Programierpraktikum Übung 4

Alexander Steding
10028034
Gottfried Wilhelm Leibniz Universität

5. Juli 2022

1 Ergebnisse

1.1 Vergleichsplot

Gefragt war in dieser Übung nach den Unterschieden in der Berechnung für eine zufällige Linienquelle nach dem Monte Carlo Modell und dem Gaußmodell. Für die Berechnung der Konzentrationen und das Visualisieren wurde Python verwendet.

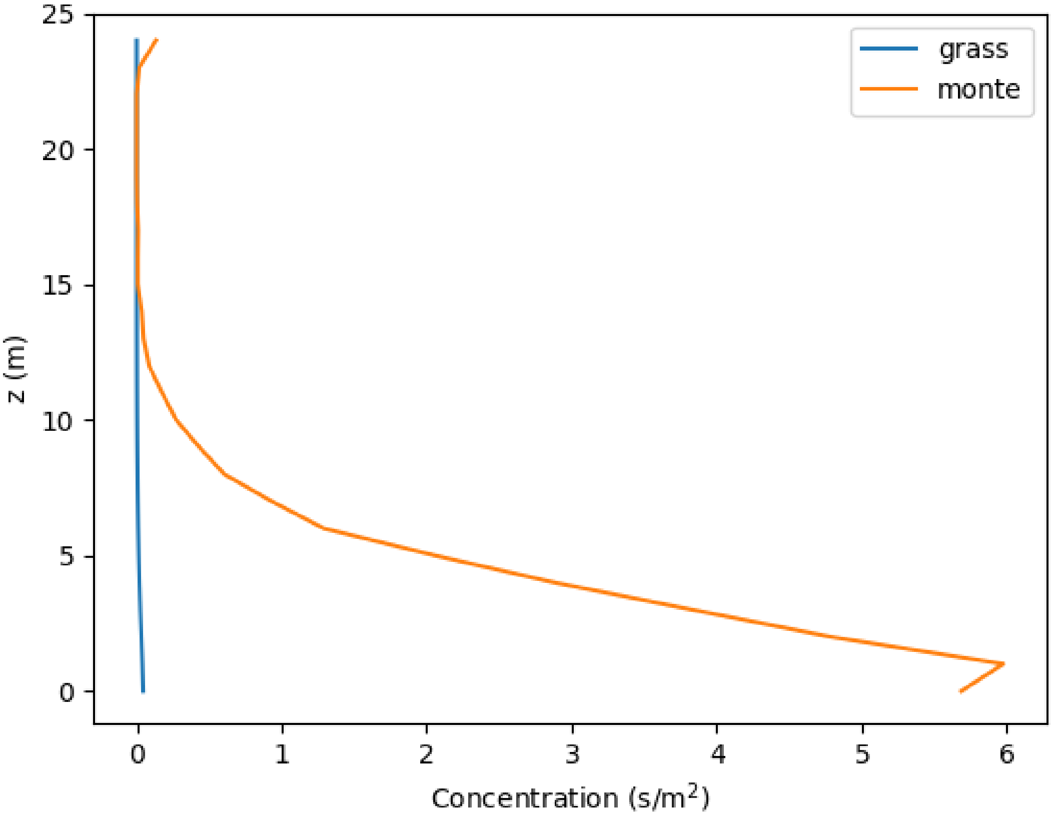

Abb. 1: Vergleich MC und Gauß für Q=150

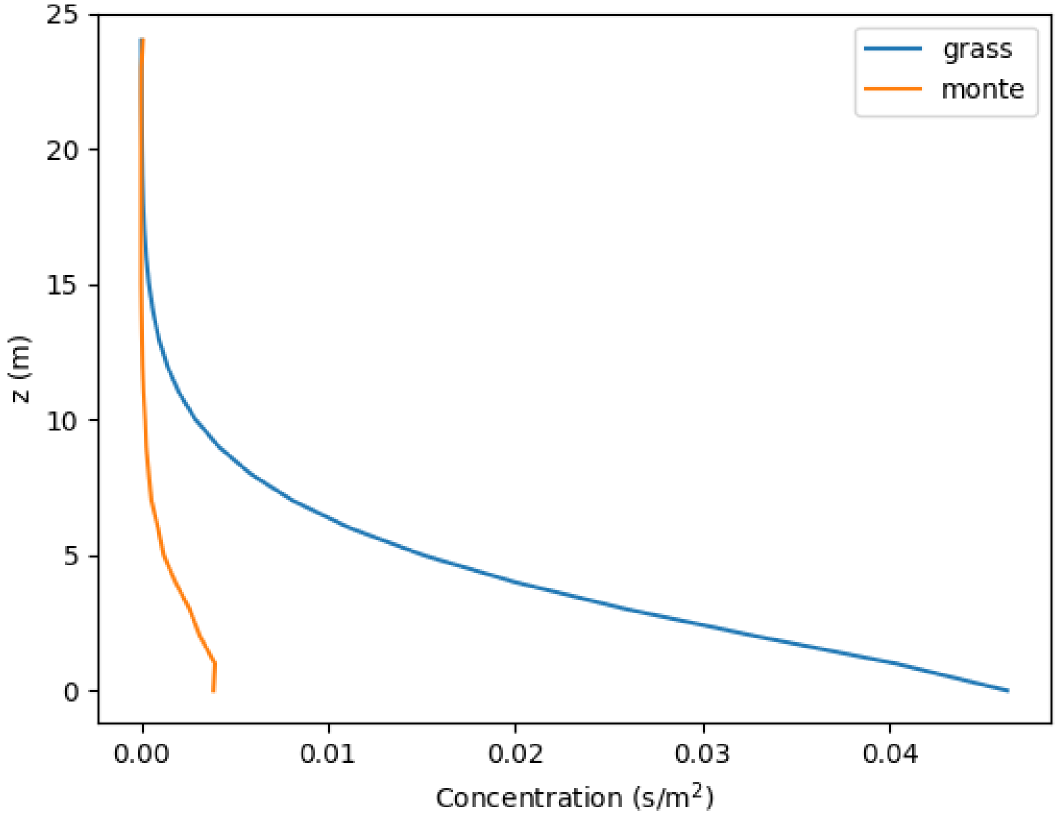

Abb. 2: Vergleich MC und Gauß für Q=0.15

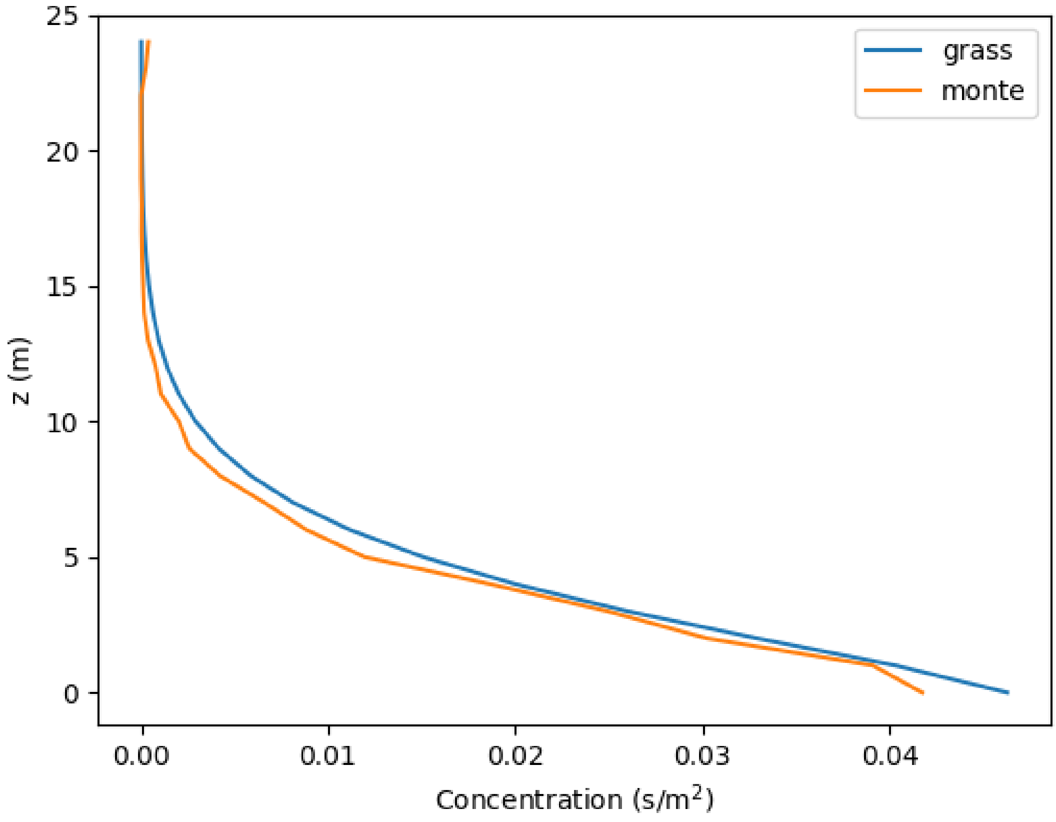

Abb. 3: Vergleich MC und Gauß für Q= 1

$$N = 20000 \tag{1}$$

2 Quellcode

2.1 Python

2.1.1 Vergleich Gauss vs MC

```
from tqdm import tqdm
import numpy as np
import random
import math
import netCDF4 as nc4
import matplotlib.pyplot as plt
from grassmodul import grass

n = 10**4 # !Anzahl Partikel
ubalken = np.float64(5)  # !m/s
wbalken = np.float64(0)  # !m/s
xq = np.float64(0)  # !m
zq = np.float64(0.5)  # !m
```

```
counter = np.float64(0)
xgrenz = 110  # !m
zgrenz = 25

dx = 1
dy = 1
dz = 1

ustern = 0.35
k = 0.38
znull = 0.008
sigu = 2.5 * ustern  # m/s
sigw = 1.3 * ustern  # m/s

gitter = np.zeros((int(xgrenz), int(zgrenz)))
konk = np.zeros((int(xgrenz), int(zgrenz)))
q = 0.1
def gg(xold, zold, xi, zi, t):

    xg = xold + t * (xi - xold)
    zg = zold + t * (zi - zold)
    #print("der wert",xg, t, xold,xi)
    if xg>xgrenz:
        xg=xgrenz-1
    return int(xg), int(zg)

def prandl(zi):

    if zi < znull:
        ubalken = 0
    else:
        ubalken = (ustern / k) * math.log(abs(zi) / znull)

    return ubalken

def prandltl(zi):
    tl = ((k * ustern) / sigw ** 2) * abs(zi)
    if (0.1*tl)>((k * ustern) / sigw ** 2) * abs(2): #falls dt kleiner als tl in 2 m Hoehe
        dt = 0.1*tl
    else:
        dt = ((k * ustern) / sigw ** 2) * abs(2)

    return tl,  dt

def rangecheck(xi, xold, zi, zold):
    rangex = int(xi - xold)
    rangez = int(zi - zold)
    if (rangex + rangez) < 2:
        gitweis(xi, zi)
    else:
        exaktgitter(xi, xold, zi, zold)


def exaktgitter(xi, xold, zi, zold):
    ti = []
    tj = []
    toks = []
```

```python
    rangex = int(xi - xold)
    rangez = int(zi - zold)

    for i in range(0, rangex):
        if i == 0:
            xsi = math.ceil(xold)
            toks.append((xsi - xold) / (xi - xold))
        else:
            xsi += 1
            toks.append((xsi - xold) / (xi - xold))

    for i in range(0, rangez):
        if i == 0:
            zsi = math.ceil(zold)
            toks.append((zsi - zold) / (zi - zold))
        else:
            zsi += 1
            toks.append((zsi - zold) / (zi - zold))
    tku = sorted(toks)
    for i in range(1, len(tku)):
        ti = tku[i]
        told = tku[i - 1]
        t = np.mean(np.array([told, ti]))
        posx, posz = gg(xold, zold, xi, zi, t)
        if posz>zgrenz or posx>xgrenz:
            break
        gitter[posx, posz] += (tku[i] - tku[i-1] )* dt
        konk[posx, posz] += (tku[i] - tku[i-1])* dt*((q * dt)/(n * dx * dz))
