

Auswertung

July 9, 2024

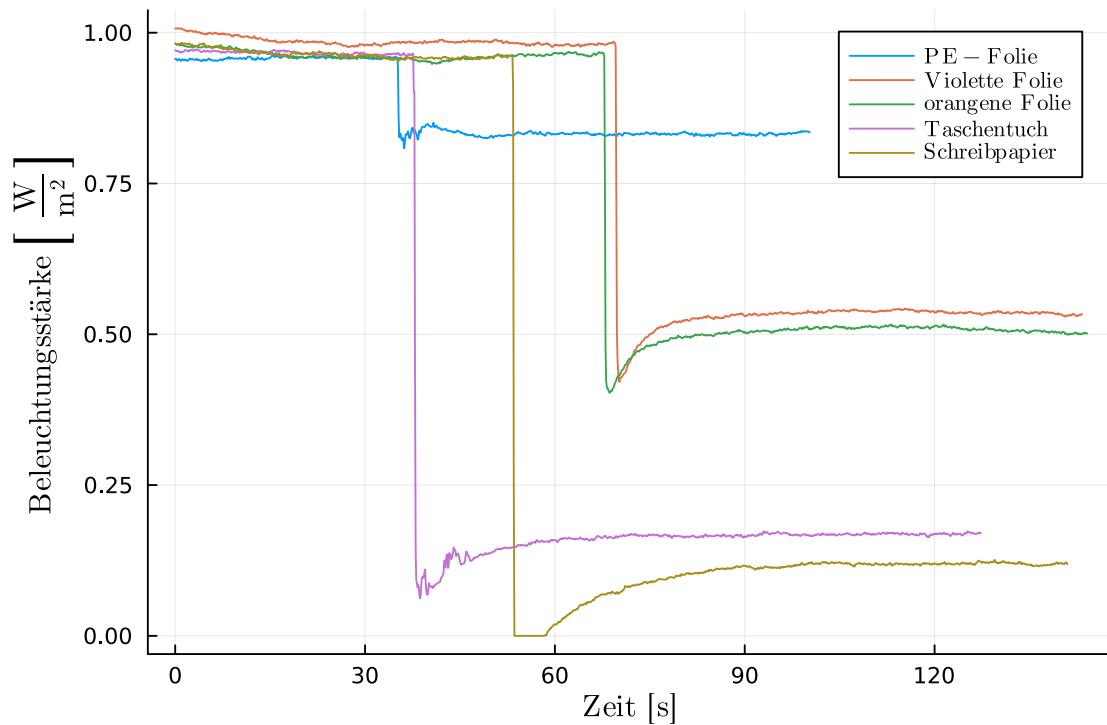
```
[1]: using CSV
using DataFrames
using Plots
using Statistics
using LaTeXStrings
using LinearAlgebra
using LsqFit
```

0.0.1 Auswertungsteil 2: IR-Absorption und -Emission von verschiedenen Materialien

```
[2]: alu = CSV.read("IR-Transmission/Alufolie.csv", DataFrame)
orangeneFolie = CSV.read("IR-Transmission/Orangene_Folie.csv", DataFrame)
taschentuch = CSV.read("IR-Transmission/Papiertaschentuch_1_Lage_doppelt.csv", DataFrame)
peFolie = CSV.read("IR-Transmission/PE-Folie.csv", DataFrame)
schreibpapier = CSV.read("IR-Transmission/Schreibpapier.csv", DataFrame)
violetteFolie = CSV.read("IR-Transmission/Violett-Blaue_Folie.csv", DataFrame);
```

```
[3]: materialien = plot(peFolie.seconds, peFolie.beleuchtungsstaerke, label=L"\mathrm{PE-Folie}")
plot!(violetteFolie.seconds, violetteFolie.beleuchtungsstaerke, label=L"\mathrm{Violette \enspace Folie}")
plot!(orangeneFolie.seconds, orangeneFolie.beleuchtungsstaerke, label=L"\mathrm{orangene \enspace Folie}")
plot!(taschentuch.seconds, taschentuch.beleuchtungsstaerke, label=L"\mathrm{Taschentuch}")
plot!(schreibpapier.seconds, schreibpapier.beleuchtungsstaerke, label=L"\mathrm{Schreibpapier}")
#plot!(alu.seconds, alu.beleuchtungsstaerke, label=L"\mathrm{Aluminium}")
xlabel!(L"\mathrm{Zeit} \enspace [\mathrm{s}]")
ylabel!(L"\mathrm{Beleuchtungsstärke} \enspace \left[\frac{W}{m^2}\right]")
```

[3]:



```
savefig(materialien, “..../media/B1.1/materialien.pdf”);
```

Mittelwert + Fehler Funktion:

```
[4]: function mittelwert(a)
    # Hilfsvariablen:
    n = length(a)

    # Mittelwert:
    mittel = 0
    for i in 1:n
        mittel += a[i]
    end
    mittel = mittel/n

    # Standardabweichung des Mittewerts:
    err = 0
    for i in 1:n
        err += (a[i] - mittel)^2
    end
    err = sqrt( (1 / (n * (n-1))) * err)

    return mittel, err
end
```

