
        string(zq)*"m   ="*string(lambda), textsize = 30)
342     Label(fig[2, :], text = "x(m)")
343     Label(fig[:, 0], text = "z(m)")
344     save("Bericht/Bilder/3_single_x = "*string(xq)*".png", fig)
345     #L schen der Grafik aus dem Zwischenspeicher
346     empty!(fig)
347 end
348
349 ### Funktion zur Erstellung des Lambda-Vergleichsplots
350 function lambdaplot(xq,zq)
351     #Initialisierung der Parameter
352     n= 1000
353     sigma=5
354     #Initialisieren der Lambda Werte
355     la= 10
356     lb= 5
357     lc =2
358     ld= 1
359
```

```
    #Grafeneinstellungen
    levels= [0.00001,0.01,0.025,0.05,0.5,0.75,1.0,1.25,1.5,2]
    fig = Figure(resolution=(1080,1080))
    xs=LinRange(0, xgrenz,xgrenz)
    ys=LinRange(0, zgrenz, zgrenz)

    # Erster Plot
    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 )

    #Zweiter Plot
    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 )

    # Dritter Plot
    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 )

    #Vierter Plot
    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 )

    #Label
    Colorbar(fig[1:2,3], limits = (0.00001, 5), colormap = :viridis,flipaxis
    = false, label = "kg/s")
    Label(fig[3, :], text = "x(m)")
    Label(fig[:, 0], text = "z(m)")
    Label(fig[0, :], text = "Konzentrationen im Vergleich x = " *string(xq)*
   "m z = " *string(zq)*" m bei verschiedenen   ", textsize = 30)
    save( "Bericht/Bilder/3_lambda_x = "*string(xq)*".png", fig)

    #L schen der Grafik aus dem Zwischenspeicher
    empty!(fig)
end

### Funktion zur Erstellung des Sigma-Vergleichsplots
function sigmaplot(xq,zq)
    #Initialisierung der Parameter
    l=5
    n= 1000
    siga=1
    sigb=2
    sigc=5
    sigd=10

    #Grafeneinstellungen
    levels=[0.001,0.01,0.025,0.05,0.5,0.75,1.0,1.25,1.5,2,5]
    fig = Figure(resolution=(1080,1080))
    xs=LinRange(0, xgrenz,xgrenz)
    ys=LinRange(0, zgrenz, zgrenz)

    #Erster Plot
    a=monte(xq,zq,n,l,siga)
    contourf(fig[1, 1],ys, xs, a,levels=levels)
```

```
416     contour!(fig[1, 1],ys, xs,a ,levels=levels )
417
418     #Zweiter Plot
419     b=monte(xq,zq,n,l,sigb)
420     contourf(fig[1, 2],xs, ys, b,levels=levels)
421     contour!(fig[1, 2],ys, xs,b ,levels=levels )
422
423     #Dritter Plot
424     c=monte(xq,zq,n,l,sigc)
425     contourf(fig[2, 1],xs, ys,c,levels=levels)
426     contour!(fig[2, 1],ys, xs,c ,levels=levels )
427
428     #Vierter Plot
429     d=monte(xq,zq,n,l,sigd)
430     contourf(fig[2, 2],xs, ys,d,levels=levels)
431     contour!(fig[2, 2],ys, xs,d ,levels=levels )
432
433     #Label
434     Colorbar(fig[1:2,3], limits = (0.00001, 5), colormap = :viridis,flipaxis
        = false, label = "kg/s")
435     Label(fig[3, :], text = "x(m)")
436     Label(fig[:, 0], text = "z(m)")
437     Label(fig[0, :], text = "Konzentrationen im Vergleich x = " *string(xq)*
       "m z = " *string(zq)*" m bei verschiedenen  u  &  w ", textsize = 30)
438     save("Bericht/Bilder/3_sigma_x = "*string(xq)*".png", fig)
439
440     #L schen der Grafik aus dem Zwischenspeicher
441     empty!(fig)
442 end
443
444
445 # Main Funktion
446 function main()
447
448     ### Aufgbabe a
449     xq = 60.5   # m
450     zq = 0.5    # m
451     einzelplot(xq,zq)
452     partikelanzahlplot(xq,zq)
453     lambdaplot(xq,zq)
454     sigmaplot(xq,zq)
455
456     ### Aufgbabe b
457     xq = 15.5  # m
458     zq =  65.5 # m
459     einzelplot(xq,zq)
460     partikelanzahlplot(xq,zq)
461     lambdaplot(xq,zq)
462     sigmaplot(xq,zq)
463
464 end
465
466 #Ausf hren der Mainfunktion
467 main()
```