

Programmierpraktikum

Der Bericht

Alexander Steding

10028034

Gottfried Wilhelm Leibniz Universität

22. September 2022

Inhaltsverzeichnis

1	Einleitung	4
2	Aufgabe 1	4
2.1	Aufgabenteil a	4
2.2	Aufgabenteil b	6
3	Aufgabe 2	8
4	Aufgabe 3	9
4.1	Einfluss der Teilchenanzahl	11
4.2	Einfluss von λ	13
4.3	Einfluss von σ	14
5	Aufgabe 4	15
6	Quellcode	16
6.1	Aufgabe 2	16
6.2	Aufgabe 3	20

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 GIMakie.

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 kontinuierliche 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 kontinuierlichen 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.

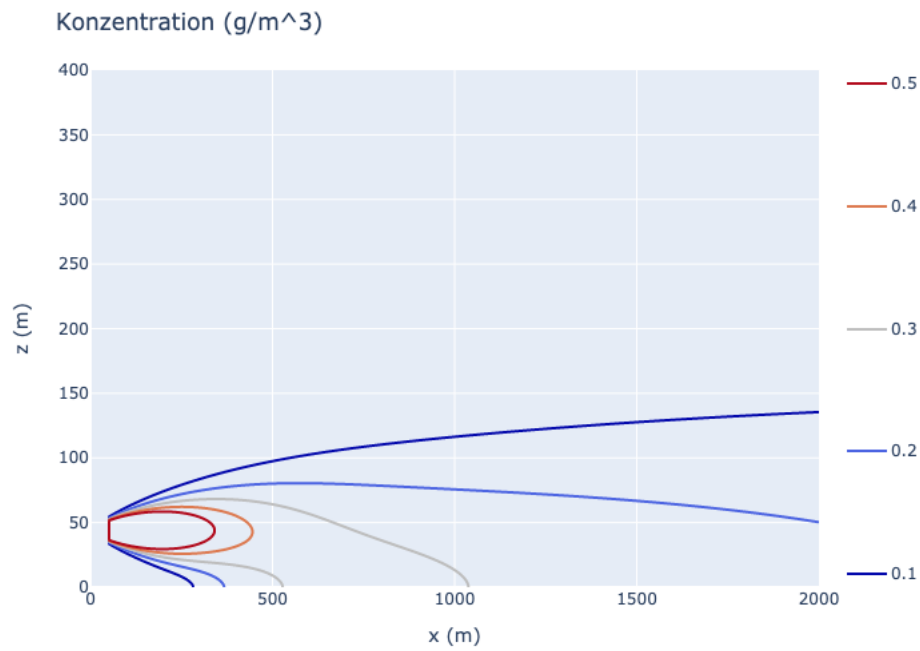


Abb. 1: Konzentrationsverteilung als Contour-Plot. Die Konzentration ist in $\frac{\text{g}}{\text{m}^3}$ dargestellt.

Für eine feinere grafische Analyse wurde in Abbildung 2. die Konzentrationsverteilung am Erdboden, also für $z = 0$ m, visualisiert.

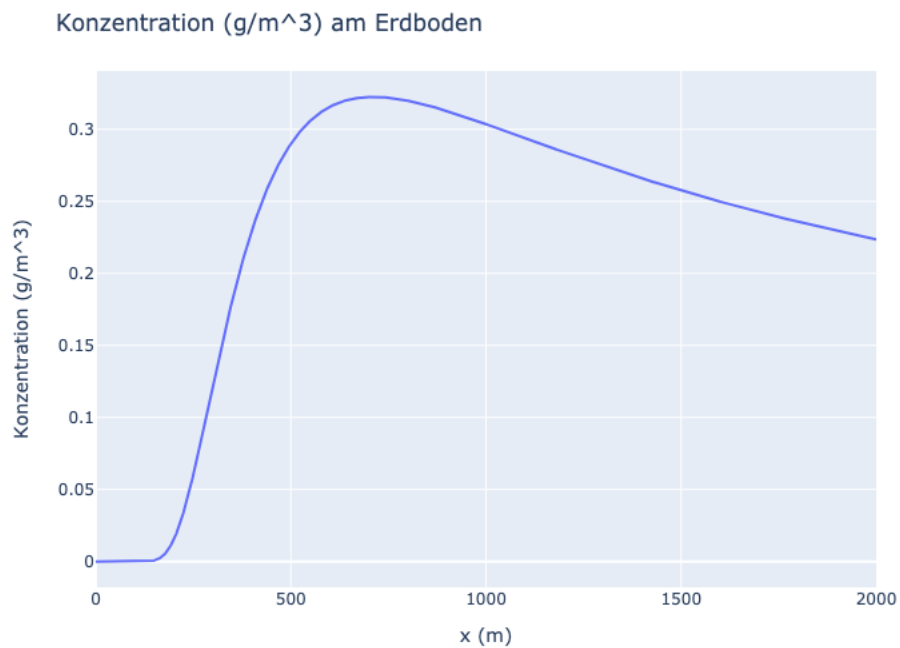


Abb. 2: Konzentrationsverteilung am Erdboden. Die Konzentration ist in $\frac{\text{g}}{\text{m}^3}$ dargestellt.

Die maximale Konzentration wurde final rechnerisch mittels Julia bestimmt als

$$c[0, 711] = 0,32263 \frac{\text{g}}{\text{m}^3} \quad (7)$$

2.2 Aufgabenteil b

In Aufgabenteil b wird nun das Monte-Carlo-Modell mit einer genährten Gitterauswertung. 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 $N = 1000$