[4]: mittelwert (generic function with 1 method)

```
[5]: I_min_all = [minimum(peFolie[:, :beleuchtungsstaerke]), minimum(violetteFolie[:, :beleuchtungsstaerke]), minimum(orangeneFolie[:, :beleuchtungsstaerke]), minimum(taschentuch[:, :beleuchtungsstaerke]), minimum(schreibpapier[:, :beleuchtungsstaerke])]

I_min_error_all = [0.000005, 0.000005, 0.000005, 0.000005, 0.000005]

I_mean_all = [mittelwert(peFolie[601:end, :beleuchtungsstaerke])[1], mittelwert(violetteFolie[901:end, :beleuchtungsstaerke])[1], mittelwert(orangeneFolie[901:end, :beleuchtungsstaerke])[1], mittelwert(taschentuch[751:end, :beleuchtungsstaerke])[1], mittelwert(schreibpapier[901:end, :beleuchtungsstaerke])[1]]

I_mean_error_all = [mittelwert(peFolie[601:end, :beleuchtungsstaerke])[2], mittelwert(violetteFolie[901:end, :beleuchtungsstaerke])[2], mittelwert(orangeneFolie[901:end, :beleuchtungsstaerke])[2], mittelwert(taschentuch[751:end, :beleuchtungsstaerke])[2], mittelwert(schreibpapier[901:end, :beleuchtungsstaerke])[2]]

I_0_all = [mittelwert(peFolie[1:301, :beleuchtungsstaerke])[1], mittelwert(violetteFolie[301:601, :beleuchtungsstaerke])[1], mittelwert(orangeneFolie[301:601, :beleuchtungsstaerke])[1], mittelwert(taschentuch[1:301, :beleuchtungsstaerke])[1], mittelwert(schreibpapier[201:501, :beleuchtungsstaerke])[1]]

I_0_error_all = [mittelwert(peFolie[1:301, :beleuchtungsstaerke])[2], mittelwert(violetteFolie[301:601, :beleuchtungsstaerke])[2], mittelwert(orangeneFolie[301:601, :beleuchtungsstaerke])[2], mittelwert(taschentuch[1:301, :beleuchtungsstaerke])[2], mittelwert(schreibpapier[201:501, :beleuchtungsstaerke])[2]]

T_mean_all = [I_mean_all[1]/I_0_all[1], I_mean_all[2]/I_0_all[2], I_mean_all[3]/I_0_all[3], I_mean_all[4]/I_0_all[4], I_mean_all[5]/I_0_all[5]]

T_min_all = [I_min_all[1]/I_0_all[1], I_min_all[2]/I_0_all[2], I_min_all[3]/I_0_all[3], I_min_all[4]/I_0_all[4], I_min_all[5]/I_0_all[5]]
```

Fehler per Gaußscher Fehlerfortpflanzung:

```
T_mean_error(i) = sqrt( (I_mean_error_all[i] / I_0_all[i])^2 + (I_mean_all[i] * I_0_error_all[i] / I_0_all[i]^2)^2 )
T_min_error(i) = sqrt( (I_min_error_all[i] / I_0_all[i])^2 + (I_min_all[i] * I_0_error_all[i] / I_0_all[i]^2)^2 )

T_mean_error_all = [T_mean_error(1), T_mean_error(2), T_mean_error(3), T_mean_error(4), T_mean_error(5)]
T_min_error_all = [T_min_error(1), T_min_error(2), T_min_error(3), T_min_error(4), T_min_error(5)]
```

;

```
[6]: df = DataFrame(
    Material=["PE", "Violett", "Orange", "Taschentuch", "Schreibpapier"],
```

```

I_min = I_min_all,
I_mean = I_mean_all,
I_mean_error = I_mean_error_all,
I_0 = I_0_all,
I_0_error = I_0_error_all,
T_mean = T_mean_all,
T_mean_err = T_mean_error_all,
T_min = T_min_all,
T_min_err = T_min_error_all
)
show(df, allcols = true)

```

5×10 DataFrame

Row	Material	I_min	I_mean			
I_mean_error	I_0	I_0_error	T_mean			
T_mean_err	T_min	T_min_err				
	String	Float64	Float64	Float64		
	Float64	Float64	Float64	Float64		
	Float64	Float64				

```

 1 PE          0.80835  0.832129   8.52815e-5  0.957965  0.000138624
0.868643 0.00015403  0.84382    0.000122218
 2 Violett      0.42075  0.53625    0.00011692  0.983375  0.000143505
0.545316 0.00014307  0.427863   6.26452e-5
 3 Orange        0.4029   0.508392   0.000166094  0.958185  0.000233947
0.530578 0.000216401  0.420482   0.000102796
 4 Taschentuch   0.06222  0.167711   9.25885e-5  0.967341  0.00013491
0.173373 9.87213e-5  0.0643206  1.03531e-5
 5 Schreibpapier 0.0       0.118639   0.000121608  0.960271  0.000197376
0.123548 0.000129161  0.0       5.20687e-6

```

0.0.2 Auswertungsteil 3: IR-Absorption in CO_2 bei bekannter Konzentration

[7]:

```

m11 = CSV.read("Konzentrationskalibrierung/1ml.csv", DataFrame)
m12 = CSV.read("Konzentrationskalibrierung/2ml.csv", DataFrame)
m13 = CSV.read("Konzentrationskalibrierung/3ml.csv", DataFrame)
m14 = CSV.read("Konzentrationskalibrierung/4ml.csv", DataFrame)
m15 = CSV.read("Konzentrationskalibrierung/5ml.csv", DataFrame)
m110 = CSV.read("Konzentrationskalibrierung/10ml.csv", DataFrame)
m115 = CSV.read("Konzentrationskalibrierung/15ml.csv", DataFrame)
m120 = CSV.read("Konzentrationskalibrierung/20ml.csv", DataFrame)
m130 = CSV.read("Konzentrationskalibrierung/30ml.csv", DataFrame)
m145 = CSV.read("Konzentrationskalibrierung/45ml.csv", DataFrame)
m160 = CSV.read("Konzentrationskalibrierung/60ml.csv", DataFrame)

