

Programmierpraktikum Übung 4

Alexander Steding

10028034

Gottfried Wilhelm Leibniz Universität

22. August 2022

Inhaltsverzeichnis

1	Einleitung	4
2	Aufgabe 1	4
2.1	Aufgabenteil a	4
2.2	Aufgabenteil b	5
3	Aufgabe 2	8
4	Aufgabe 3	9
4.1	Aufgabenteil a	9
4.2	Aufgabenteil b	10
5	Aufgabe 4	10
6	Quellcode	10
6.1	Aufgabe 2	10
6.2	Aufgabe 3	14

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 kontinuierliche 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.

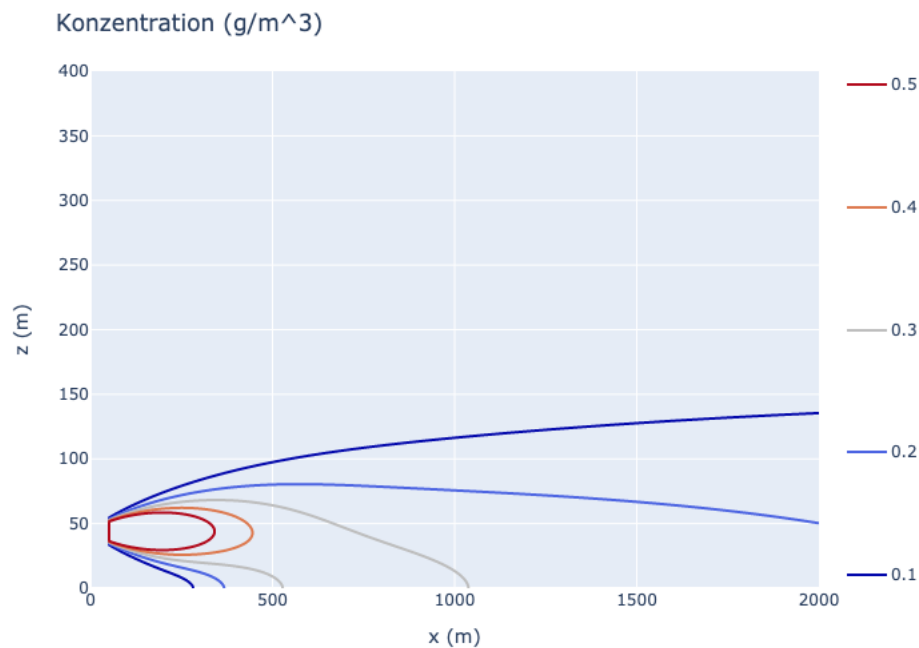


Abb. 1: Konzentrationsverteilung

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

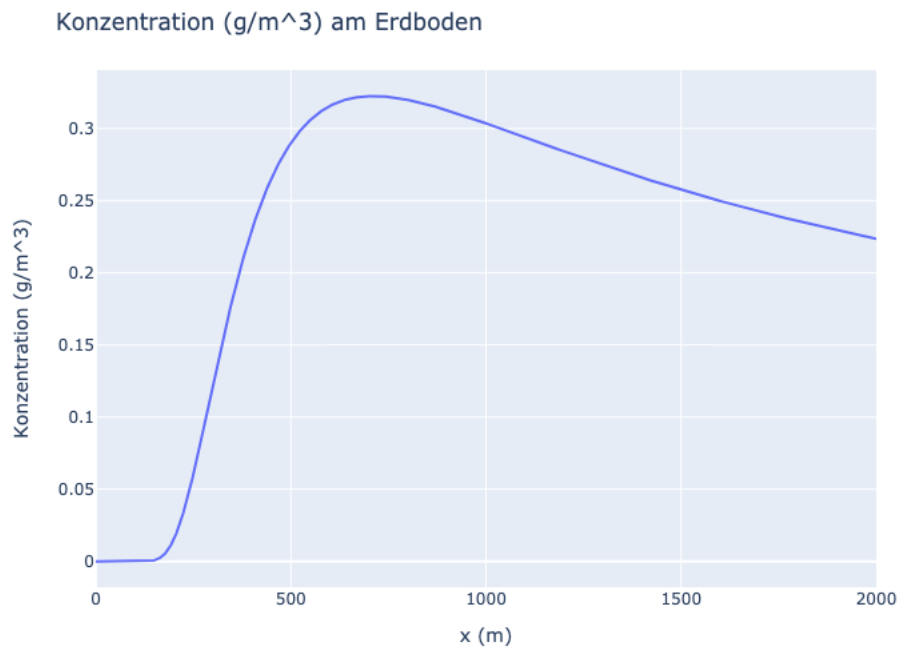