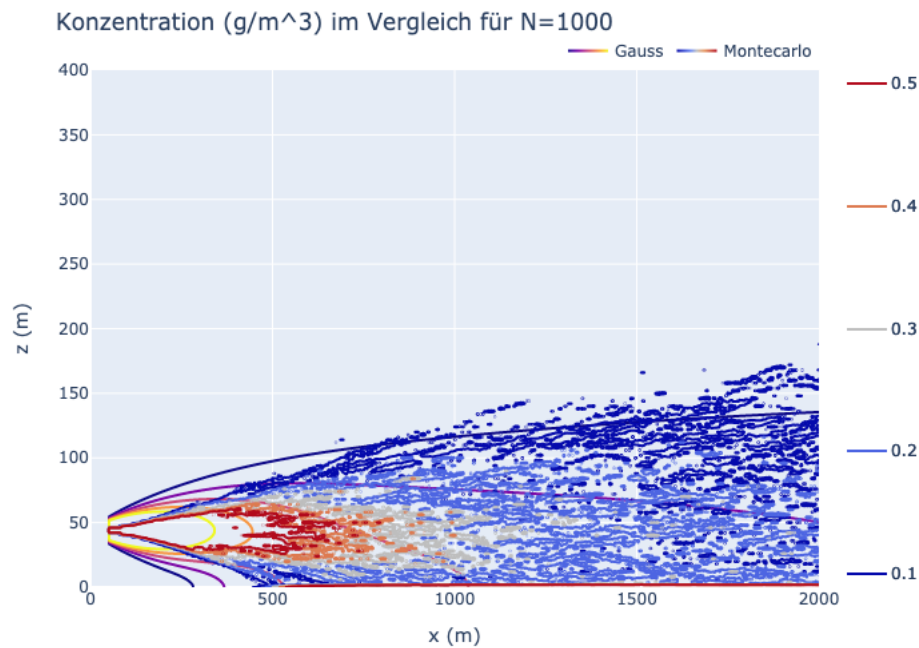


Abb. 3: Vergleich für $N=1000$. Die Konzentration ist in $\frac{\text{g}}{\text{m}^3}$ dargestellt.

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

$$N = 1000 \quad (8)$$

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

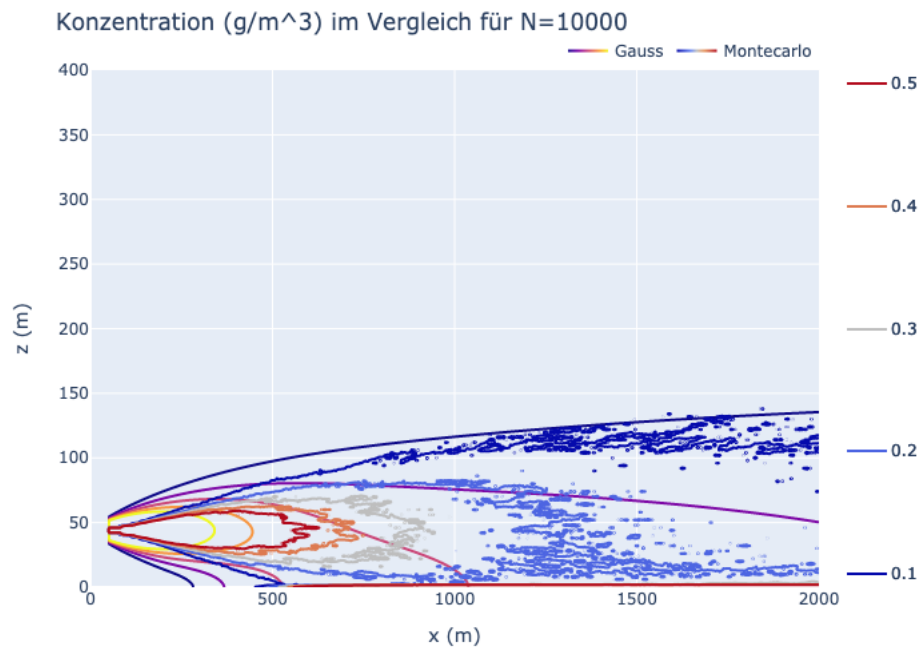


Abb. 4: Vergleich für $N=10000$. Die Konzentration ist in $\frac{\text{g}}{\text{m}^3}$ dargestellt.

Bei einer mittleren Anzahl wie

$$N = 10000 \quad (9)$$

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

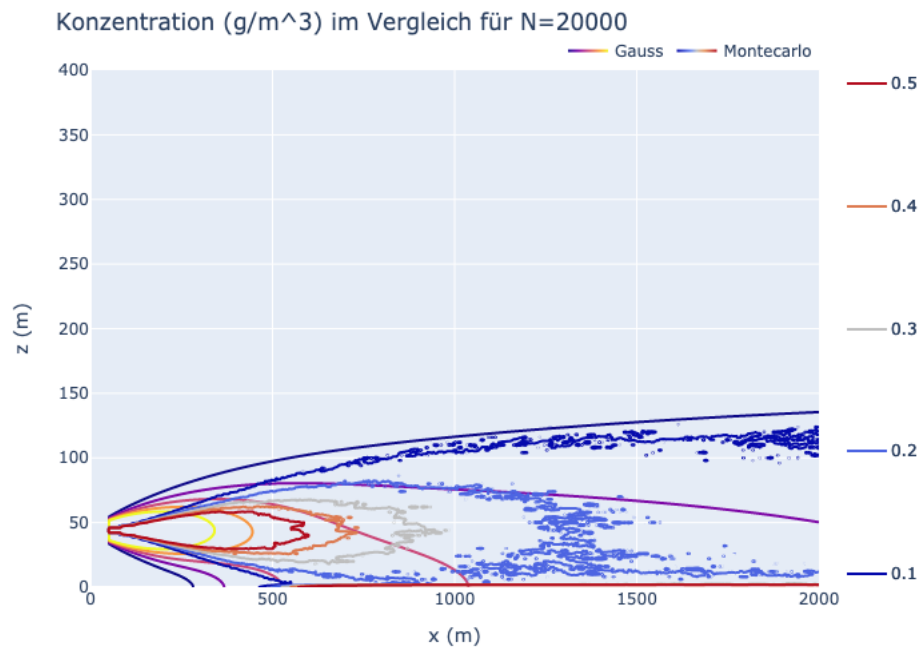


Abb. 5: Vergleich für $N=20000$. Die Konzentration ist in $\frac{\text{g}}{\text{m}^3}$ dargestellt.

