

# Programmierpraktikum Übung 4

Alexander Steding

10028034

Gottfried Wilhelm Leibniz Universität

5. Juli 2022

# 1 Ergebnisse

## 1.1 Vergleichsplot

Gefragt war in dieser Übung nach einem Vergleich der MonteCarlo Methode und dem Prairie-Grass-Experiment. Zusätzlich sollte das MonteCarlo Modell um eine Integration der Prandtlschicht erweitert werden. Eine besondere Herausforderung war zu dem die Integration einer exakten Gitterauswertung. Für die Berechnung der Konzentrationen und das Visualisieren wurde Python verwendet. Dargestellt wurden jeweils die berechnete Konzentrationen mit dem MonteCarlo Modell und die gemessenen Werte aus dem Experiment für verschiedene Höhen.

In dem Praire-Grass-Experiment wurden die Quellstärke nicht weiter konkretisiert, deshalb wurden verschieden Quellstärken ausprobiert. Zunächst mit der Quellstärke

$$Q = 150 \frac{g}{s} \quad (1)$$

Dies liefert den folgenden Graphen:

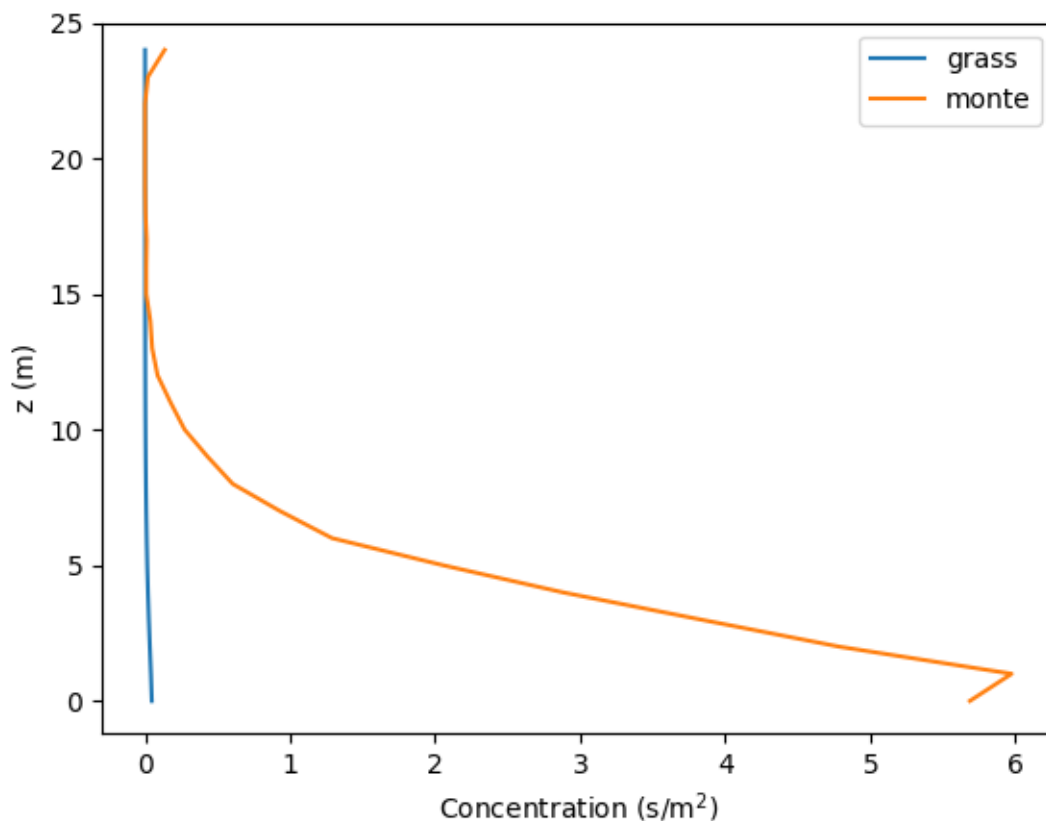


Abb. 1: Vergleich MC und Grass für  $Q=150$

Das MonteCarlo Modell liefert mit diesem  $Q$  nur eine sehr schlechte annäherung für Höhen unter 10m.