```

```

mlsaettigung = CSV.read("Konzentrationskalibrierung/Saettigung_CO2.csv", DataFrame);
;

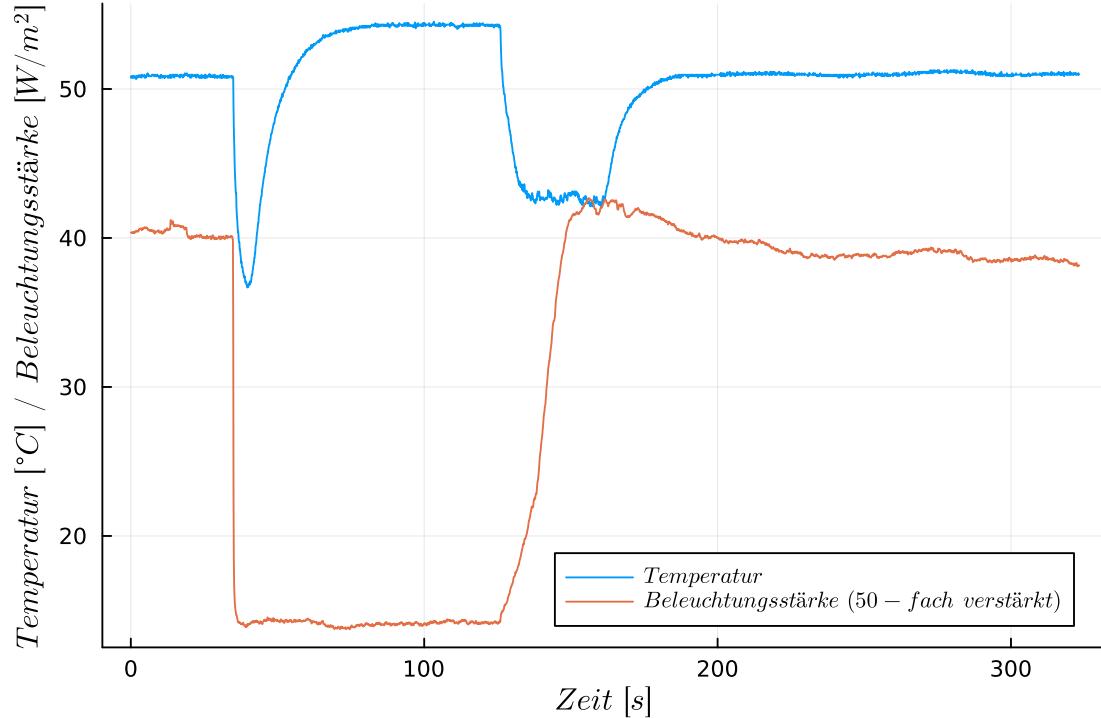
```

```

[8]: plot(mlsaettigung.seconds, mlsaettigung.temperature, label = L"Temperatur", ↴
       legend=:bottomright)
plot!(mlsaettigung.seconds, mlsaettigung.beleuchtungsstaerke.*50,
      label = L"Beleuchtungsstärke \enspace (50-fach \enspace verstärkt)")
xaxis!(L"Zeit \enspace [s]")
yaxis!(L"Temperatur \enspace [°C] \enspace / \enspace Beleuchtungsstärke\enspace ↴
      \enspace [W/m^2]")

```

[8]:

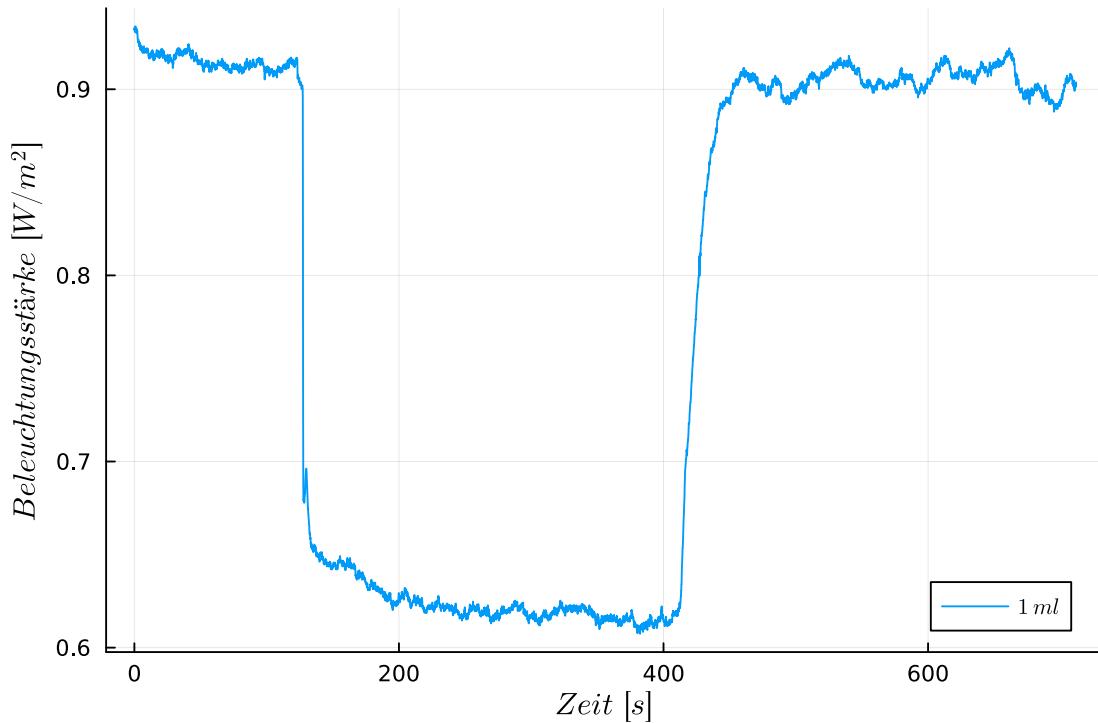


```

[9]: ml1SecondsCut = ml1[1301:end,:seconds] .-130
plot(ml1SecondsCut, ml1.beleuchtungsstaerke[1301:end], label=L"1 \, ml", ↴
      legend=:bottomright)
xlabel!(L"Zeit \enspace [s]")
ylabel!(L"Beleuchtungsstärke \enspace [W/m^2]")

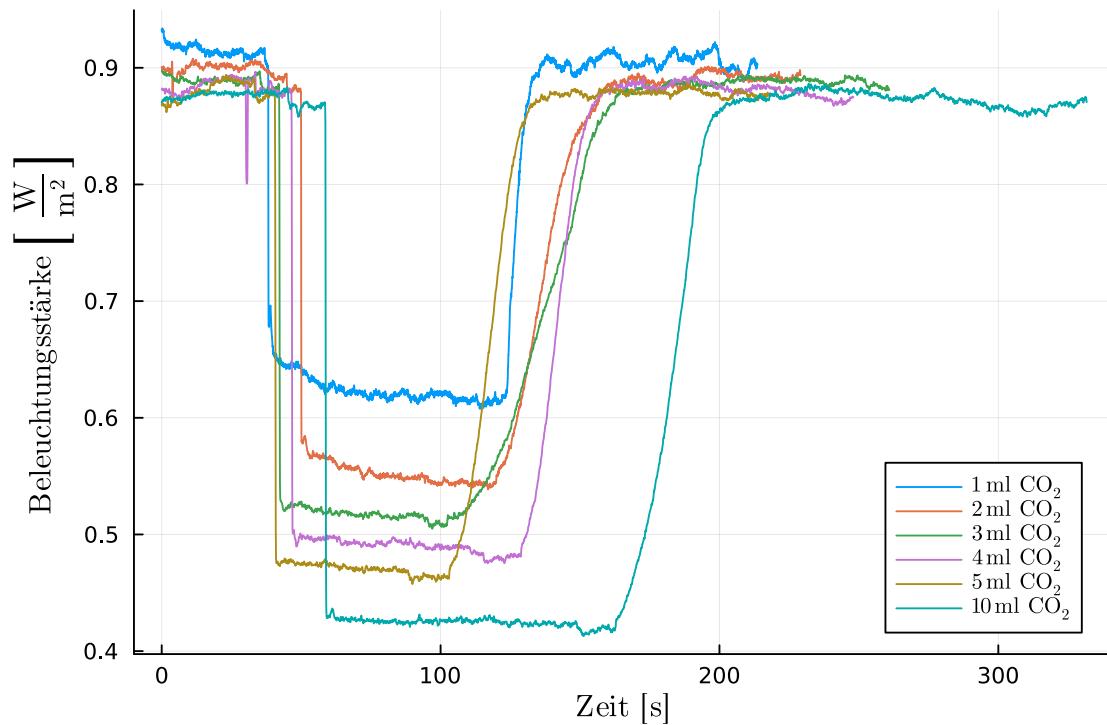
```