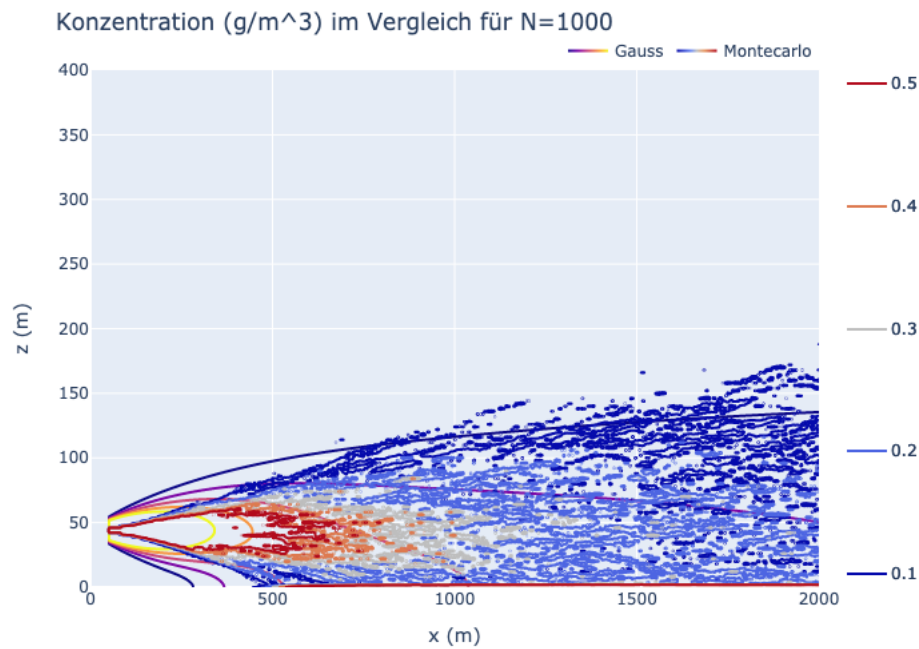
Abb. 2: Konzentrationsverteilung am Erdboden

Die maximale Konzentration wurde final rechnerisch mittels Julia bestimmt als

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

2.2 Aufgabenteil b

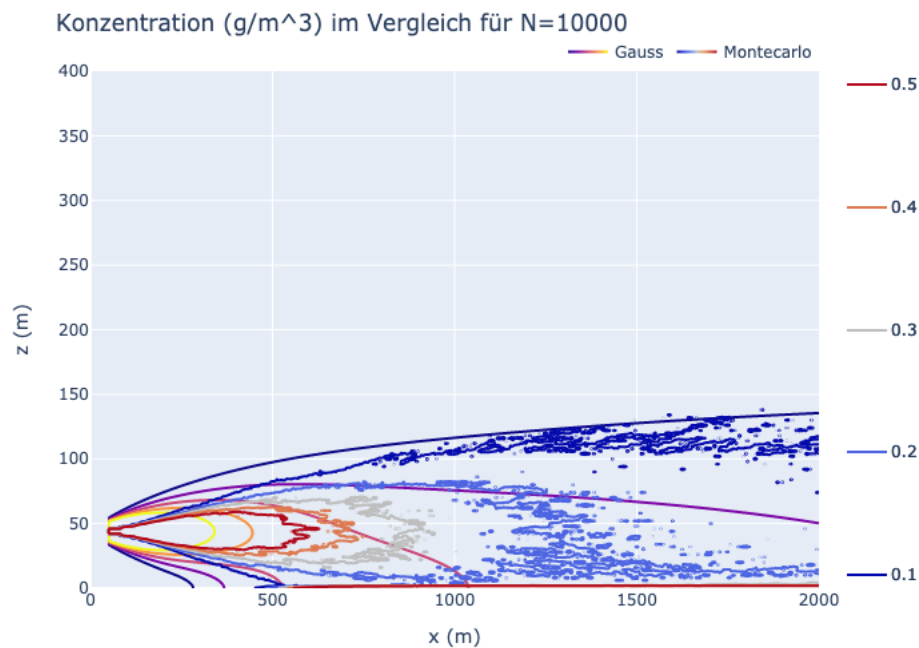
Für den Vergleich zwischen Monte-Carlo-Modell und Gauß-Modell wurden beide Modelle in einem Contour-Plot visualisiert. Für die optimale Visualisierung wurden verschiedene Partikelanzahlen visualisiert.

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

$$N = 1000 \quad (2)$$

sind extrem große Abweichungen zum Gaußmodell zu erkennen.