Bei einer hohen Anzahl wie

$$N = 20000 \quad (10)$$

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

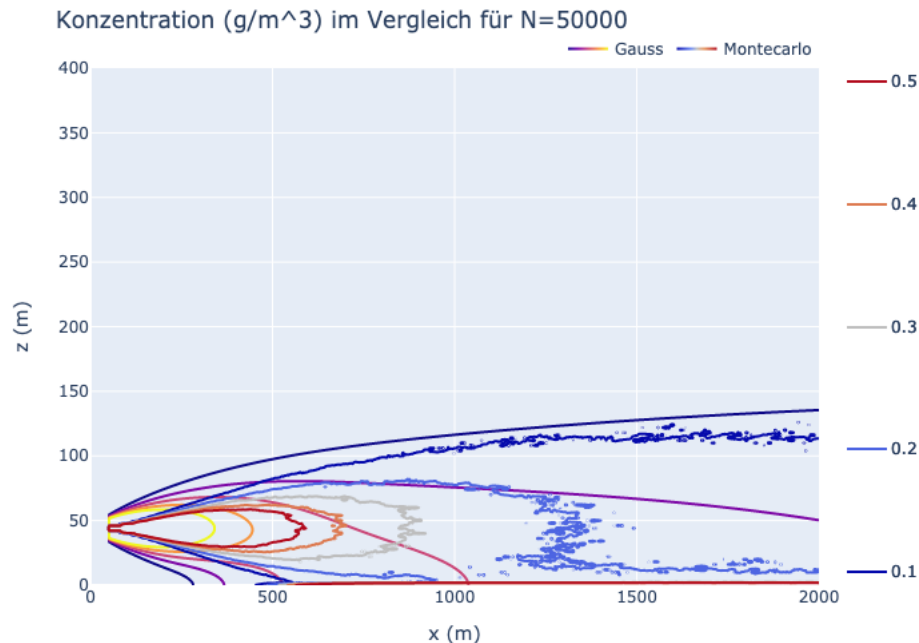


Abb. 6: Vergleich für $N=50000$. Die Konzentration ist in $\frac{\text{g}}{\text{m}^3}$ dargestellt.

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

$$N = 50000 \quad (11)$$

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

$$N = 20000 \quad (12)$$

Die größten Unterschiede zwischen dem Monte-Carlo-Modell und dem Gauß-Modell entstehen 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 niedriger als beim Gaußmodell, allen voran aber fehlt hier die Variation der Konzentration, die mit dem Gaußmodell 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 Validierung 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:

$$x = 0\text{m} \quad (13)$$

$$z = 0.5\text{m} \quad (14)$$

$$z_0 = 0.008\text{m} \quad (15)$$

$$u_* = 0.35 \frac{\text{m}}{\text{s}} \quad (16)$$

$$\kappa = 0.38 \quad (17)$$

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

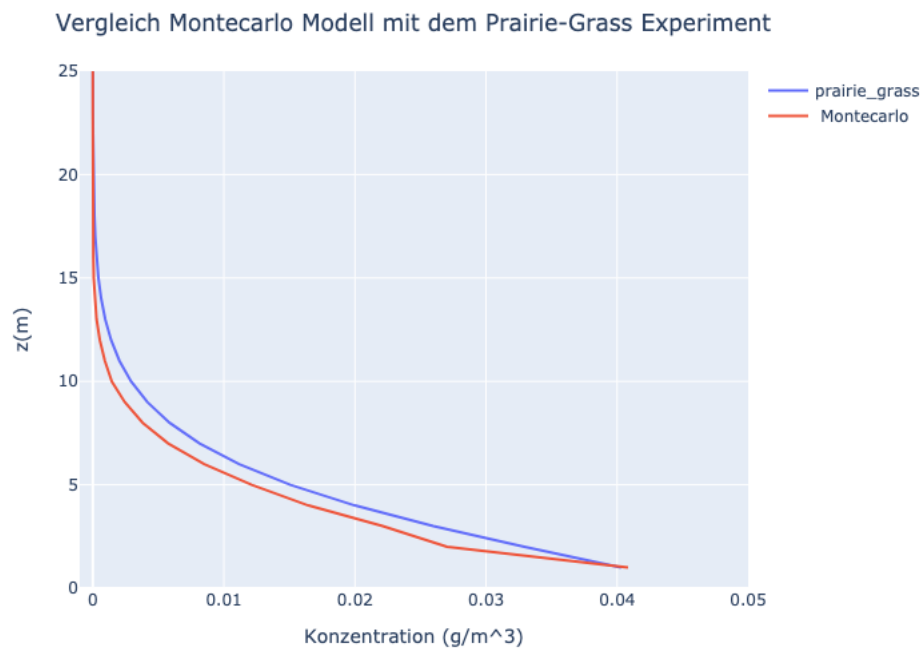


Abb. 7: Vergleich Prairie-Grass, Monte-Carlo. Die Konzentration ist in $\frac{\text{g}}{\text{m}^3}$ dargestellt.

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 gesamten 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 gegeneinander 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 Standardabweichung 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 GIMakie als Visualisierungs-Backend verwendet. Im ersten Szenario wurde ein Schadstoffquelle im Straßenverkehr mit Quellort

$$x = 60.5\text{m} \quad (18)$$

$$z = 0.5\text{m} \quad (19)$$

simuliert.

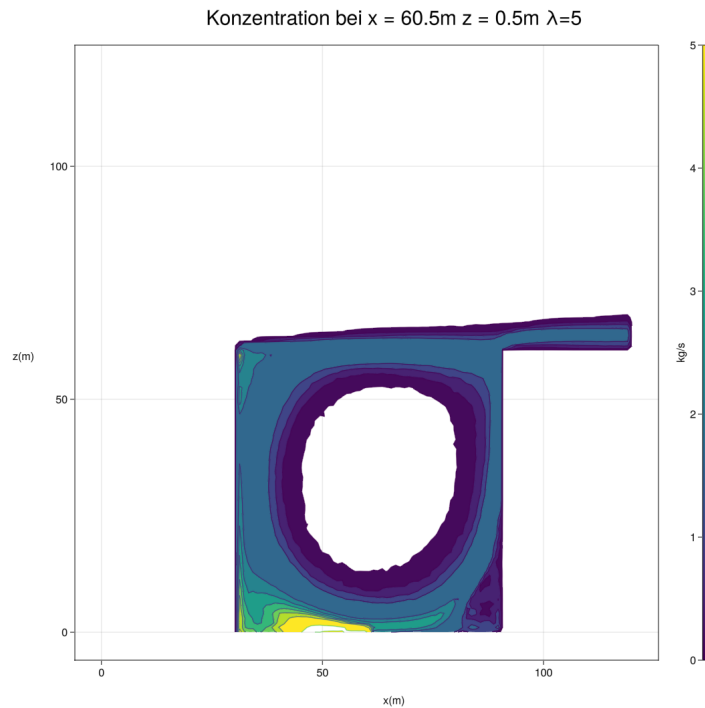


Abb. 8: Konzentrationsverteilung am Erdboden. Die Konzentration ist in $\frac{\text{kg}}{\text{s}}$ dargestellt. $N=1000$, $\lambda = 5$

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 tendenziell 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 Schadstoff zu negativen gesundheitlichen Folgen für Fußgänger*innen, Radfahrer*innen, Anwohner*innen, welche sich in der Nähe aufhalten

Im zweiten Szenario wurde als Schadstoffquelle ein Hausbrand mit Quellort

$$x = 15.5\text{m} \quad (20)$$

$$z = 65.5\text{m} \quad (21)$$

simuliert.