Ein  $Q$  von

$$Q = 0.15 \frac{g}{s} \quad (2)$$

liefert folgenden Graphen:

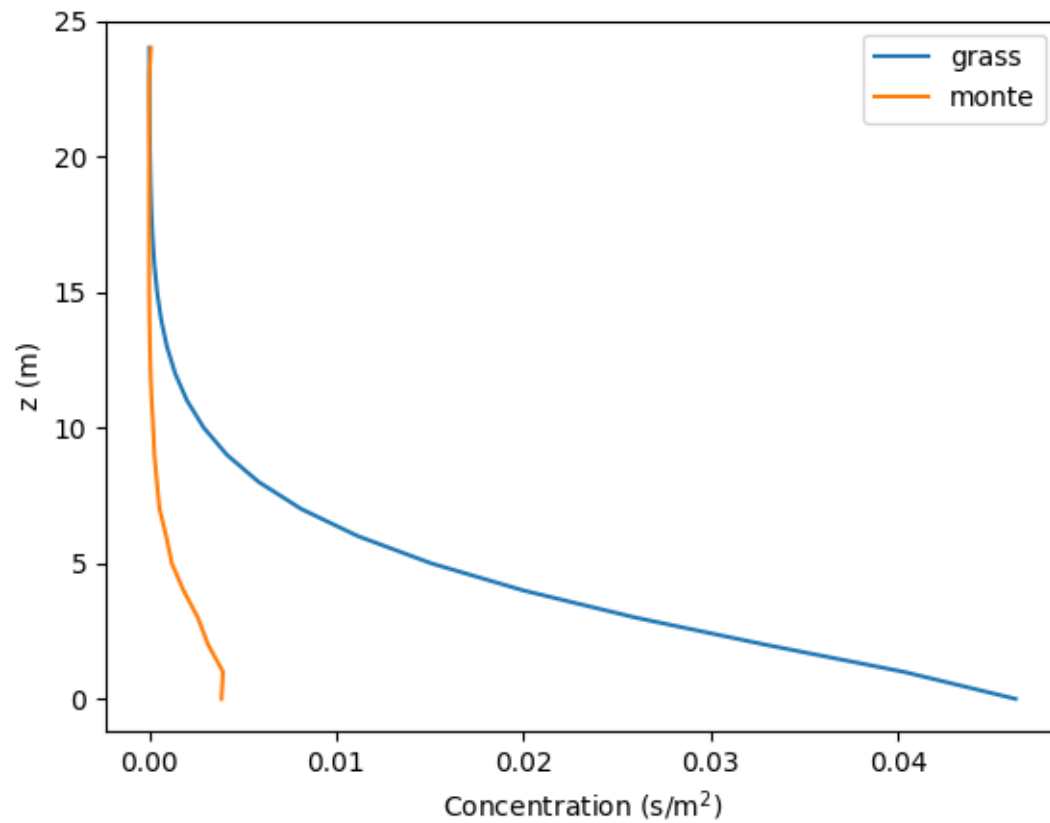


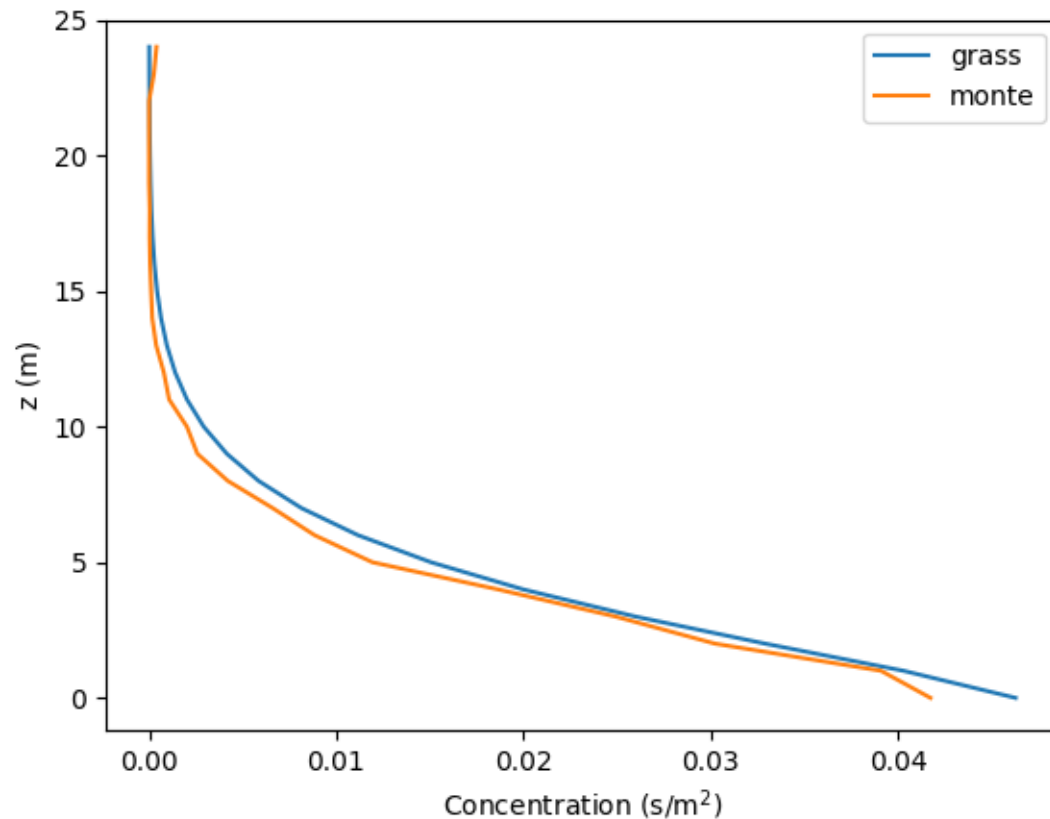
Abb. 2: Vergleich MC und Grass für  $Q=0.15$

Diese Quellstärke ist zu gering und das MonteCarlo liefert auch hier nur eine schlechte Anpassung.