    return


def positionen(xi, wi, zi, tl, ui, dt ):
    rl = math.exp(- dt / tl)
    rr = np.float64(random.gauss(0, 1))
    xi = xi + ui * dt
    wi = rl * wi + math.sqrt((1 - rl ** 2)) * sigw * rr
    zi = zi + wi * dt
    return xi, wi, zi


def gitweis(xi, zi):
    xm = int((xi)-1)
    zm = int((zi)-1)
    gitter[xm, zm] = gitter[xm, zm] + 1
    konk[xm, zm] += 1*((q * dt)/(n * dx * dz))
    return

for i in tqdm(range(n)):
    xi = xq
    zi = zq
    posi = []
    wi = wbalken
    dt=0
    while (math.ceil(xi + ubalken * dt) < xgrenz) and (math.ceil(zi) < zgrenz):
        xold = xi
        zold = zi

        if (zi < znull):
            difz= abs(znull-zi)
            zi = zi +2*difz
            wi = -wi
            tl, dt = prandltl(zi)
            ui = prandl(zi)
            xi, wi, zi = positionen(xi, wi, zi, tl, ui, dt )
            rangecheck(xi, xold, zi, zold)


        else:
            tl, dt = prandltl(zi)
            ui = prandl(zi)
            xi, wi, zi = positionen(xi, wi, zi, tl, ui, dt)
            rangecheck(xi, xold, zi, zold)

print(np.max(konk))
print(np.max(gitter))

d = nc4.Dataset('alla.nc', 'w', format='NETCDF4')  # 'w' stands for write
d.createDimension('x', xgrenz)
d.createDimension('z',  zgrenz)
cnet = d.createVariable('c', 'f8', ('z', 'x'))
xnet = d.createVariable('x', 'f8', 'x')
znet = d.createVariable('z', 'f8', 'z')
cnet[:] = np.transpose(konk)
znet[:] = np.arange(0., zgrenz, dx, dtype=float)
xnet[:] = np.arange(0., xgrenz, dz, dtype=float)
d.close()
grass()
```