[9]:



```
[10]: data1 = plot(ml1SecondsCut.*0.3, ml1.beleuchtungsstaerke[1301:end], label = L"1 \u21d3, \u21d3\mathbf{ml}\u21d3\enspace\mathbf{C0_2}\u21d3")
plot!(ml2.seconds.*0.6, ml2.beleuchtungsstaerke, legend=:bottomright, label=L"2 \u21d3, \u21d3\mathbf{ml}\u21d3\enspace\mathbf{C0_2}\u21d3")
plot!(ml3.seconds, ml3.beleuchtungsstaerke, label=L"3 \u21d3, \u21d3\mathbf{ml}\u21d3\enspace\mathbf{C0_2}\u21d3")
plot!(ml4.seconds, ml4.beleuchtungsstaerke, label=L"4 \u21d3, \u21d3\mathbf{ml}\u21d3\enspace\mathbf{C0_2}\u21d3")
plot!(ml5.seconds, ml5.beleuchtungsstaerke, label=L"5 \u21d3, \u21d3\mathbf{ml}\u21d3\enspace\mathbf{C0_2}\u21d3")
plot!(ml10.seconds, ml10.beleuchtungsstaerke, label=L"10 \u21d3, \u21d3\mathbf{ml}\u21d3\enspace\mathbf{C0_2}\u21d3")
xlabel!(L"\u21d3\mathbf{Zeit}\u21d3 \u21d3\enspace [\u21d3\mathbf{s}]\u21d3")
ylabel!(L"\u21d3\mathbf{Beleuchtungsst\u00e4rke}\u21d3 \u21d3\enspace \u21d3\left[\u21d3\mathbf{\frac{W}{m^2}}\u21d3}\u21d3\right]\u21d3")
```

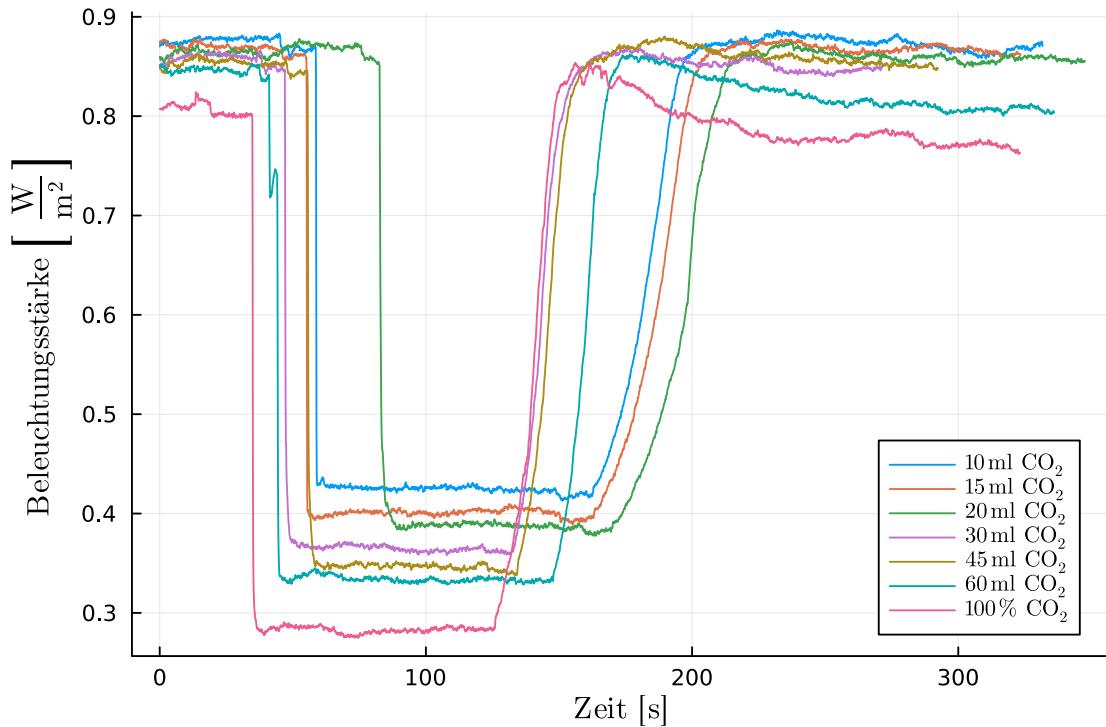
[10]:



```
savefig(data1, “..../media/B1.1/Konzentrationskalibrierung_data_1.pdf”);
```

```
[11]: data2 = plot(ml10.seconds, ml10.beleuchtungsstaerke, label=L"10 \,\u2225\mathbf{m}^2\mathbf{s}^{-1}\mathbf{CO}_2")
plot!(ml15.seconds, ml15.beleuchtungsstaerke, label=L"15 \,\u2225\mathbf{m}^2\mathbf{s}^{-1}\mathbf{CO}_2")
plot!(ml20.seconds, ml20.beleuchtungsstaerke, label=L"20 \,\u2225\mathbf{m}^2\mathbf{s}^{-1}\mathbf{CO}_2")
plot!(ml30.seconds, ml30.beleuchtungsstaerke, label=L"30 \,\u2225\mathbf{m}^2\mathbf{s}^{-1}\mathbf{CO}_2")
plot!(ml45.seconds, ml45.beleuchtungsstaerke, label=L"45 \,\u2225\mathbf{m}^2\mathbf{s}^{-1}\mathbf{CO}_2")
plot!(ml60.seconds, ml60.beleuchtungsstaerke, label=L"60 \,\u2225\mathbf{m}^2\mathbf{s}^{-1}\mathbf{CO}_2")
plot!(mlsaettigung.seconds, mlsaettigung.beleuchtungsstaerke, label=L"100 \,\u2225\mathbf{m}^2\mathbf{s}^{-1}\mathbf{CO}_2")
xlabel!(L"\mathbf{Zeit} \,\u2225\mathbf{s}")
ylabel!(L"\mathbf{Beleuchtungsst\u00e4rke} \,\u2225\mathbf{W/m}^2")
```

```
[11]:
```



```
savefig(data2, “..../media/B1.1/Konzentrationskalibrierung_data_2.pdf”);
```