Eine Quellstärke von

$$Q = 1 \frac{g}{s} \quad (3)$$

liefert optimale Ergebnisse:

Abb. 3: Vergleich MC und Grass für  $Q=1$ 

Das MonteCarlo Modell schafft hier eine sehr gute Annäherung an das Experiment über alle Höhenbereiche hinweg. Lediglich bei Höhen unter 2m gibt es eine stärkere Abweichung. Dies könnte in der nicht optimierten komplexen Reflexion unter  $z_0$  liegen, welche in der nächsten Übung in den Ablauf integriert werden kann.

## 2 Quellcode

### 2.1 Python

#### 2.1.1 Angepasstes MC Modell

```

1 from tqdm import tqdm
2 import numpy as np
3 import random
4 import math
5 import netCDF4 as nc4
6 import matplotlib.pyplot as plt
7 from grassmodul import grass
8
9 n = 10**4 # !Anzahl Partikel
10 ubalken = np.float64(5) # !m/s

```

```
11 wbalken = np.float64(0) # !m/s
12 xq = np.float64(0) # !m
13 zq = np.float64(0.5) # !m
14 counter = np.float64(0)
15 xgrenz = 110 # !m
16 zgrenz = 25
17
18 dx = 1
19 dy = 1
20 dz = 1
21
22 ustern = 0.35
23 k = 0.38
24 znull = 0.008
25 sigu = 2.5 * ustern # m/s
26 sigw = 1.3 * ustern # m/s
27
28 gitter = np.zeros((int(xgrenz), int(zgrenz)))
29 konk = np.zeros((int(xgrenz), int(zgrenz)))
30 q = 0.1
31 def gg(xold, zold, xi, zi, t):
32
33     xg = xold + t * (xi - xold)
34     zg = zold + t * (zi - zold)
35     #print("der wert",xg, t, xold,xi)
36     if xg>xgrenz:
37         xg=xgrenz-1
38     return int(xg), int(zg)
39
40 def prandl(zi):
41
42     if zi < znull:
43         ubalken = 0
44     else:
45         ubalken = (ustern / k) * math.log(abs(zi) / znull)
46
47     return ubalken
48
49 def prandltl(zi):
50     tl = ((k * ustern) / sigw ** 2) * abs(zi)
51     if (0.1*tl)>((k * ustern) / sigw ** 2) * abs(2): #falls dt kleiner als
52         tl in 2 m Hoehe
53         dt = 0.1*tl
54     else:
55         dt = ((k * ustern) / sigw ** 2) * abs(2)
56
57     return tl, dt
58
59 def rangecheck(xi, xold, zi, zold):
60     rangex = int(xi - xold)
61     rangez = int(zi - zold)
62     if (rangex + rangez) < 2:
63         gitweis(xi, zi)
64     else:
65         exaktgitter(xi, xold, zi, zold)
66
67 def exaktgitter(xi, xold, zi, zold):
```

```

68     ti = []
69     tj = []
70     toks = []
71     rangex = int(xi - xold)
72     rangez = int(zi - zold)
73
74     for i in range(0, rangex):
75         if i == 0:
76             xsi = math.ceil(xold)
77             toks.append((xsi - xold) / (xi - xold))
78         else:
79             xsi += 1
80             toks.append((xsi - xold) / (xi - xold))
81
82     for i in range(0, rangez):
83         if i == 0:
84             zsi = math.ceil(zold)
85             toks.append((zsi - zold) / (zi - zold))
86         else:
87             zsi += 1
88             toks.append((zsi - zold) / (zi - zold))
89     tku = sorted(toks)
90     for i in range(1, len(tku)):
91         ti = tku[i]
92         told = tku[i - 1]
93         t = np.mean(np.array([told, ti]))
94         posx, posz = gg(xold, zold, xi, zi, t)
95         if posz > zgrenz or posx > xgrenz:
96             break
97         gitter[posx, posz] += (tku[i] - tku[i-1]) * dt
98         konk[posx, posz] += (tku[i] - tku[i-1]) * dt * ((q * dt) / (n * dx * dz))
99     return
100
101
102 def positionen(xi, wi, zi, tl, ui, dt ):
103     rl = math.exp(- dt / tl)
104     rr = np.float64(random.gauss(0, 1))
105     xi = xi + ui * dt
106     wi = rl * wi + math.sqrt((1 - rl ** 2)) * sigw * rr
107     zi = zi + wi * dt
108     return xi, wi, zi
109
110
111 def gitweis(xi, zi):
112     xm = int((xi)-1)
113     zm = int((zi)-1)
114     gitter[xm, zm] = gitter[xm, zm] + 1
115     konk[xm, zm] += 1 * ((q * dt) / (n * dx * dz))
116     return
117
118 for i in tqdm(range(n)):
119     xi = xq
120     zi = zq
121     posi = []
122     wi = wbalken
123     dt=0
124     while (math.ceil(xi + ubalken * dt) < xgrenz) and (math.ceil(zi) <
zgrenz):