Abb. 4: Vergleich für $N=10000$

Bei einer mittleren Anzahl wie

$$N = 10000 \quad (3)$$

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

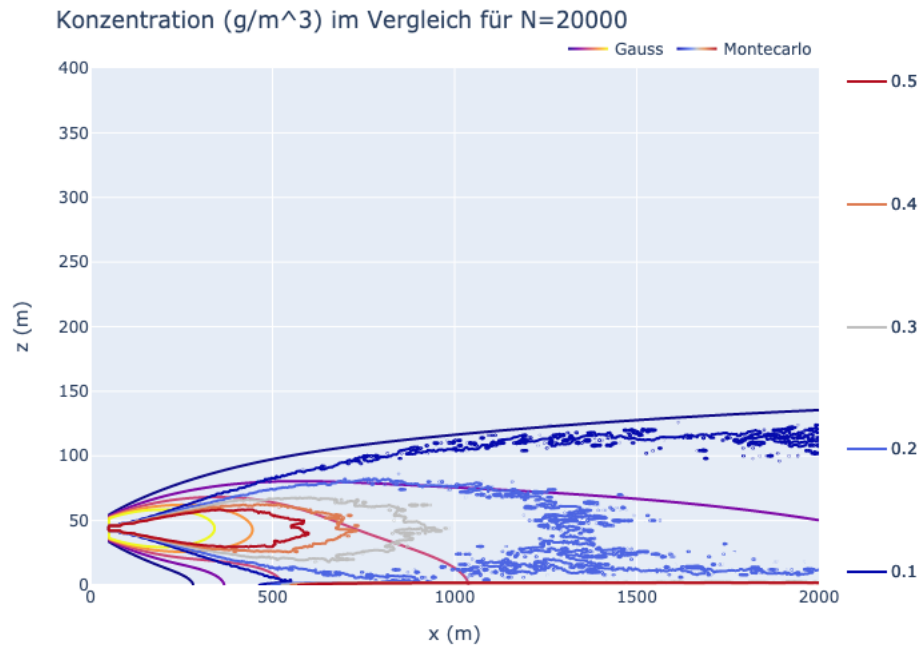


Abb. 5: Vergleich für $N=20000$

Bei einer hohen Anzahl wie

$$N = 20000 \quad (4)$$

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

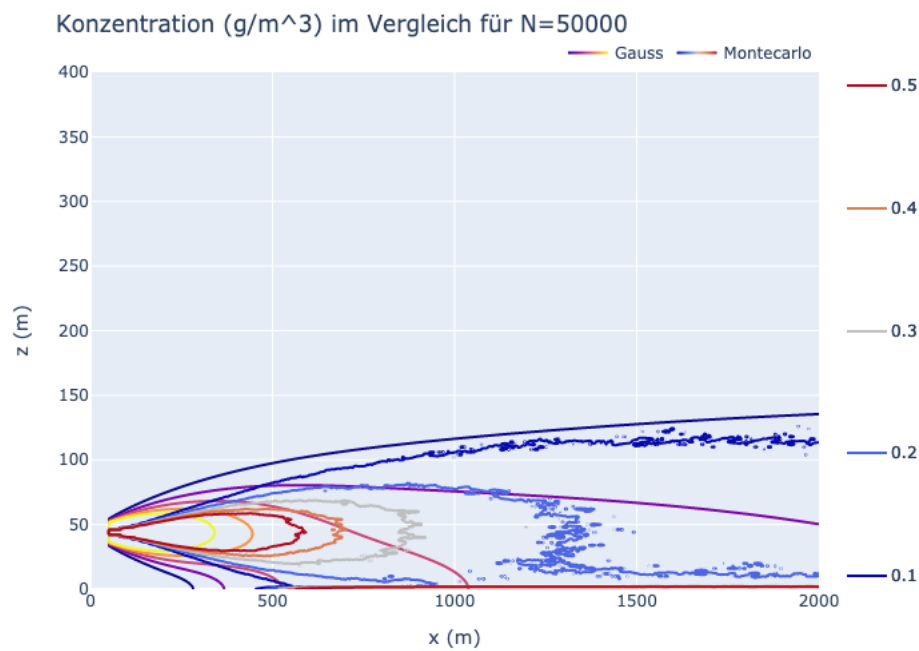
Abb. 6: Vergleich für $N=50000$

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

$$N = 50000 \quad (5)$$

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

$$N = 20000 \quad (6)$$

.

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.