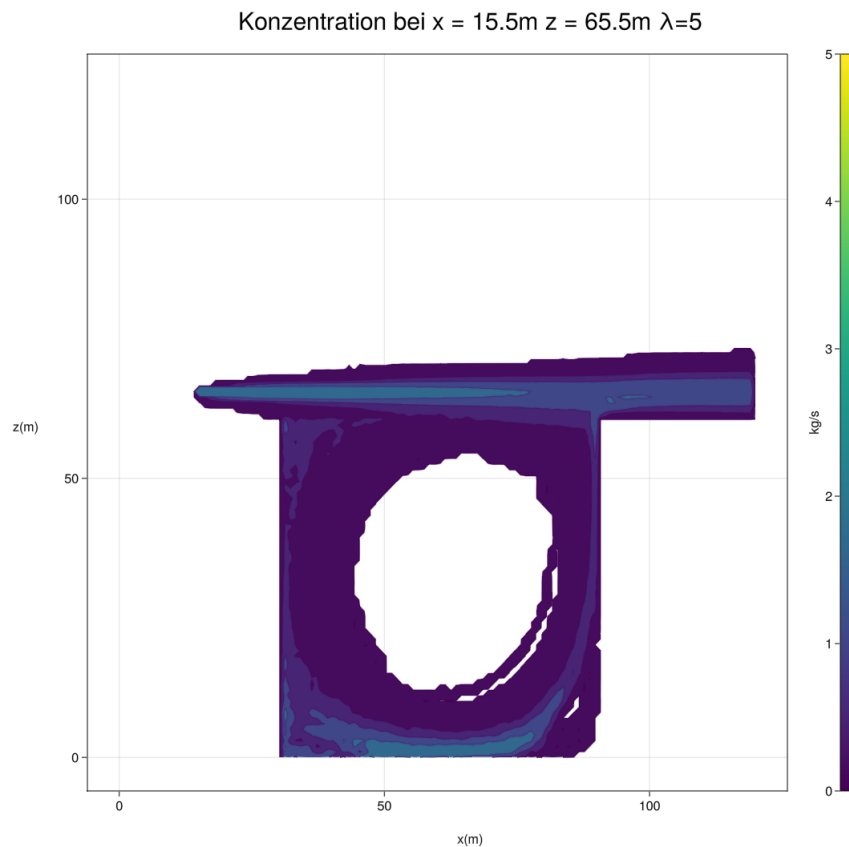


Abb. 9: Konzentrationsverteilung am Erdboden. Die Konzentration ist in $\frac{\text{kg}}{\text{s}}$ dargestellt. $N=1000$, $\lambda = 5$

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, wodurch 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.

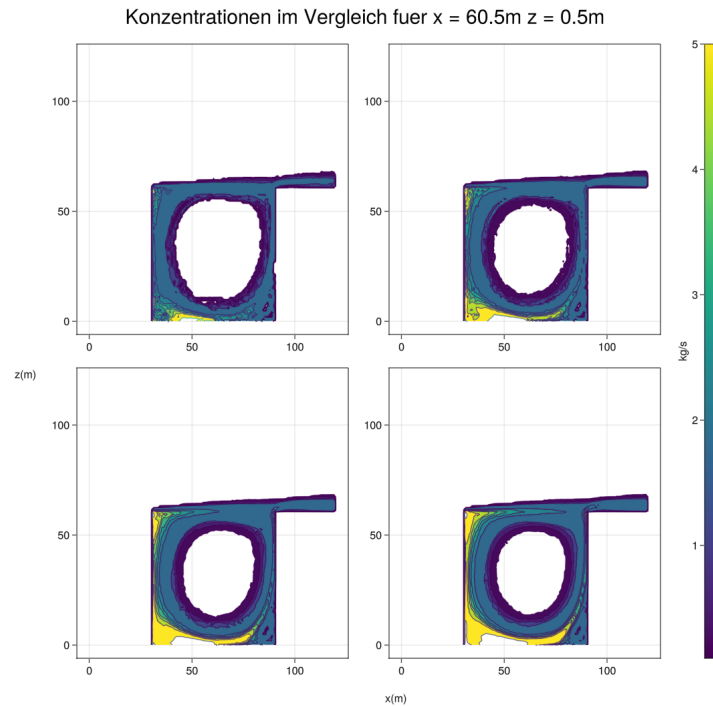


Abb. 10: Verschiedene Konzentrationsverteilung im Vergleich. $N \in [10^2, 10^3, 5 * 10^3, 10^4]$ v.L.n.R.
 $\lambda = 5$

Für die Situation a) bringt eine Erhöhung der Teilchenzahl von $N = 100$ dargestellt in Abbildung 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 $N = 5000$ ermöglicht noch eine gering höhere Auflösung; $N = 10000$ führen trotz höherer Rechenzeit nur zu einer marginalen Verbesserung. Aus ökonomischen Gesichtspunkten ist eine Teilchenanzahl zwischen 1000 und 5000 optimal.