```

```

125     xold = xi
126     zold = zi
127
128     if (zi < znull):
129         difz= abs(znull-zi)
130         zi = zi +2*difz
131         wi = -wi
132         tl, dt = prandltl(zi)
133         ui = prandl(zi)
134         xi, wi, zi = positionen(xi, wi, zi, tl, ui, dt )
135         rangecheck(xi, xold, zi, zold)
136
137
138     else:
139         tl, dt = prandltl(zi)
140         ui = prandl(zi)
141         xi, wi, zi = positionen(xi, wi, zi, tl, ui, dt)
142         rangecheck(xi, xold, zi, zold)
143
144 print(np.max(konk))
145 print(np.max(gitter))
146
147 d = nc4.Dataset('alla.nc', 'w', format='NETCDF4') # 'w' stands for write
148 d.createDimension('x', xgrenz)
149 d.createDimension('z', zgrenz)
150 cnet = d.createVariable('c', 'f8', ('z', 'x'))
151 xnet = d.createVariable('x', 'f8', 'x')
152 znet = d.createVariable('z', 'f8', 'z')
153 cnet[:] = np.transpose(konk)
154 znet[:] = np.arange(0., zgrenz, dx, dtype=float)
155 xnet[:] = np.arange(0., xgrenz, dz, dtype=float)
156 d.close()
157 grass()

```

## 2.1.2 Darstellung und Vergleich mit dem Grass Experiment

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 def grass():
4     import matplotlib
5     import matplotlib.pyplot as plt
6     #matplotlib.use('GTK4Agg')
7     import numpy as np
8     from numpy import ma
9     from matplotlib import colors, ticker, cm
10    from netCDF4 import Dataset
11    import math
12
13    def nc_read_from_file_2d_all(filename, varname):
14
15        import numpy as np
16        import sys
17
18        try:
19            f = open(filename)
20            f.close()
21            # print("Load: " + filename + ".")
22        except FileNotFoundError:

```

```
23         print("Error: " + filename + ". No such file. Aborting...")
24         sys.exit(1)
25
26         nc_file = Dataset(filename, "r", format="NETCDF4")
27         tmp_array = np.array(nc_file.variables[varname][:, :], dtype=type(
nc_file.variables[varname]))
28
29         return tmp_array
30
31     def nc_read_from_file_1d_all(filename, varname):
32
33         import numpy as np
34         import sys
35
36         try:
37             f = open(filename)
38             f.close()
39             #         print("Load: " + filename + ".")
40         except FileNotFoundError:
41             print("Error: " + filename + ". No such file. Aborting...")
42             sys.exit(1)
43
44         nc_file = Dataset(filename, "r", format="NETCDF4")
45         tmp_array = np.array(nc_file.variables[varname][:], dtype=type(
nc_file.variables[varname]))
46
47         return tmp_array
48
49     filename = "alla.nc"
50     fileout = "uebung4.png"
51     units = "s/m$^2$"
52
53     conc = nc_read_from_file_2d_all(filename, "c")
54
55     conc = np.where(conc == -9999.0, np.nan, conc)
56     x = nc_read_from_file_1d_all(filename, "x")
57     z = nc_read_from_file_1d_all(filename, "z")
58
59     for i in range(0, len(x)):
60         #print("ja")
61         if (x[i] >= 100.0):
62             print(x[i])
63             print("nein")
64             pg_mod = conc[:, i]
65             break
66
67     c0 = 4.63E-02
68     gamma = 0.68
69     my = 1.3
70     zs = 3.4
71     pg = np.zeros(len(z))
72     for k in range(0, len(z)):
73         pg[k] = c0 * math.exp(-gamma * (z[k] / zs) ** my)
74
75     print("plotting....")
76     fig = plt.figure()
77     ax = plt.axes()
78
```



```
79     ax.plot(pg, z, label='grass')
80     ax.plot(pg_mod, z, label='monte')
81
82     plt.xlabel('Concentration (' + units + ')')
83     plt.ylabel('z (m)')
84     plt.legend()
85     plt.ylim(top=25)
86     plt.show()
87     return
```