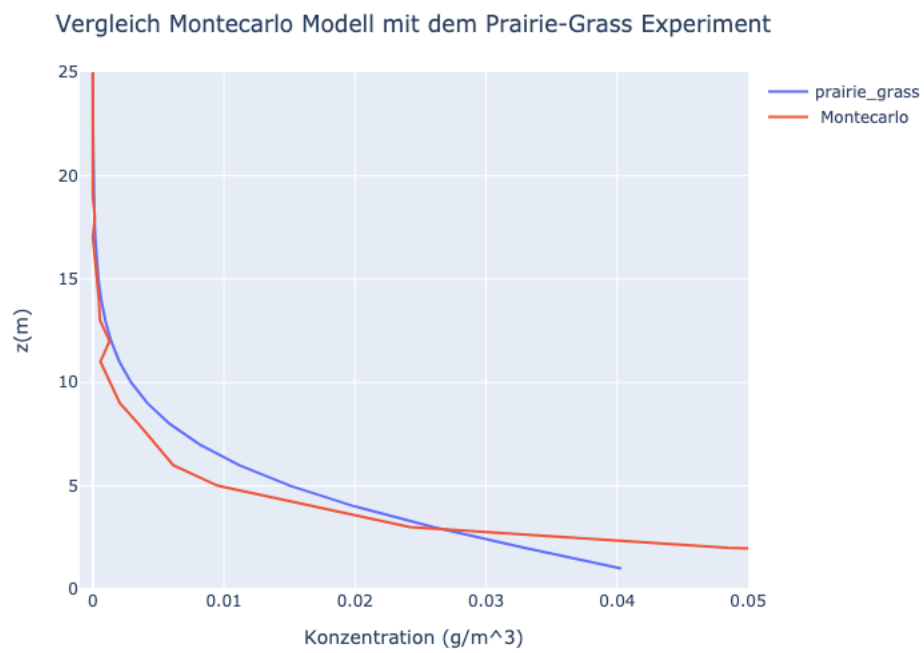


Abb. 7: Vergleich Prairie-Grass

4 Aufgabe 3

4.1 Aufgabenteil a

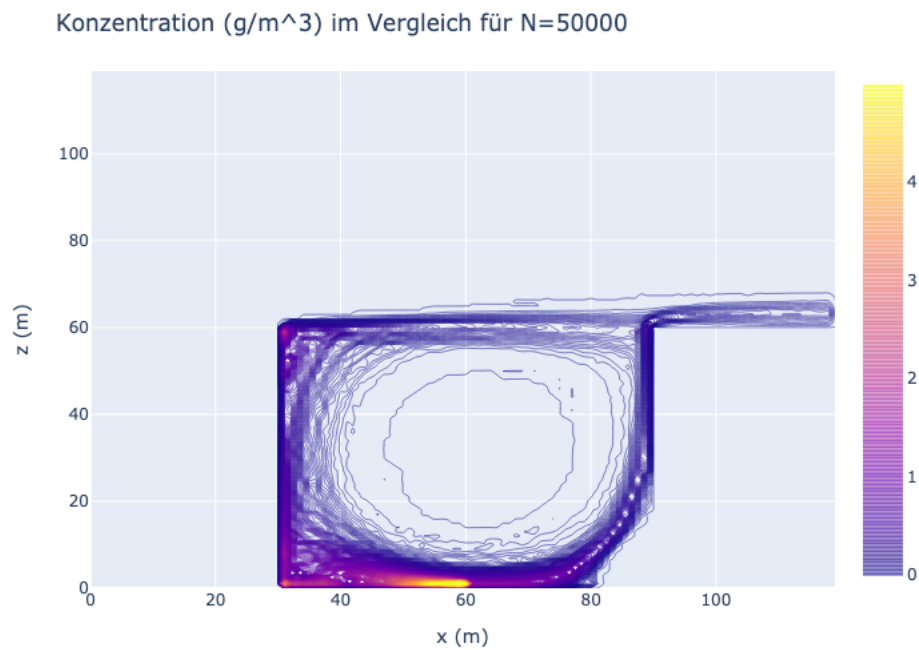


Abb. 8: Konzentrationsverteilung am Erdboden

4.2 Aufgabenteil b

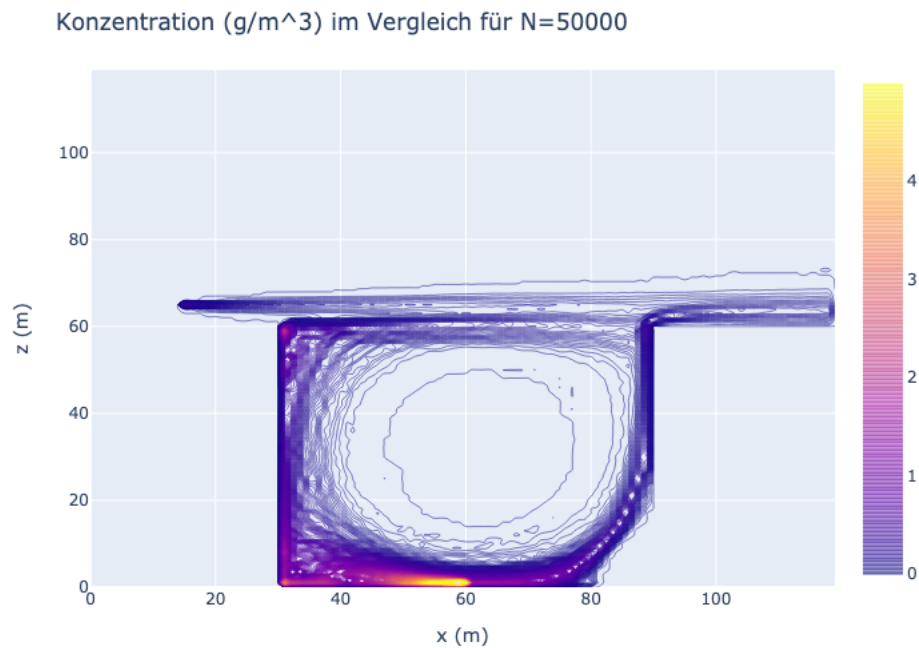


Abb. 9: Konzentrationsverteilung am Erdboden

5 Aufgabe 4

6 Quellcode

6.1 Aufgabe 2

```

1 using NetCDF
2 using Random, Distributions
3 using ProgressBars
4 using PlotsJS
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, ustern, k, znull, q::Float64
9 global units::String
10 global gitter, cd, x, cdground::Array
11
12 n= 10^3           # Anzahl Partikel
13 ubalken = 5       # mittlere Windkomponente in u Richtung in m/s
14 wbalken = 0       # mittlere Windkomponente in w Richtung in m/s
15 zq = 0           # Quellort z Komponente in m
16 xq = 0.5         # Quellort x Komponente in m
17 xgrenz= 110      # Grenze in x Richtung in m
18 zgrenz=25        # Grenze in z Richtung in m
19 tl = 100         #s Zeit
20 dt = 0.4         # Zeitschritt in s
21 dx = 1           # Gitterweite x Richtung in m

```

```
22 dz = 1           # Gitterweite z Richtung in m
23 ui=5             #
24 ustern = 0.35     #
25 k = 0.38         # Kappa
26 znull = 0.008     # Rauigkeitslaenge
27 sigu = 2.5 * ustern # Standartabweichung u m/sm/s
28 sigw = 1.3 * ustern # Standartabweichung w m/s m/s
29 q= 0.7           # Konzentration fuer das Montecarlo Modell in m/s
30 rl= exp(- dt/tl)  # Berechnung von rl
31 units = "g/m^3"   # Einheit fuer Graphen
32
33 ## Arrays Initialisieren
34
35
36 gitter=zeros(xgrenz,zgrenz)
37 konk=zeros(xgrenz,zgrenz)
38
39
40 ##Funktionen ##
41
42 ### Geradengleichung ### Fuer die Exakte Gitterauswertung ist es notwendig
43 function gg(xold, zold, xi, zi, t)
44     xg = xold + t * (xi - xold)
45     zg = zold + t * (zi - zold)
46     if xg>xgrenz
47         xg=xgrenz
48     end
49     if floor(zg) <1
50         zg=1
51     end
52     if floor(xg) <1
53         xg=1
54     end
55     return convert(Int64, floor(xg)), convert(Int64, floor(zg))
56 end
57
58
59 ### Manager###
60 function rangecheck(xi, xold, zi, zold,dt)
61     rangex = floor(xi - xold)
62     rangez = floor(zi - zold)
63     if (rangex + rangez) < 2
64         gitweis(xi, zi,dt)
65     else
