Programierpraktikum Übung 4

Alexander Steding
10028034
Gottfried Wilhelm Leibniz Universität
18. August 2022

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 kontinuirliche 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.

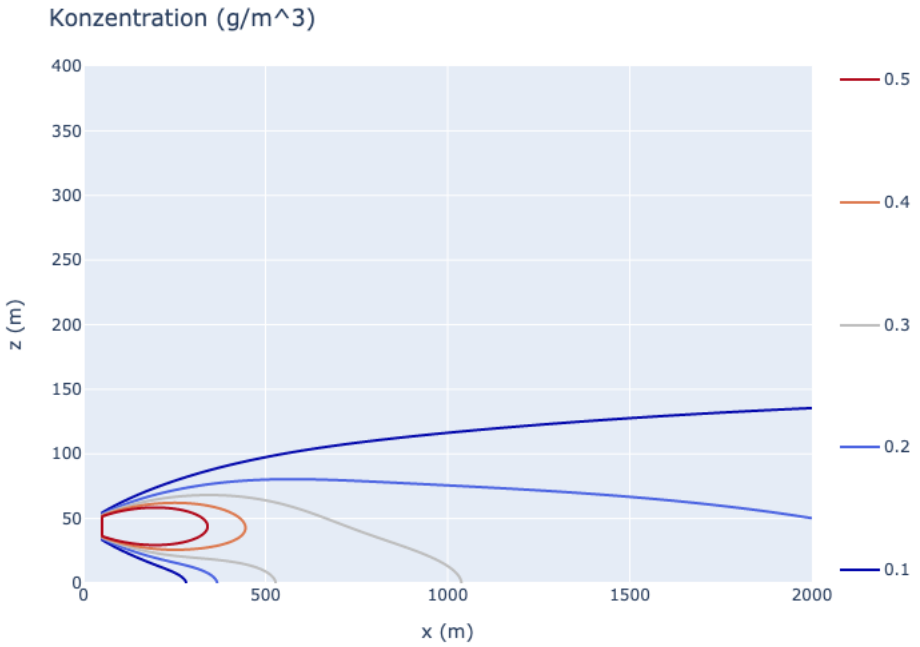

Abb. 1: Konzentrationsverteilung

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

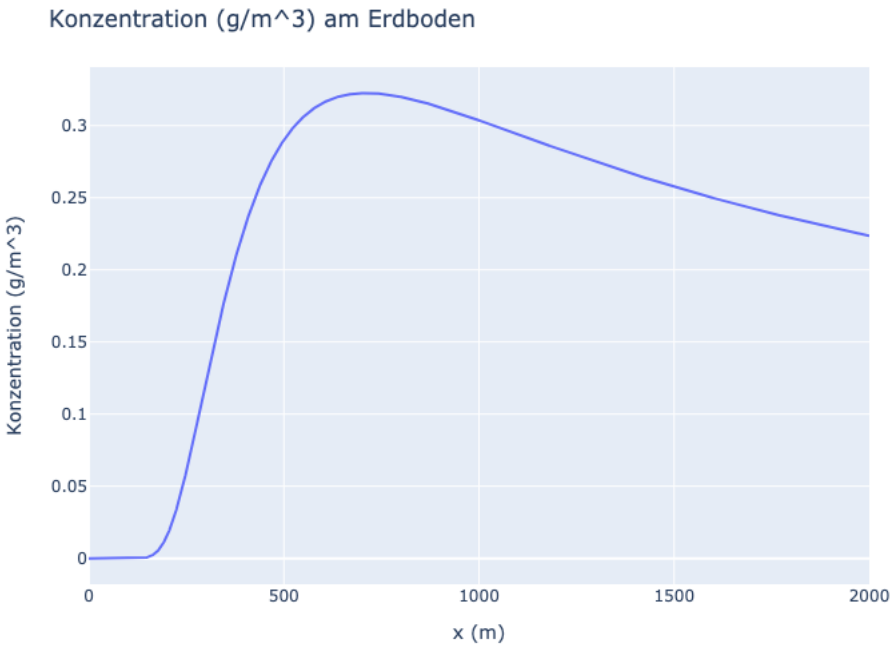

Abb. 2: Konzentrationsverteilung am Erdboden

Die maximale Konzentration wurde final rechnerisch mittels Julia bestimmt als

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

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.