Lambert-Beer-Gauß-Funktion:

```
[12]: I_0 = 1 # ?
px = 50 # = _0 * x ?
_0 = 0
Δ = 1 # =
C = 10 # (Testwert) 2. Parameter
I() = I_0 * exp(- px * exp(- 0.5 * (( - _0)/Δ)^2) * C) # 0.5 steht nicht im
˓→Text aber ist in Bild!!!
```

```
[12]: I (generic function with 1 method)
```

```
[13]: C = 0.001
plot(I, xaxis=[-5,5], yaxis=[0,1], legend=:bottomright, label="\$ C = \$C \$", u
˓→title="mit 0.5 statt 1")
C = 0.005
plot!(I, label="\$ C = \$C \$")
C = 0.02
plot!(I, label="\$ C = \$C \$")
C = 0.05
plot!(I, label="\$ C = \$C \$")
C = 0.1
```

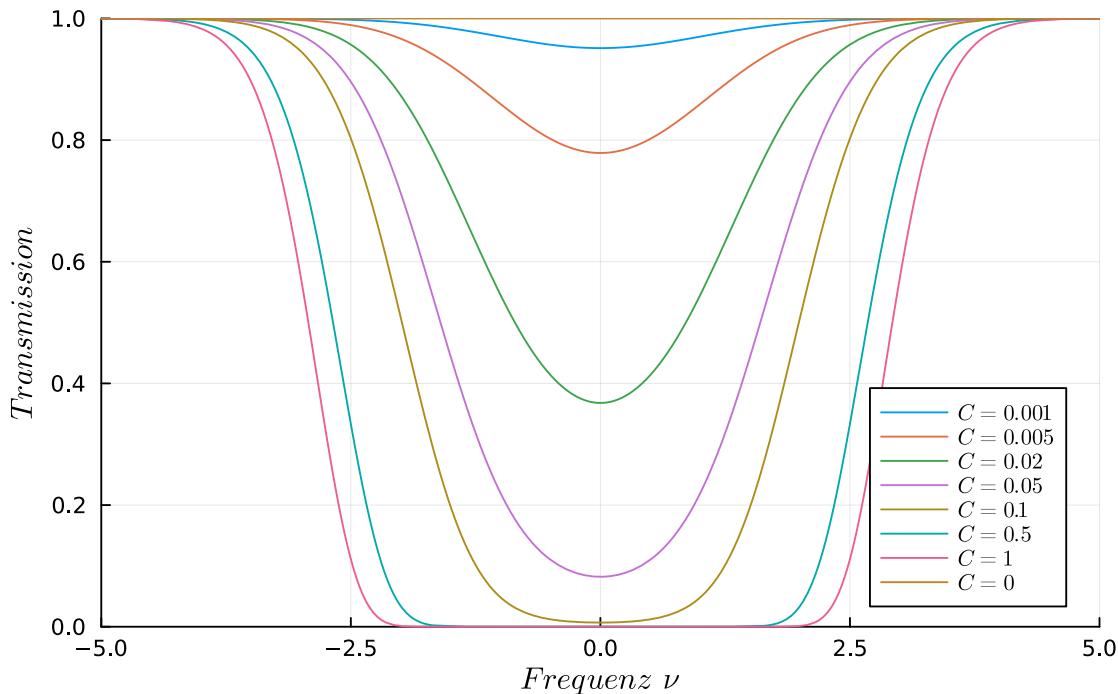
```

plot!(I, label="\$ C = \$C \$")
C = 0.5
plot!(I, label="\$ C = \$C \$")
C = 1
plot!(I, label="\$ C = \$C \$")
C = 0
plot!(I, label="\$ C = \$C \$")
xlabel!(L"Freu\nenz \enspace ")
ylabel!(L"Transmission")

```

[13]:

mit 0.5 statt 1



Messwerte:

```

[14]: volumenZelle = 15 *   * (3.93/2)^2 # cm^3 = ml
volumenZelle_err = sqrt( (0.5 *   * (3.93/2)^2)^2 + (15 *   * 3.93/2 * 0.005)^2 )
# Fehler des Zellvolumens = 0

konzentrationen = [0, 1/volumenZelle, 2/volumenZelle, 3/volumenZelle, 4/
                   ↪volumenZelle, 5/volumenZelle, 10/volumenZelle,
                   15/volumenZelle, 20/volumenZelle, 30/volumenZelle, 45/volumenZelle, 60/
                   ↪volumenZelle, 1]
# Fehler der CO2 Mengen per Spritze: geschätzt auf 0.1 ml
vol_co2_err = 0.1 # ml (geschätzt)