66         exaktgitter(xi, xold, zi, zold,dt)
67     end
68 end
69
70 ### Berechnung der Prandtlschicht###
71 function prandltl(zi,xi)
72     if zi < znull
73         ubalken = 0
74     else
75         ubalken = (ustern / k) * log(abs(zi) / znull)
76     end
77
78     tl = ((k * ustern) / sigw ^ 2) * abs(zi)
79     if (0.1*tl)>((k * ustern) / sigw ^ 2) * abs(2)
```

```

80     dt = 0.1*tl                                # Normalfall
81     else
82         dt = ((k * ustern) / sigw ^ 2) * abs(2) #falls dt kleiner als tl in
12 m Hoehe wird dt auf tl(2m) gesetzt
83     end
84     return tl, dt, ui
85 end
86
87 ### Exakte Gitterauswertung###
88 function exaktgitter(xi, xold, zi, zold,dt)
89     ti = []
90     toks = []
91     rangex = convert(Int64,floor(xi - xold))    # Bestimmung der Anzahl an
Schnittpunkten mit der X-Achse
92     rangez = convert(Int64,floor(zi - zold))    # Bestimmung der Anzahl an
Schnittpunkten mit der Z-Achse
93     xsi = ceil(xold)
94     zsi = ceil(zold)
95     for i in 0:rangex
96
97         if i == 0
98             xsi = ceil(xold)
99             push!(toks,(xsi - xold) / (xi - xold))
100         else
101             xsi += 1
102             push!(toks,(xsi - xold) / (xi - xold))
103         end
104     end
105     for i in 0:rangez
106         if i == 0
107             zsi = ceil(zold)
108             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:xcgrenz
```

```
191         for j in 1:zgrenz
192             #print(cd[j,1])
193             push!(pg_mod, konk[i,j])
194         end
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
223         scatter(
224             y=collect(1:zgrenz),
225             x=pg_mod,
226             name=" Montecarlo",
227             showlegend=true ),
228
229         Layout(
230             title="Vergleich Montecarlo Modell mit dem Prairie-Grass Experiment",
231             xaxis_title="Konzentration (" * units * ")",
232             yaxis_title="z(m)",
233             xaxis_range=[-0.001, 0.05],
234             yaxis_range=[0, 25]
235         )), "Bericht/Bilder/2.png")
236     end
237
238
239
240
241 function main()
242     konzentrationen=monte()
243     pg,pg_mod=prairie_grass(konzentrationen)
244     grafen(pg,pg_mod)
245
246 end
247
248
```

```
249 main()
```