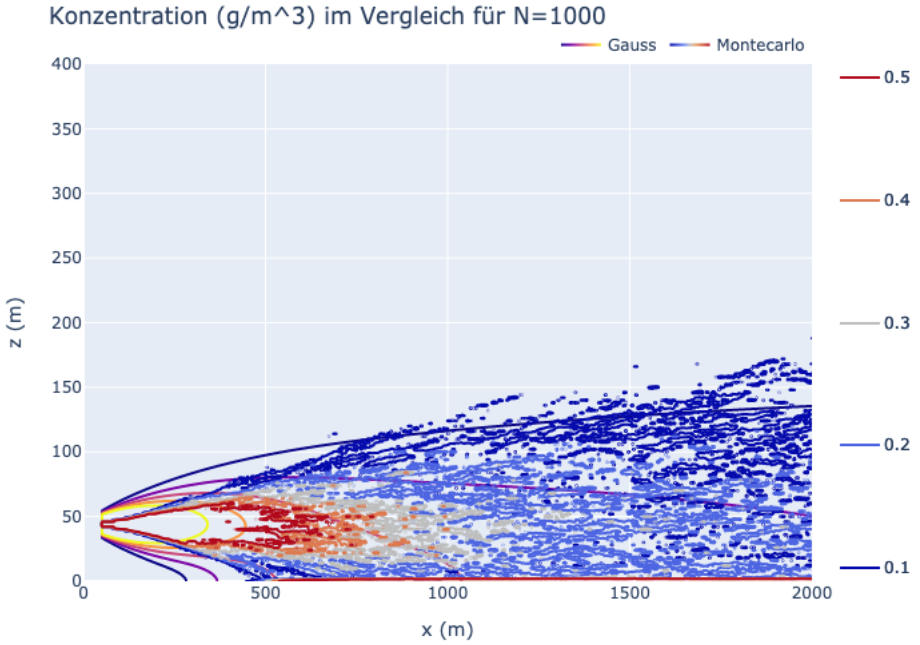

Abb. 3: Vergleich für N=1000

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

$$N = 1000 \tag{2}$$

sind extrem große Abweichungen zum Gaußmodell zu erkennen.