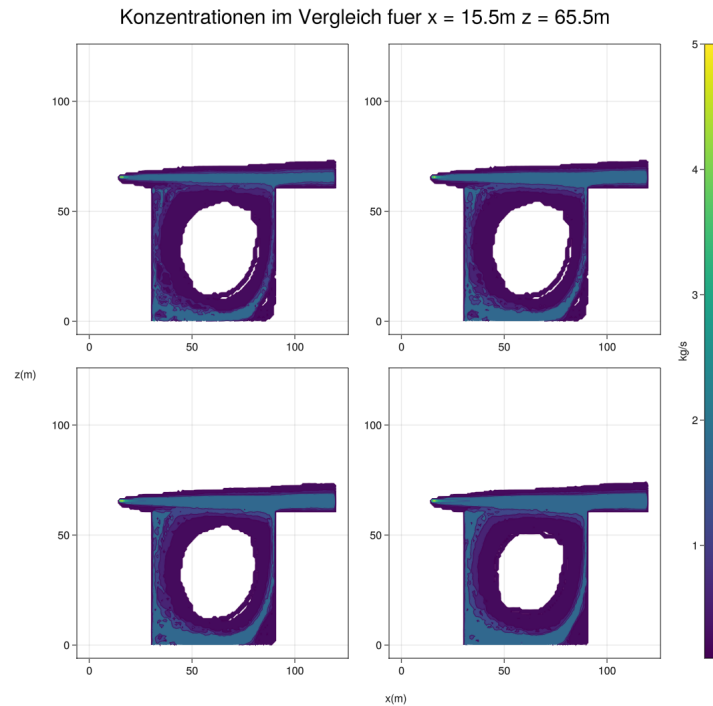


Abb. 11: Verschiedene Konzentrationsverteilung im Vergleich. $N \in [10^2, 10^3, 5 * 10^3, 10^4]$ v.L.n.R., $\lambda = 5$

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 Situation a) verwendet werden.

4.2 Einfluss von λ

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

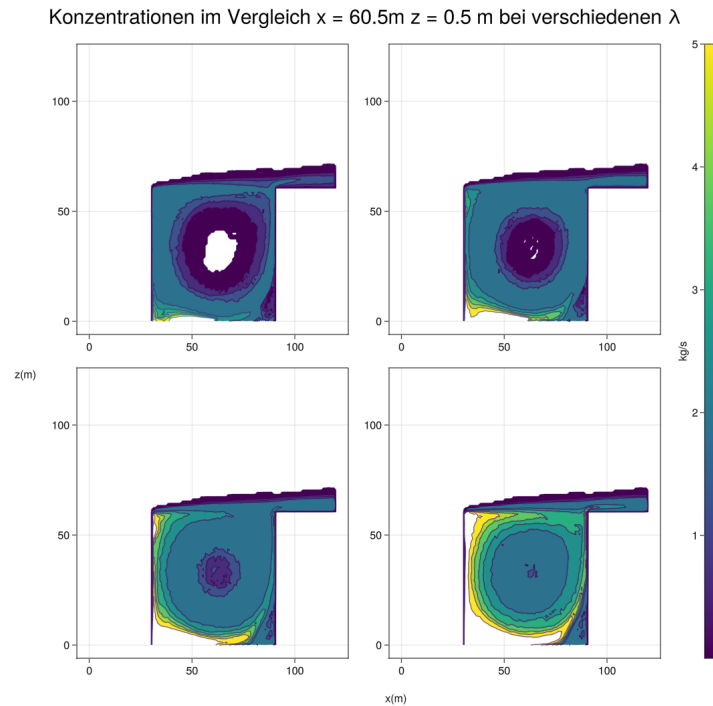


Abb. 12: Verschiedene λ im Vergleich. $\lambda \in [10, 5, 2, 1]$ v.L.n.R., $N=1000$

Da es sich bei λ um eine Komponente des Mischungsweg handelt, führt eine Vergrößerung dieses zu einer Vergrößerung 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 σ_w analog betrachtet werden. Es wurden jeweils beide Standartabweichungen simultan verändert.

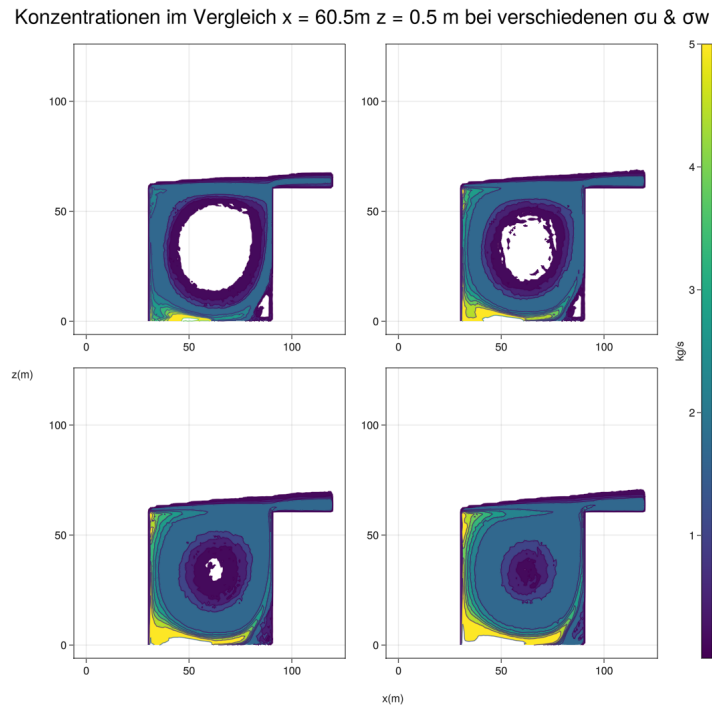


Abb. 13: Verschiedene σ im Vergleich. $\sigma \in [1, 2, 5, 10]$ v.L.n.R $N=1000$, $\lambda = 5$

Eine Veränderung der Standardabweichung hat in diesem Simulations-Modell direkten Einfluss auf den turbulenten Anteil der Geschwindigkeitskomponente. Eine Zunahme der Standardabweichung führt folglich zu einem stärker ausgeprägtem Wirbel, eine Reduzierung zu einem schwächerem. Analog dazu verhält sich aufgrund der variierenden 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 dreidimensionalen Stadtgebiet müssten neben der zusätzlichen räumlichen Ausbreitungscoordinate, auch die neue Gitterdimension, sowie die Windkomponenten angepasst und erweitert werden. Auch muss überlegt werden ob statt dem Mischungsweg der turbolente Diffusionskoeffizient als multidimensionale Größe verwendet werden kann. Dieser könnte möglicherweise aus dem Strömungsmodell stammen.