6.2 Aufgabe 3

```
1 using NetCDF
2 using Random, Distributions
3 using ProgressBars
4 using LinearAlgebra
5 using PlotlyJS
6 #using SymPy
7 n = 10^3# !Anzahl Partikel
8
9 xgrenz = 120 # !m
10 zgrenz = 120
11 units = "g/m^3" # Einheit fuer Graphen
12 #r = symbols("r")
13 xlist=[]
14 zlist=[]
15
16 dx = 1
17 dy = 1
18 dz = 1
19 ges=[]
20 #ustern = 0.35
21 k = 0.38
22 znull = 0.008
23 #sigu = 2.5 * ustern # m/s
24 #sigw = 1.3 * ustern # m/s
25 q = 1
26 gitter=zeros(xgrenz,zgrenz)
27 konk=zeros(xgrenz,zgrenz)
28
29 marongu = ncread("Bericht/input_uebung5.nc", "u")
30 marongw = ncread("Bericht/input_uebung5.nc", "w")
31 marongus = ncread("Bericht/input_uebung5.nc", "u2")
32 marongws = ncread("Bericht/input_uebung5.nc", "w2")
33 gurkenlist=findall(x->x==-9999.0,marongu)
34 gurkenlistw=findall(x->x==-9999.0,marongw)
35 for i in 1: length(gurkenlist)
36 marongu[gurkenlist[i]]=NaN
37 end
38
39 for i in 1: length(gurkenlistw)
40 marongw[gurkenlist[i]]=NaN
41 end
42
43 println(marongus[61,2])
44 function gg(xold, zold, xi, zi, t)
45
46     xg = xold + t * (xi - xold)
47     zg = zold + t * (zi - zold)
48     if xg>xgrenz
49         xg=xgrenz
50     end
51     if floor(zg) <1
52         zg=1
53     end
54     return convert{Int64, floor(xg)}, convert{Int64, floor(zg)}
```

```
55 end
56
57 function prandltl(zi,xi)
58     xii=convert(Int64,floor(xi))
59     zii=convert(Int64,floor(zi))
60     if floor(zi)==0
61     zii=1
62     end
63
64     tl= 0.05*((k*zii)/(1+k*(zii/5)))/(0.23*sqrt(marongus[xii+1,zii+1]+
65     marongws[xii+1,zii+1]))
66     if (0.1*tl)>0.05*((k*2)/(1+k*(2/5)))/(0.23*sqrt(marongus[xii+1,zii+1]+
67     marongws[xii+1,zii+1])) #falls dt kleiner als tl in 2 m Hoehe
68     dt = 0.1*tl
69     else
70     dt = 0.05*((k*2)/(1+k*(2/5)))/(0.23*sqrt(marongus[xii+1,zii+1]+
71     marongws[xii+1,zii+1]))
72     end
73     return tl, dt
74 end
75
76 function rangecheck(xi, xold, zi, zold,dt)
77     rangex = floor(xi - xold)
78     rangez = floor(zi - zold)
79     if (rangex + rangez) < 2
80     gitweis(xi, zi,dt)
81     else
82     exaktgitter(xi, xold, zi, zold,dt)
83     end
84 end
85
86 function exaktgitter(xi, xold, zi, zold,dt)
87     ti = []
88     toks = []
89     rangex = convert(Int64,floor(xi - xold))
90     rangez = convert(Int64,floor(zi - zold))
91     xsi = ceil(xold)
92     zsi = ceil(zold)
93     for i in 0:rangex
94
95         if i == 0
96             xsi = ceil(xold)
97             push!(toks,(xsi - xold) / (xi - xold))
98         else
99             xsi += 1
100             push!(toks,(xsi - xold) / (xi - xold))
101         end
102     end
103     for i in 0:rangez
104         if i == 0
105             zsi = ceil(zold)
106             push!(toks,(zsi - zold) / (zi - zold))
107         else
108             zsi += 1
109             push!(toks,(zsi - zold) / (zi - zold))
110         end
111     end
112 end
```

```

110     tku = sort!(toks)
111     for i in 2:length(tku)
112         ti = tku[i]
113         told = tku[i - 1]
114         t = mean([told, ti])
115         posx, posz = gg(xold, zold, xi, zi, t)
116         gitter[posx, abs(posz)] += ((tku[i] - tku[i-1]) * dt)
117         konk[abs(posx), abs(posz)] += ((tku[i] - tku[i-1]) * dt * ((q * dt) / (n
* dx * dz)))
118     return
119
120 end
121 end
122
123 function positionen(xi, wi, zi, tl, ui, dt, xold, zold, xolder, zolder)
124
125 ixolder = floor(xolder) + 1
126 izolder = floor(zolder) + 1
127 ixold = floor(xold) + 1
128 izold = floor(zold) + 1
129 izi = floor(zi) + 1
130 ixi = floor(xi) + 1
131 rl = exp(- dt / tl)
132 Random.seed!()
133 d = Normal()
134 rr = rand(d, 1)[1]
135 if izi == 1
136     izi = 2
137 end
138 if ixi == 91
139     ixi = 90
140 end
141 if ixold == 91
142     ixold = 90
143 end
144 if izold == 1
145     izold = 2
146 end
147
148 difqu = abs(marongus[abs(convert(Int64, (ixolder))), abs(convert(Int64, (
izolder)))] - marongus[abs(convert(Int64, (ixi)), abs(convert(Int64, (izi))
)])) / 2 * abs(xolder - xi)
149 difqw = abs(marongws[abs(convert(Int64, (ixolder))), abs(convert(Int64, (
izolder)))] - marongws[abs(convert(Int64, (ixi)), abs(convert(Int64, (izi))
)])) / 2 * abs(xolder - xi)
150
151
152 ukack = rl * ui + sqrt((1 - rl ^ 2)) * sqrt(marongus[abs(convert(Int64, (
ixi))), abs(convert(Int64, (izi)))] * rr + (1 - rl) * tl * difqu)
153 #println(ukack, " mit ", ixi, " und ", izi, " und ", tl)
154 ui = marongu[abs(convert(Int64, (ixold))), abs(convert(Int64, (izold)))] +
ukack
155 xi = xi + ui * dt
156 #print(xi)
157 wkack = rl * wi + sqrt((1 - rl ^ 2)) * sqrt(marongws[abs(convert(Int64, (
ixi))), abs(convert(Int64, (izi)))] * rr + (1 - rl) * tl * difqw)
158 wi = marongw[abs(convert(Int64, (ixold))), abs(convert(Int64, (izold)))] +
wkack