2.1.2 Darstellung und Vergleich mit dem Grass Experiment

```
# !/usr/bin/env python3
# -*- coding: utf-8 -*-
def grass():
    import matplotlib
    import matplotlib.pyplot as plt
    #matplotlib.use('GTK4Agg')
    import numpy as np
    from numpy import ma
    from matplotlib import colors, ticker, cm
    from netCDF4 import Dataset
    import math

    def nc_read_from_file_2d_all(filename, varname):

        import numpy as np
        import sys

        try:
            f = open(filename)
            f.close()
        #      print("Load: " + filename + ".")
        except FileNotFoundError:
            print("Error: " + filename + ". No such file. Aborting...")
            sys.exit(1)

        nc_file = Dataset(filename, "r", format="NETCDF4")
        tmp_array = np.array(nc_file.variables[varname][:, :], dtype=type(nc_file.variables[varname]))

        return tmp_array

    def nc_read_from_file_1d_all(filename, varname):

        import numpy as np
        import sys

        try:
            f = open(filename)
            f.close()
        #      print("Load: " + filename + ".")
        except FileNotFoundError:
            print("Error: " + filename + ". No such file. Aborting...")
            sys.exit(1)

        nc_file = Dataset(filename, "r", format="NETCDF4")
        tmp_array = np.array(nc_file.variables[varname][:], dtype=type(nc_file.variables[varname]))

        return tmp_array

    filename = "alla.nc"
    fileout = "uebung4.png"
    units = "s/m$^2$"

    conc = nc_read_from_file_2d_all(filename, "c")

    conc = np.where(conc == -9999.0, np.nan, conc)
    x = nc_read_from_file_1d_all(filename, "x")
    z = nc_read_from_file_1d_all(filename, "z")

    for i in range(0, len(x)):
        #print("ja")
        if (x[i] >= 100.0):
            print(x[i])
            print("nein")
            pg_mod = conc[:,i]
            break

    c0 = 4.63E-02
    gamma = 0.68
    my = 1.3
    zs = 3.4
    pg = np.zeros(len(z))
    for k in range(0, len(z)):
        pg[k] = c0 * math.exp(-gamma * (z[k] / zs) ** my)

    print("plotting....")
    fig = plt.figure()
    ax = plt.axes()

    ax.plot(pg, z, label='grass')
    ax.plot(pg_mod, z, label='monte')

```

```
    plt.xlabel('Concentration (' + units + ')')
    plt.ylabel('z (m)')
    plt.legend()
    plt.ylim(top=25)
    plt.show()
    return
```