```

```

konz_err_gauß(a) = sqrt( (vol_co2_err/volumenZelle)^2 + (a * volumenZelle_err /  

    ↪volumenZelle^2)^2 )
konzentrationen_err = [0, konz_err_gauß(1), konz_err_gauß(2), konz_err_gauß(3),  

    ↪konz_err_gauß(4), konz_err_gauß(5),
    konz_err_gauß(10), konz_err_gauß(15), konz_err_gauß(20), konz_err_gauß(30),  

    ↪konz_err_gauß(45), konz_err_gauß(60), 0]
transmissionswerte = [1, 0.61891/0.91470, 0.54653/0.90009, 0.51851/0.88955, 0.  

    ↪49127/0.88510,
    0.47169/0.88250, 0.42554/0.87689, 0.40244/0.86985, 0.38815/0.86675, 0.36510/  

    ↪0.86045, 0.34723/0.85468,
    0.33312/0.84604, 0.28149/0.80104]
# Fehler per Gaußscher Fehlerfortpflanzung mittels einzelner Fehler aus  

    ↪Mittelwertbildung von CASSY Lab:
trans_err_gauß(a, b, Δa, Δb) = sqrt( (Δa / b)^2 + (a * Δb / b^2)^2 )
transmissionswerte_err = [0, trans_err_gauß(0.61891, 0.91470, 0.00009, 0.00011),
    trans_err_gauß(0.54653, 0.90009, 0.00012, 0.00019), trans_err_gauß(0.51851,  

    ↪0.88955, 0.00016, 0.00013),
    trans_err_gauß(0.49127, 0.88510, 0.00011, 0.00030), trans_err_gauß(0.47169,  

    ↪0.88250, 0.00013, 0.00040),
    trans_err_gauß(0.42554, 0.87689, 0.00007, 0.00012), trans_err_gauß(0.40244,  

    ↪0.86985, 0.00012, 0.00013),
    trans_err_gauß(0.38815, 0.86675, 0.00008, 0.00017), trans_err_gauß(0.36510,  

    ↪0.86045, 0.00011, 0.00012),
    trans_err_gauß(0.34723, 0.85468, 0.00010, 0.00016), trans_err_gauß(0.33312,  

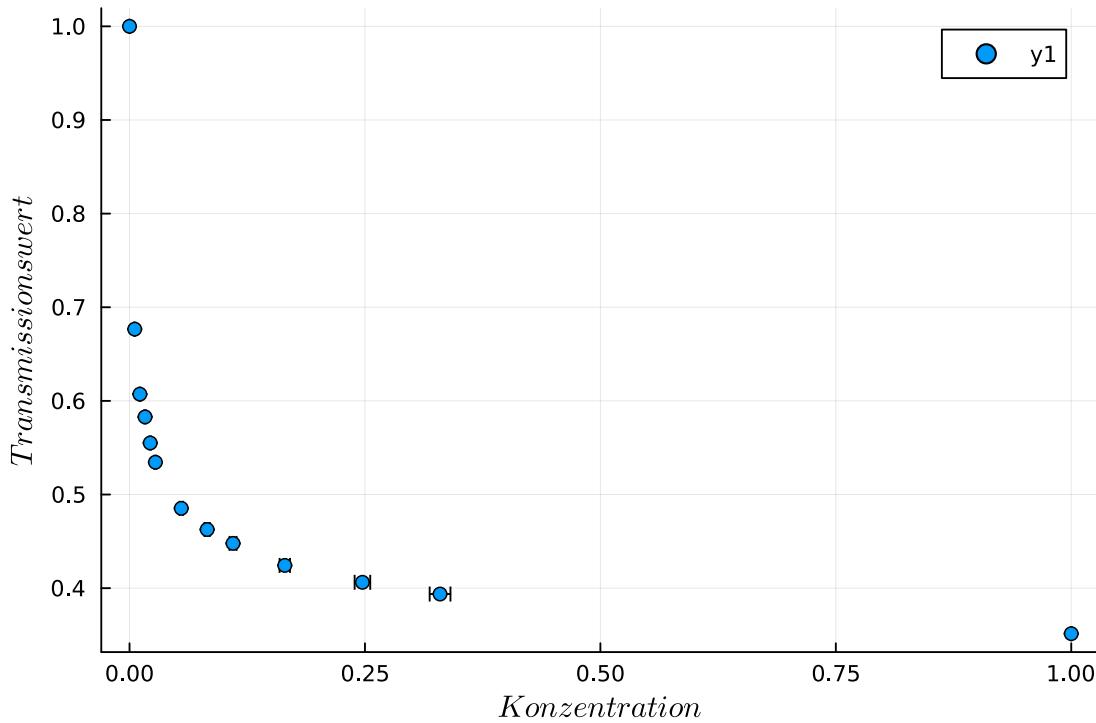
    ↪0.84604, 0.00008, 0.00014),
    trans_err_gauß(0.28149, 0.80104, 0.00014, 0.00012)]
;

```

```
[15]: # Plot
scatter(konzentrationen, transmissionswerte, xerr = konzentrationen_err, yerr =  

    ↪transmissionswerte_err)
xlabel!(L"Konzentration")
ylabel!(L"Transmissionswert")
```

[15]:



Integralannäherung: - Idee: Schrittweite * Summer über alle Funktionswerte - $\nu_1 = -10$, $\nu_2 = 10$

```
[16]: function t_rel(px_given)
    I_0 = 1 # ?
    px = px_given
    _0 = 0
    Δ = 1 # = ?
    C = 10 # (Testwert) 2. Parameter
    I_t() = I_0 * exp(- px * exp(- 1 * (( - _0)/Δ)^2) * C) # statt 1 ist 0.5
    ↪ für die Form im Bild verwendet worden!
    _1 = -10
    _2 = 10
    schrittweite = 0.01

    summen = Array{Float64}(undef, length(konzentrationen))
    for i in 1:length(konzentrationen)
        C = konzentrationen[i]
        for j in _1:schrittweite:_2
            summen[i] += I_t(j)
        end
    end
    integrale = schrittweite .* summen
```

```

T_rel = (integrale .- integrale[length(konzentrationen)]) ./ (integrale[1] -  

integrale[length(konzentrationen)])
offset = transmissionswerte[length(konzentrationen)]
faktor = 1-offset
T_rel_skaliert = faktor .* T_rel .+ offset
# integrale[length(konzentrationen)] = T(100%)
# integrale[1] = T(0%)
return integrale, T_rel, T_rel_skaliert
end

```

[16]: t_rel (generic function with 1 method)

Plot beide zusammen:

```

[17]: # Plot
tkFit = scatter(konzentrationen.*100, transmissionswerte.*100,  

label=L"\mathrm{Messwerte}", xaxis=[0,100],  

xerr = konzentrationen_err.*100, yerr = transmissionswerte_err.*100)  

i = 1000  

plot!(konzentrationen.*100, t_rel(i)[3].*100, label=L"\mathrm{Fit} \enspace mit \enspace px = 10^3", markeralpha=0.5)  

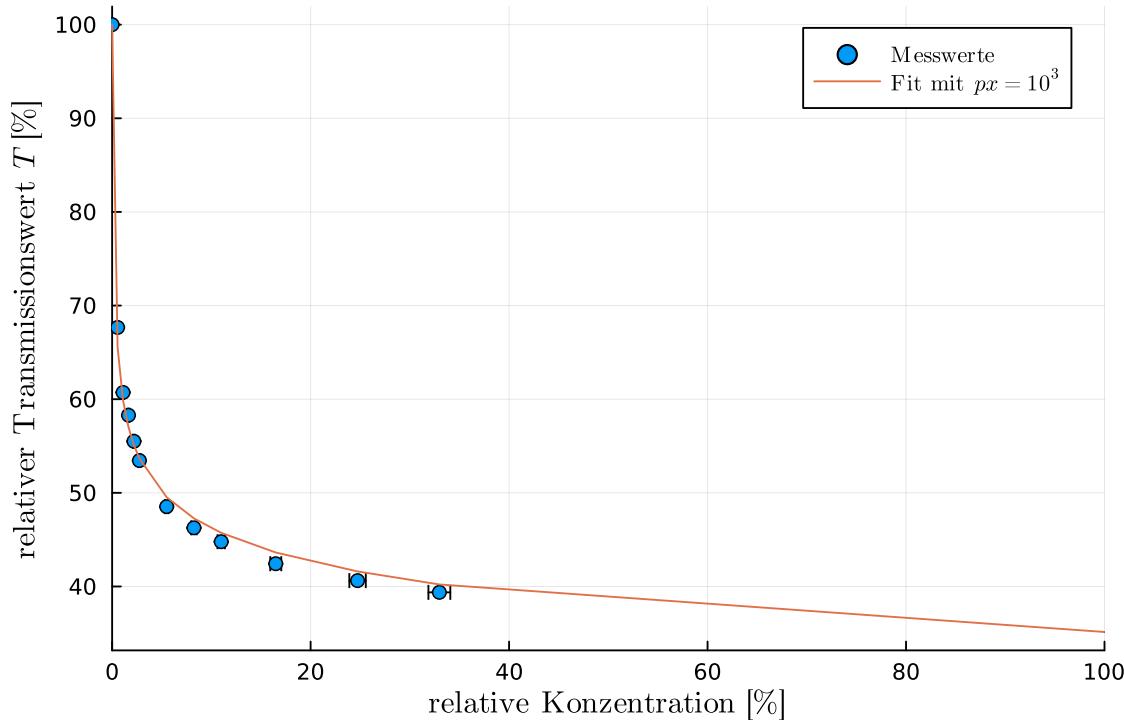
#, markershape=:diamond)  

xlabel!(L"\mathrm{relative \enspace Konzentration} \enspace [\%]")  

ylabel!(L"\mathrm{relativer \enspace Transmissionswert} \enspace T \enspace [\%]")