```



```

159     zi = zi + wi * dt
160
161     #print(marongu[abs(convert(Int64,floor(xold))),abs(convert(Int64,floor(
162     zold))), " mit ",xold," zold: ",zold)
163     #println(dt," alla ",tl," neues Glueck ", xi ," ", zi)
164     #println(xi, " ", zi)
165     while ((xi<=31 && zi<=61)|| (xi>=90 && zi <=61)|| (30<=xi<=90 && zi<=1)||
166     zi<1)
167         #print("irgendwas")
168         if (xi>=90 && zi<=61) # rechte Wand
169             #println("alarm rechte Wand")
170             #ui,wi, br= eckendreck(xi,zi,xold,zold,ui,wi)
171             br=1
172             #println(br)
173             if br==1
174                 xi=xi-2*(abs(90-xi))
175                 ui=-ui
176             end
177         elseif (xi<=31 && zi<=61) && zi>1 #linke Wand
178             #println("alarm linke Wand")
179             #ui,wi,br= eckendreck(xi,zi,xold,zold,ui,wi)
180             br=1
181             if br==1
182                 xi=xi+2*(abs(31-xi))
183                 ui=-ui
184             end
185         #elseif abs(zi-60) < abs(zi-1) #Decke
186         # #println("alarm Decke")
187         # wi=-wi
188         # zi=zi+2*(abs(61-zi))
189     else #Boden
190         #println("Boden ist aus Lava")
191         wi=-wi
192         zi=zi+2*(abs(1-zi))
193     end
194 end
195 return xi, wi, zi,ui
196 end
197
198 """
199 function eckendreck(xi,zi,xold,zold,ui,wi)
200     if xi<=30&& typeof(solve(r*[1,1]+[30,60]-[xold,zold],r)) !=Vector{Any}
201     && typeof(solve(r*[1,1]+[30,60]-[xi,zi],r)) !=Vector{Any}
202         ui=-ui
203         wi=-wi
204         xi=xold
205         zi=zold
206         br=2
207     elseif xi<=30&& typeof(solve(r*[1,1]+[30,0]-[xold,zold],r)) !=Vector{Any}
208     } && typeof(solve(r*[1,1]+[30,0]-[xi,zi],r)) !=Vector{Any}
209         ui=-ui
210         wi=-wi
211         xi=xold
212         zi=zold
213         br=2