6 Quellcode

6.1 Aufgabe 2

```
1 using NetCDF
2 using Random, Distributions
3 using ProgressBars
4 using PlotlyJS
5
6 # Definition der globalen Variablen
7 global n, ubalken, wbalken, zq, xq, xgrenz, zgrenz, tl, nx, ny, nz, dx, dy, dz::Int
8 global dt, sigu, sigw, uster, k, znull, q::Float64
9 global units::String
10 global gitter, cd, x, cdground::Array
11
12 n= 10^3           # Anzahl Partikel
13 zq = 0           # Quellort z Komponente in m
14 xq = 0.5         # Quellort x Komponente in m
15 xgrenz= 110      # Grenze in x Richtung in m
16 zgrenz=25        # Grenze in z Richtung in m
17 dx = 1           # Gitterweite x Richtung in m
18 dz = 1           # Gitterweite z Richtung in m
19 uster = 0.35     # Schubspannungsgeschwindigkeit in m/s
20 k = 0.38         # Kappa
21 znull = 0.008    # Rauigkeitslaenge
22 sigu = 2.5 * uster # Standartabweichung u m/sm/s
23 sigw = 1.3 * uster # Standartabweichung w m/s m/s
24 q= 0.7           # Konzentration fuer das Montecarlo Modell in m/s
25 rl= exp(- dt/tl) # Berechnung von rl
26 units = "g/m^3"  # Einheit fuer Graphen
27
28 ## Arrays Initialisieren
29
30
31 #Array wird in der Gr   e xgrenz*zgrenz aufgespannt und mit 0 gef llt
32 konk=zeros(xgrenz,zgrenz)
33
34
35
36 # Montecarlo-Modell #
37
38 ##Funktionen zur Berechnung des Montecarlo Modell ##
39
40 ### Berechnung der Prandtlschicht###
41 function prandltl(zi,xi)
42     #### Berechnung der mittleren Windgeschwindigkeit ####
43     if zi < znull
44         ubalken = 0 #kein mittlerer Wind f r zi < znull
45     else
46         ubalken = (uster / k) * log(abs(zi) / znull) # Log. Windprofil
47     end
48     ##Berechnung des Lagrangeschen Zeitschrittes tl
49     tl = ((k * uster) / sigw ^ 2) * abs(zi)
50
51     #### Berechnung des Zeitschrittes dt
52     if (0.1*tl)>((k * uster) / sigw ^ 2) * abs(2)
53         dt = 0.1*tl # Normalfall
```

```

54     else
55         dt = ((k * ustern) / sigw ^ 2) * abs(2) #falls dt kleiner als tl in
12 m Hoehe wird dt auf tl(2m) gesetzt
56     end
57     return tl, dt, ubalken
58 end
59
60
61 ### Berechnung der Positionen###
62 function positionen(xi, wi, zi,tl, ui, dt)
63     #Lagrangesche Autokorellationsfunktion
64     rl = exp(- dt / tl)
65     Random.seed!()
66     d = Normal()
67     rr = rand(d, 1)[1]
68
69     xi = xi + ui * dt
70     wi=rl*wi + sqrt((1 - rl^2))*sigw* rr
71     zi = zi + wi * dt
72     return xi, wi, zi
73
74 end
75 ### Gitterauswertung ###
76 function exaktgitter(xi, xold, zi, zold,dt)
77     #Initialisierung
78     ti = []
79     toks = []
80     # Berechnung der Anzahl an Gitterschnittpunkten
81     rangex = convert{Int64,floor(abs(xi - xold))}
82     rangez = convert{Int64,floor(abs(zi - zold))}
83
84     # Fallunterscheidung: Gibt es Schnittpunkte
85     if rangex >0 && rangez>0
86         #Falls es sie gibt muss die Konzentration pro Anteil Giterpunkt
berechnet werden
87         # Berechnung von ti und tj
88         for i in 0:rangex
89             if xold >xi
90                 xsi=ceil(xold) +i
91             else
92                 xsi=ceil(xold) -i
93             end
94             push!(toks,(xsi - xold) / abs(xi - xold)) # Einh ngen in die
Liste
95         end
96         for i in 0:rangez
97             if zold >zi
98                 zsi=ceil(zold) +i
99             else
100                 zsi=ceil(zold) -i
101             end
102             push!(toks,(zsi - zold) / abs(zi - zold)) # Einh ngen in die
Liste
103         end
104         tku = sort!(toks) #Sortieren der Liste
105         for i in 2:length(tku)
106             ti = tku[i]
107             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 können die Betroffenen Gitterpunkte ermittelt werden
111         posx, posz = gg(xold, zold, xi, zi, t) #Ausführen der
Geradengleichung
112         #Anschließend wird die Konzentrationen am mit der GG
berechneten Punkt um den entsprechenden Betrag erhöht
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 Gitterpunkt verbleibt
117     konk[convert(Int64,ceil(xi)),convert(Int64,ceil(zi))] += ((q * dt)/(n
* dx * dz)) #Erhöhung um die gesamte 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 über alle Partikel
142         #Initialisieren der Parameter für 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 für x
150             zold=zi #Zwischenspeichern der alten Werte für y
151             if zi<znull #Berücksichtigen 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

```

```

158         tl, dt, ui = prandltl(zi, xi) #Berechnung der Prandtlschicht
159         xi, wi, zi = positionen(xi, wi, zi, tl, ui, dt) #Berechnung der
        neuen Positionen
160         exaktgitter(xi, xold, zi, zold, dt) #Exakte Gitterauswertung
161     end
162 end
163
164 end
165 return konk #Zur ckgeben der Konzentration als Feld
166 end
167
168 # Prairie-Grass-Experiments #
169 function prairie_grass(konk)
170     pg_mod= [] #Initialisieren
171     #Ermitteln der ben tigten Werte aus dem Montecarlo-Modells
172     for i in 100:xgrenz
173         for j in 1:zgrenz
174             push!(pg_mod, konk[i,j])
175         end
176     end
177     c0 = 4.63E-02
178     gamma= 0.68
179     my = 1.3
180     zs = 3.4
181     z= collect(1:zgrenz)
182     pg = zeros(length(z)+1)
183     for k in 1:zgrenz
184         pg[k] = c0 * exp(-gamma * (z[k]/zs)^my)
185     end
186     return pg, pg_mod
187 end
188
189 ### Visualisierung ###
190
191 function grafen(pg, pg_mod)
192     #Darstellung mit PlotlyJS
193     title="Vergleich Montecarlo Modell mit dem Prairie-Grass Experiment"
194     xaxis_title="Konzentration (" * units * ")"
195     plot_prairie=scatter(y=collect(1:zgrenz), x=pg, name="prairie_grass",
        showlegend=true ,)
196     plot_monte= scatter(y=collect(1:zgrenz), x=pg_mod, name=" Montecarlo",
        showlegend=true ,)
197     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")
198 end
199
200
201
202
203 ### Main #####
204 function main()
205     #Zun chst wird das Montecarlo-Modell berechnet
206     konzentrationen=monte() #Die Konzentration wird in einem Array
        zwischengespeichert
207     #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 benutzt 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,lambdak,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/lambdak)))/(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/lambdak)))/(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
38         # Normalfall
39     else
40         dt = 0.05*((k*2)/(1+k*(2/lambdak)))/(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
41     end
42     return tl, dt