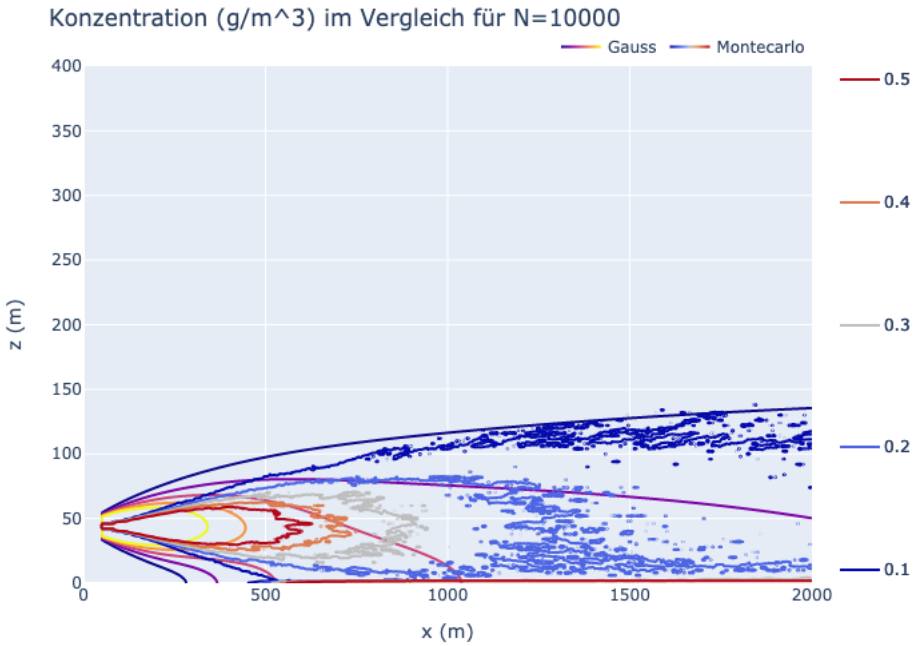

Abb. 4: Vergleich für N=10000

Bei einer mittleren Anzahl wie

$$N = 10000 \tag{3}$$

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

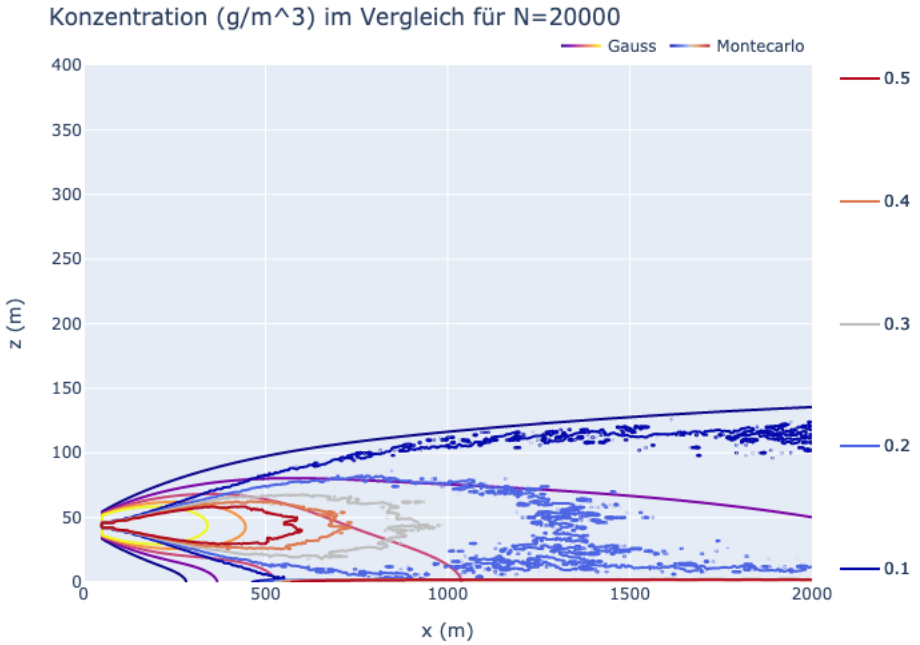

Abb. 5: Vergleich für N=20000

Bei einer hohen Anzahl wie

$$N = 20000 \tag{4}$$

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

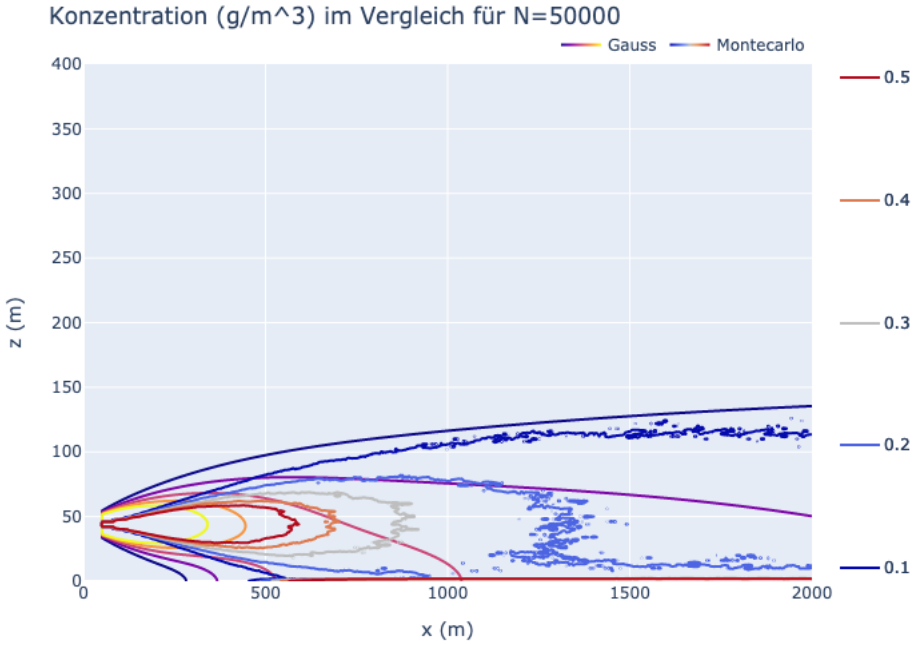

Abb. 6: Vergleich für N=50000

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

$$N = 50000 \tag{5}$$

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

$$N = 20000 \tag{6}$$

.

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.