```

```

213     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}
214         ui=-ui
215         wi=-wi
216         xi=xold
217         zi=zold
218         br=2
219     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}
220         ui=-ui
221         wi=-wi
222         xi=xold
223         zi=zold
224         br=2
225 else
226     ui=ui
227     wi=wi
228     br=1
229 end
230 return ui,wi, br
231 end
232 """
233
234 function gitweis(xi, zi,dt)
235     if floor(zi) <0
236         zi=0
237     end
238     xm = abs(convert(Int64,floor((xi +1))))
239     zm = abs(convert(Int64,floor((zi +1))))
240     gitter[xm, zm] = gitter[xm, zm] + 1
241     konk[xm, zm] += 1*((q * dt)/(n * dx * dz))
242     return
243 end
244
245 function monte(xq,zq)
246
247 for i in ProgressBar(1:n)
248     xi = xq
249     zi = zq
250     dt=0
251     ui=0
252     wi=0
253     xold=xq
254     zold=zq
255     while (ceil(xi+ui*dt) < xgrenz) && (ceil(zi) < zgrenz)
256         xolder=xold
257         zolder=zold
258         xold = xi
259         zold = zi
260         #wi = marongw[abs(convert(Int64,floor(xold))),abs(convert(Int64,
floor(zold)))]
261         #ui= marongu[abs(convert(Int64,floor(xold))),abs(convert(Int64,floor
(zold)))]
262         #print(xi)
263         tl, dt = prandltl(zi,xi)
264         xi, wi, zi,ui = positionen(xi, wi, zi, tl, ui, dt,xold,zold,
xolder,zolder)
265

```

```
266         rangecheck(xi, xold, zi, zold,dt)
267         #println(zi)
268         push!(xlist, xi)
269         push!(zlist, zi)
270     end
271
272 end
273 return xlist, zlist, konk
274 end
275
276 #if isfile("test.nc") == true
277 #    rm("test.nc",force=true)
278 #end
279
280 #nccreate("test.nc", "c", "x", collect(0:xgrenz), "z", collect(0:zgrenz))
281 #ncwrite(konk, "test.nc", "c")
282
283 #plot(xlist,zlist, xlims=(0, 120), ylims=(0,120))
284
285 function grafen(xlist,zlist,konk,xq)
286     quellenstring=string(xq)
287     """
288     savefig(plot(
289
290         scatter(
291             y= zlist,#collect(1:zgrenz),
292             x=xlist,
293             name="prairie_grass",
294             ),
295
296         Layout(
297             title="Montecarlo in der Haeuserschlucht",
298             xaxis_title="x(m)",
299             yaxis_title="z(m)",
300             xaxis_range=[0, 120],
301             yaxis_range=[0, 120]
302         )), "Bericht/Bilder/3_xq="*quellenstring*".png")
303     """
304
305     savefig(plot(contour(x=collect(0:zgrenz),y=collect(0:xgrenz),z=transpose
306     (konk),
307     contours_coloring="lines",
308     #showlabels=true,
309     #labelfont = attr( # label font properties
310         #     size = 12,
311         #     color = "white",
312         # ),
313     #line_width=2,
314     #colorscale="Hot",
315     contours_start=0.0001,
316     contours_end=5,
317     contours_size=0.01,
318     name="Gauss",),
319
320     Layout(
321         title="Konzentration (" *units * ") im Vergleich fuer N=50000",
```

```
323     xaxis_title="x (m)",
324     yaxis_title="z (m)",
325     legend=attr(
326         x=1,
327         y=1,
328         yanchor="bottom",
329         xanchor="right",
330         orientation="h"
331     )
332 ), "Bericht/Bilder/3k_xq="*quellenstring*".png")
333 end
334
335
336
337
338 function main()
339     xq = 60.5 # !m
340     zq = 0.5 # !m
341     xlist,zlist,konk=monte(xq,zq)
342     grafen(xlist,zlist,konk,xq)
343     xq = 15.5 # !m
344     zq = 65.5 # !m
345     xlist,zlist,konk=monte(xq,zq)
346     grafen(xlist,zlist,konk,xq)
347
348 end
349 main()
```