```

```

42 end
43
44 ### Gitterauswertung ###
45 function exaktgitter(xi, xold, zi, zold, dt, n)
46     #Initialisierung
47     ti = []
48     toks = []
49     # Berechnung der Anzahl an Gitterschnittpunkten
50     rangex = convert(Int64, floor(abs(xi - xold)))
51     rangez = convert(Int64, floor(abs(zi - zold)))
52
53     # Fallunterscheidung: Gibt es Schnittpunkte
54     if rangex > 0 && rangez > 0
55         # Falls es sie gibt muss die Konzentration pro Anteil Giterpunkt
56         berechnet werden
57         # Berechnung von ti und tj
58         for i in 0:rangex
59             if xold > xi
60                 xsi=ceil(xold) + i
61             else
62                 xsi=ceil(xold) - i
63             end
64             push!(toks, (xsi - xold) / abs(xi - xold)) # Einhängen in die
65             Liste
66         end
67         for i in 0:rangez
68             if zold > zi
69                 zsi=ceil(zold) + i
70             else
71                 zsi=ceil(zold) - i
72             end
73             push!(toks, (zsi - zold) / abs(zi - zold)) # Einhängen in die
74             Liste
75         end
76         tku = sort!(toks) #Sortieren der Liste
77         for i in 2:length(tku)
78             ti = tku[i]
79             told = tku[i - 1]
80             t = mean([told, ti])
81             # Der Mittelwert t der beiden Punkte told und ti kann in einer
82             Geradengleichung verwendet werden
83             # So können die Betroffenen Gitterpunkte ermittelt werden
84             posx, posz = gg(xold, zold, xi, zi, t) #Ausführen der
85             Geradengleichung
86             #Anschließend wird die Konzentrationen am mit der GG
87             berechneten Punkt um den entsprechenden Betrag erhöht
88             konk[abs(posx), abs(posz)] += ((tku[i] - tku[i-1]) * dt * ((q * dt)
89             / (n * dx * dz)))
90         end
91     else
92         #Gitterauswertung falls Partikel in dem Giterpunkt verbleibt
93         konk[convert(Int64, ceil(xi)), convert(Int64, ceil(zi))] += ((q * dt) / (n
94         * dx * dz)) #Erhöhung um die gesamte Konzentration
95     end
96 end
97
98 ### Berechnung der Geradengleichung ###
99 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 können Gitterpunkte jenseits der Grenzen
    berechnet werden
96     #Um einen Abbruch zu verhindern werden in diesem 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 müssen 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 möglichen Grenzfällen für 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 Differentialquotient
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

```

```

140     wturbo = rl * wi + sqrt((1 - rl ^ 2)) * sqrt(ws[abs(convert(Int64,(ixi))
141     ),abs(convert(Int64,(izi)))])*rr +(1-rl)*tl*difqw
142
143     #Berechnung der Windgeschwindigkeit (Mittelwert + Turbulenz)
144     ui = u[abs(convert(Int64,(ixold)),abs(convert(Int64,(izold)))] +uturbo
145     wi = w[abs(convert(Int64,(ixold)),abs(convert(Int64,(izold)))] +wturbo
146
147     #Berechnung der neuen Positionen
148     xi = xi + ui * dt
149     zi = zi + wi * dt
150
151     #Reflexion    berprfen
152     while ((xi<=31 && zi<=61)|| (xi>=90 && zi <=61)|| (30<=xi<=90 && zi<=0))
153         if (xi>=90 && zi<=61) # Reflexion an der rechten Wand
154             eckenreflexion(xi,zi,xold,zold,ui,wi)
155             if br==1 # Falls dies der Fall ist gab es keine Eckenreflexion
156                 xi= xi-2*(abs(90-xi))
157                 ui=-ui
158             end
159         elseif (xi<=31 && zi<=61) # Reflexion an der linken Wand
160             eckenreflexion(xi,zi,xold,zold,ui,wi)
161             if br==1 # Falls dies der Fall ist gab es keine Eckenreflexion
162                 xi=xi+2*(abs(31-xi))
163                 ui=-ui
164             end
165         elseif (30<=xi<=90 && zi<=0) #Reflexion am Boden
166             eckenreflexion(xi,zi,xold,zold,ui,wi)
167             if br==1 # Falls dies der Fall ist gab es keine Eckenreflexion
168                 wi=-wi
169                 zi= -zi
170             end
171         else
172             print("Fehler in der Reflexion")
173         end
174     end
175 end
176 return xi, wi, zi,ui
177
178 end
179
180
181 ### Funktion zur    berprfung    der Eckenreflexion ###
182 function  eckenreflexion(xi,zi,xold,zold,ui,wi)
183     #Hier wird die Reflexion an den Ecken    berprft    .
184     #Falls es eine Eckenreflexion gibt muss eine der folgenden Gleichungen
185     l sbar sein, wodurch sich deren Typ ver ndert
186     #Reflexion an der linken oberen Ecke
187     if xi<=30&& typeof(solve(r*[1,1]+[30,60]-[xold,zold],r)) !=Vector{Any}
188         && typeof(solve(r*[1,1]+[30,60]-[xi,zi],r)) !=Vector{Any}
189         ui=-ui
190         wi=-wi
191         xi=xold
192         zi=zold
193         br=2
194     #Reflexion an der linken unteren Ecke
195     elseif xi<=30&& typeof(solve(r*[1,1]+[30,0]-[xold,zold],r)) !=Vector{Any}
196         && 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)