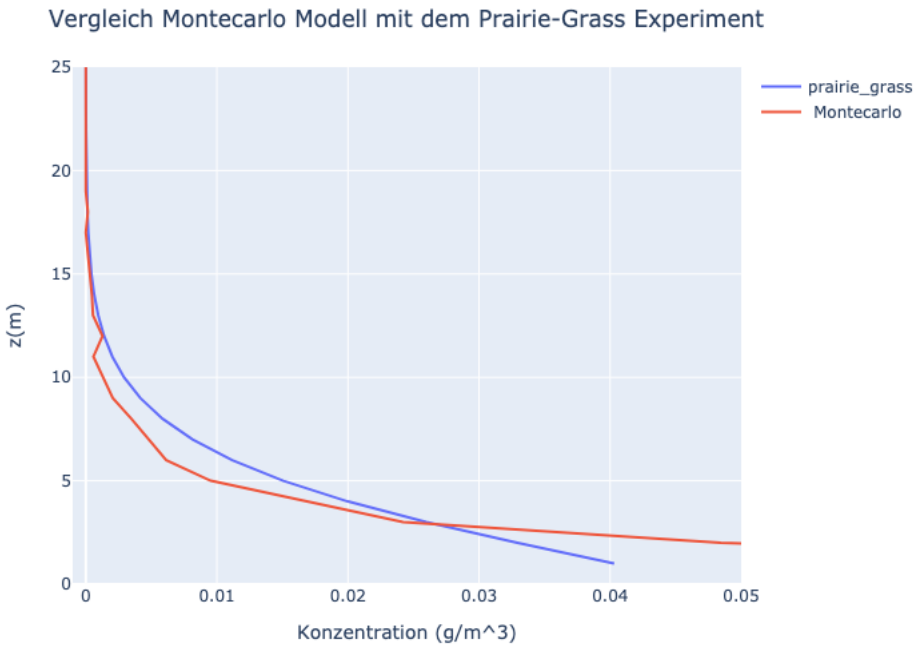

Abb. 7: Vergleich Prairie-Grass

4 Aufgabe 3

4.1 Aufgabenteil a

4.2 Aufgabenteil b

5 Aufgabe 4

6 Quellcode

6.1 Aufgabe 2

```
using NetCDF
using Random, Distributions
using ProgressBars
using PlotlyJS


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
ubalken = 5 #!m/s
wbalken = 0 #!m/s
zq = 0 #!m
xq = 0.5 #!m
```

```
counter=0
xgrenz= 110# !m
zgrenz=25
tl = 100  #s Zeit
dt = 0.4 # Zeitschritt
sigu= 0 #m/s
sigw= 0.39 #m/s

dx = 1
dy = 1
dz = 1
ui=5


## Modellparameter
nx = 2000
nz = 400
dx = 1
dz = 1
q= 0.7 #540 kg/h also 5.4e+8mg/h und so 150000

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

## Array Initialisieren

nxx=nx+1
nzz=nz+1
units = "g/m^3"
gitter=zeros(xgrenz,zgrenz)
konk=zeros(xgrenz,zgrenz)
cd= zeros(nxx,nzz)
x = range(0,nxx)
rl= exp(- dt/tl)

##Funktionen ##
### Geradengleichung ###
function gg(xold, zold, xi, zi, t)
    xg = xold + t * (xi - xold)
    zg = zold + t * (zi - zold)
    if xg>xgrenz
        xg=xgrenz
    end
    if floor(zg) <1
        zg=1
    end
    if floor(xg) <1
        xg=1
    end
    return convert(Int64, floor(xg)), convert(Int64, floor(zg))
end
### Manager###
function rangecheck(xi, xold, zi, zold,dt)
    rangex = floor(xi - xold)
```

```
    rangez = floor(zi - zold)
    if (rangex + rangez) < 2
        gitweis(xi, zi,dt)
    else
        exaktgitter(xi, xold, zi, zold,dt)
    end
end

### Berechnung der Prandtlschicht###
function prandltl(zi,xi)
    xii=convert(Int64,floor(xi))
    zii=convert(Int64,floor(zi))
    if zi < znull
        ubalken = 0
    else
        ubalken = (ustern / k) * log(abs(zi) / znull)
    end

    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)
    end
    return tl,  dt, ui
end

### Exakte Gitterauswertung###
function exaktgitter(xi, xold, zi, zold,dt)
    ti = []
    tj = []
    toks = []
    rangex = convert(Int64,floor(xi - xold))
    rangez = convert(Int64,floor(zi - zold))
    xsi = ceil(xold)
    zsi = ceil(zold)
    for i in 0:rangex

        if i == 0
            xsi = ceil(xold)
            push!(toks,(xsi - xold) / (xi - xold))
        else
            xsi += 1
            push!(toks,(xsi - xold) / (xi - xold))
        end
    end
    for i in 0:rangez
        if i == 0
            zsi = ceil(zold)
            push!(toks,(zsi - zold) / (zi - zold))
        else
            zsi += 1

            push!(toks,(zsi - zold) / (zi - zold))
        end
    end
    tku = sort!(toks)
```

```
    for i in 2:length(tku)
        ti = tku[i]
        told = tku[i - 1]
        t = mean([told, ti])
        posx, posz = gg(xold, zold, xi, zi, t)
        gitter[posx, abs(posz)] += ((tku[i] - tku[i-1] )* dt)
        konk[abs(posx), abs(posz)] += ((tku[i] - tku[i-1])* dt*((q * dt)/(n
   * dx * dz)))