```

```

250         #Abspeichern der alten Werte
251         xolder=xold
252         zolder=zold
253         xold = xi
254         zold = zi
255         #Berechnung der Prandtlschicht
256         tl, dt = prandltl(zi,xi,lambda,us,ws,k)
257         #Berechnung der neuen Positionen
258         xi, wi, zi,ui = positionen(xi, wi, zi, tl, ui, dt,xold,zold,
xolder,zolder,us,ws,u,w)
259         # Gitterauswertung der Konzentration
260         exaktgitter(xi, xold, zi, zold,dt,n)
261     end
262
263 end
264 return konk #Zur ckgeben der Konzentration als Feld
265 end
266
267 # Visualisierung #
268
269 ## Funktionen zur Visualisierung ##
270
271 ### Partikelanzahlplot ###
272 function partikelanzahlplot(xq,zq)
273     #Initialisierung der Parameter
274     lambda=5
275     sigma=1
276     # Partikelanzahlen
277     na= 100
278     nb= 1000
279     nc = 5000
280     nd= 10000
281
282     # Grafeneinstellungen
283     levels= [0.00001,0.01,0.025,0.05,0.5,0.75,1.0,1.25,1.5,2,5]
284     fig = Figure(resolution=(1080,1080))
285     xs=LinRange(0, xgrenz,xgrenz)
286     ys=LinRange(0, zgrenz, zgrenz)
287
288     # Erster Plot
289     a=monte(xq,zq,na,lambda,sigma)
290     contourf(fig[1, 1],ys, xs, a,levels=levels)
291     contour!(fig[1, 1],ys, xs,a ,levels=levels)
292
293     #Zweiter Plot
294     b=monte(xq,zq,nb,lambda,sigma)
295     contourf(fig[1, 2],xs, ys, b,levels=levels,title= "N = " *string(nb))
296     contour!(fig[1, 2],ys, xs,b ,levels=levels )
297
298     #Dritter Plot
299     c=monte(xq,zq,nc,lambda,sigma)
300     contourf(fig[2, 1],xs, ys,c,levels=levels,title= "N = " *string(nc))
301     contour!(fig[2, 1],ys, xs,c ,levels=levels )
302
303     #Vierter Plot
304     d=monte(xq,zq,nd,lambda,sigma)
305     contourf(fig[2, 2],xs, ys,d,levels=levels,title= "N = " *string(nd))
306

```



```

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

```

```

360 #Grafeneinstellungen
361 levels= [0.00001,0.01,0.025,0.05,0.5,0.75,1.0,1.25,1.5,2]
362 fig = Figure(resolution=(1080,1080))
363 xs=LinRange(0, xgrenz,xgrenz)
364 ys=LinRange(0, zgrenz, zgrenz)
365
366 # Erster Plot
367 a=monte(xq,zq,n,la,sigma)
368 contourf(fig[1, 1],ys, xs, a,levels=levels)
369 contour!(fig[1, 1],ys, xs,a ,levels=levels )
370
371 #Zweiter Plot
372 b=monte(xq,zq,n,lb,sigma)
373 contourf(fig[1, 2],xs, ys, b,levels=levels)
374 contour!(fig[1, 2],ys, xs,b ,levels=levels )
375
376 # Dritter Plot
377 c=monte(xq,zq,n,lc,sigma)
378 contourf(fig[2, 1],xs, ys,c,levels=levels)
379 contour!(fig[2, 1],ys, xs,c ,levels=levels )
380
381 #Vierter Plot
382 d=monte(xq,zq,n,ld,sigma)
383 contourf(fig[2, 2],xs, ys,d,levels=levels)
384 contour!(fig[2, 2],ys, xs,d ,levels=levels )
385
386 #Label
387 Colorbar(fig[1:2,3], limits = (0.00001, 5), colormap = :viridis,flipaxis
= false, label = "kg/s")
388 Label(fig[3, :], text = "x(m)")
389 Label(fig[:, 0], text = "z(m)")
390 Label(fig[0, :], text = "Konzentrationen im Vergleich x = " *string(xq)*
"m z = " *string(zq)*" m bei verschiedenen ", textsize = 30)
391 save( "Bericht/Bilder/3_lambda_x = " *string(xq)*".png", fig)
392
393 #L schen der Grafik aus dem Zwischenspeicher
394 empty!(fig)
395 end
396
397 ### Funktion zur Erstellung des Sigma-Vergleichsplots
398 function sigmaplot(xq,zq)
399     #Initialisierung der Parameter
400     l=5
401     n= 1000
402     siga=1
403     sigb=2
404     sigc=5
405     sigd=10
406
407     #Grafeneinstellungen
408     levels=[0.001,0.01,0.025,0.05,0.5,0.75,1.0,1.25,1.5,2,5]
409     fig = Figure(resolution=(1080,1080))
410     xs=LinRange(0, xgrenz,xgrenz)
411     ys=LinRange(0, zgrenz, zgrenz)
412
413     #Erster Plot
414     a=monte(xq,zq,n,l,siga)
415     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
435     = false, label = "kg/s")
436     Label(fig[3, :], text = "x(m)")
437     Label(fig[:, 0], text = "z(m)")
438     Label(fig[0, :], text = "Konzentrationen im Vergleich x = " *string(xq)*
439     "m z = " *string(zq)*" m bei verschiedenen u & w ", textsize = 30)
440     save("Bericht/Bilder/3_sigma_x = "*string(xq)*".png", fig)
441
442     #L schen der Grafik aus dem Zwischenspeicher
443     empty!(fig)
444 end
445
446 # Main Funktion
447 function main()
448
449     ### Aufgababe a
450     xq = 60.5 # m
451     zq = 0.5 # m
452     einzelplot(xq,zq)
453     partikelanzahlplot(xq,zq)
454     lambdaplot(xq,zq)
455     sigmaplot(xq,zq)
456
457     ### Aufgababe b
458     xq = 15.5 # m
459     zq = 65.5 # m
460     einzelplot(xq,zq)
461     partikelanzahlplot(xq,zq)
462     lambdaplot(xq,zq)
463     sigmaplot(xq,zq)
464 end
465
466 #Ausf hren der Mainfunktion
467 main()
```