    end
end

### Berechnung der Positionen###
function positionen(xi, wi, zi,tl, ui, dt)

    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

### ungefaehre Gitterauswertung ###
function gitweis(xi, zi,dt)
    if floor(zi) <1
        zi=1
    end
    xm = abs(convert(Int64,floor((xi))))
    zm = abs(convert(Int64,floor((zi))))
    gitter[xm, zm] = gitter[xm, zm] + 1
    konk[xm, zm] += 1*((q * dt)/(n * dx * dz))
    return
end


function monte()
    for i in ProgressBar(1:n+1)
        xi=xq
        zi=zq
        ui=ubalken
        wi=wbalken
        posi=[]
        dt=0


        while (ceil(xi+ui*dt) < xgrenz)
            xold=xi
            zold=zi
            if zi<1
                zi=-zi
                wi= -wi
```

```
                tl, dt,ui = prandltl(zi,xi)
                xi,wi,zi =positionen(xi, wi, zi,tl, ui, dt)
                rangecheck(xi, xold, zi, zold,dt)
                #xm = abs(convert(Int64,round(xi)))+1
                #zm = abs(convert(Int64,round(zi)))+1
                #gitter[xm,zm] = gitter[xm,zm] + 1


            else
                tl, dt,ui = prandltl(zi,xi)
                xi,wi,zi =positionen(xi, wi, zi,tl, ui, dt)
                #xm = abs(convert(Int64,round(xi)))+1
                #zm = abs(convert(Int64,round(zi)))+1
                #if zm> zgrenz
                 #   zm=zgrenz
                #end
                #gitter[xm,zm] = gitter[xm,zm] + 1
                rangecheck(xi, xold, zi, zold,dt)
            end
        end

    end
    return konk
end

function prairie_grass(konk)
    pg_mod= []
    for i in 100:xgrenz
        for j in 1:zgrenz
            #print(cd[j,1])
            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
print(pg_mod)
return pg, pg_mod
end


### Visualisierung ###

 function grafen(pg,pg_mod)
    print("grafen geht los")

    savefig(plot([scatter(y=collect(1:zgrenz),x=pg,
name="prairie_grass",
showlegend=true ,),

```

```
scatter(y=collect(1:zgrenz),x=pg_mod,name=" Montecarlo",
showlegend=true ,)],
Layout(
    title="Vergleich Montecarlo Modell mit dem Prairie-Grass Experiment",
    xaxis_title="Konzentration (" * units * ")",
    yaxis_title="z(m)",
    xaxis_range=[-0.001, 0.05],
    yaxis_range=[0, 25]
)), "Bericht/Bilder/2.png")#,width=1920, height=1080)
 end




 function expo(konzentrationen)
    if  isfile("Bericht/monte.nc") == true
        rm("Bericht/monte.nc",force=true)
    end
    cd=transpose(konzentrationen)
    nccreate("Bericht/monte.nc", "c", "x", collect(0:nxx-1),  "z", collect
   (0:nzz-1))
    ncwrite( konzentrationen,"Bericht/monte.nc", "c")
 end



function main()
    konzentrationen=monte()
    #expo(konzentrationen)
    pg,pg_mod=prairie_grass(konzentrationen)
    grafen(pg,pg_mod)

end
main()
```

6.2 Aufgabe 3