```

[17]:



```
savefig(tkFit, “..../media/B1.1/tkFit.pdf”);
```

Vergleich verschiedener Fitparameter px:

```
[18]: i = 1000
p1 = scatter(konzentrationen, transmissionswerte, label="Messwerte", □
    ↪xaxis=[0,1], title = "px = $i")
plot!(konzentrationen, t_rel(i)[3], label="px = $i, changed", markershape=:
    ↪diamond, markeralpha=0.5)

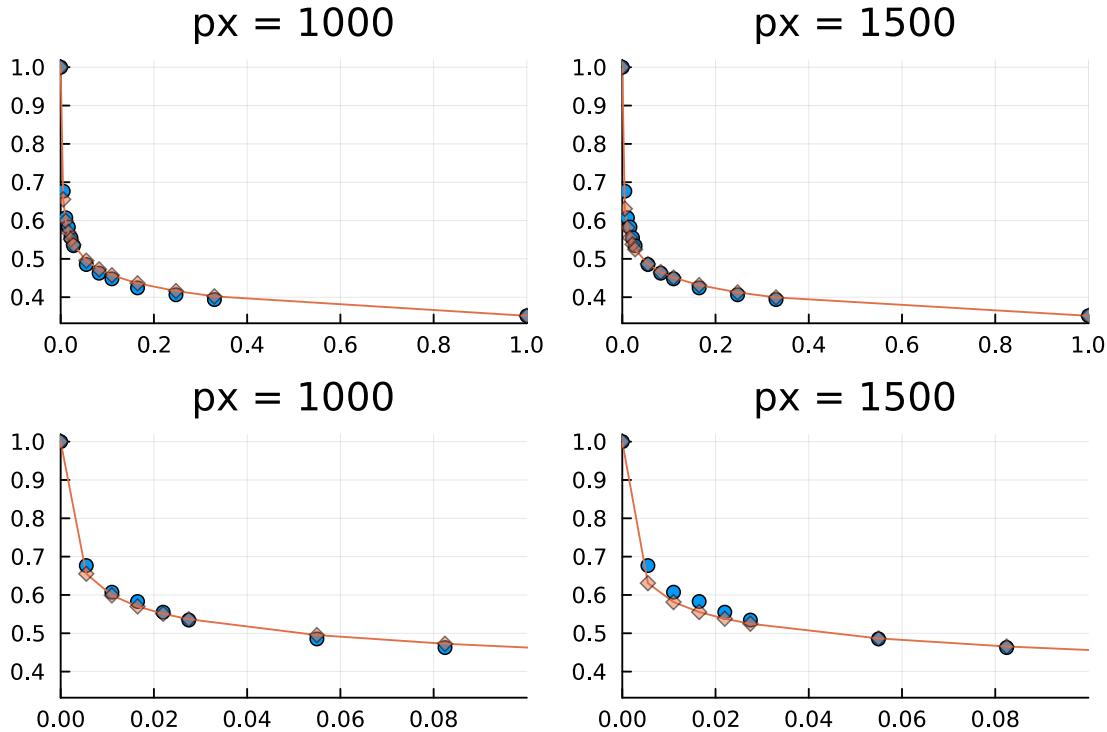
i = 1500
p2 = scatter(konzentrationen, transmissionswerte, label="Messwerte", □
    ↪xaxis=[0,1], title = "px = $i")
plot!(konzentrationen, t_rel(i)[3], label="px = $i, changed", markershape=:
    ↪diamond, markeralpha=0.5)

i = 1000
p3 = scatter(konzentrationen, transmissionswerte, label="Messwerte", xaxis=[0,0.
    ↪1], title = "px = $i")
plot!(konzentrationen, t_rel(i)[3], label="px = $i, changed", markershape=:
    ↪diamond, markeralpha=0.5)

i = 1500
p4 = scatter(konzentrationen, transmissionswerte, label="Messwerte", xaxis=[0,0.
    ↪1], title = "px = $i")
plot!(konzentrationen, t_rel(i)[3], label="px = $i, changed", markershape=:
    ↪diamond, markeralpha=0.5)

plot(p1, p2, p3, p4, layout=(2,2), legend=false)
```

```
[18]:
```



Minimale Konzentration mit Unsicherheit <= 10% (Frage 2):

```
[19]: I_0 = [0.91470, 0.91470, 0.90009, 0.88955, 0.88510, 0.88250, 0.87689, 0.86985, 0.86675, 0.86045, 0.85468, 0.84604, 0.80104]
ungenauigkeit = Array{Float64}(undef,length(I_0)-2)
for i in 2:length(I_0)-1 # Start bei 2 weil der erste keine richtige messung war
    ungenauigkeit[i-1] = mittelwert([I_0[i], I_0[i+1]])[2]
end
ΔI_therm = mittelwert(ungenauigkeit)[1]
I_0_mittel = mittelwert(I_0)[1]
T_min = 1 - 10 * ΔI_therm / I_0_mittel
```

[19]: 0.9408900567376727

```
[20]: K_min = 350 # ppm
K_min_err = 50 # ppm
```

[20]: 50

2. Versuch zum Fitten (Diesmal mit mehr Werten von C):

```
[21]: function t_rel_2(px_given)
    I_0 = 1 # ?
```

```

px = px_given
_0 = 0
Δ = 1 # =
C = 10 # (Testwert) 2. Parameter
I_t() = I_0 * exp(- px * exp(- 1 * (( - _0)/Δ)^2) * C) # statt 1 ist 0.5
↳ für die Form im Bild verwendet worden!

_1 = -10
_2 = 10
schrittweite = 0.01
C_schrittweite = 0.0001 # um zusätzliche "Messwerte" zu erzeugen
C_schrittweite_2 = 0.02
firstEnd = 0.02
C_werte = Array{Float64}(undef, Int(firstEnd/C_schrittweite + (1-firstEnd)/
C_schrittweite_2) + 1)

summen = Array{Float64}(undef, Int(firstEnd/C_schrittweite + (1-firstEnd)/
C_schrittweite_2) + 1)
summenIndex = 0
for c in 0:C_schrittweite:(firstEnd-C_schrittweite)
    C = c
    summenIndex += 1
    C_werte[summenIndex] = c
    for j in _1:schrittweite:_2
        summen[summenIndex] += I_t(j)
    end
end
for c_2 in firstEnd:C_schrittweite_2:1
    C = c_2
    summenIndex += 1
    C_werte[summenIndex] = c_2
    for j in _1:schrittweite:_2
        summen[summenIndex] += I_t(j)
    end
end
integrale = schrittweite .* summen

T_rel = (integrale .- integrale[length(integrale)]) ./ (integrale[1] -
integrale[length(integrale)])
offset = transmissionswerte[length(konzentrationen)]
faktor = 1-offset
T_rel_skaliert = faktor .* T_rel .+ offset
# integrale[length(konzentrationen)] = T(100%)
# integrale[1] = T(0%)
return integrale, T_rel, T_rel_skaliert, summen, C_werte
end

```

```
[21]: t_rel_2 (generic function with 1 method)
```

```
[22]: # Plot
#scatter(konzentrationen, transmissionswerte, label=L"Messwerte", xaxis=[0,0.
↪001], yaxis=[0.8,1],
#           xerr = konzentrationen_err, yerr = transmissionswerte_err)
i = 1000
t1, t2, y_array, t3, x_array = t_rel_2(i)
fitZoom = plot([1], label="") # Nur um Farben wie oben anzupassen
plot!(x_array.*10^6, y_array.*100, label=L"\mathrm{Fit} \enspace \mathrm{mit} \enspace
↪\enspace px = 10^3",
markeralpha=0.5, markershape=:diamond, xaxis=[0,1000], yaxis=[80,100])
xlabel!(L"\mathrm{Konzentration} \enspace [\mathrm{ppm}]")
ylabel!(L"\mathrm{Transmissionswert} \enspace T \enspace [\%]")

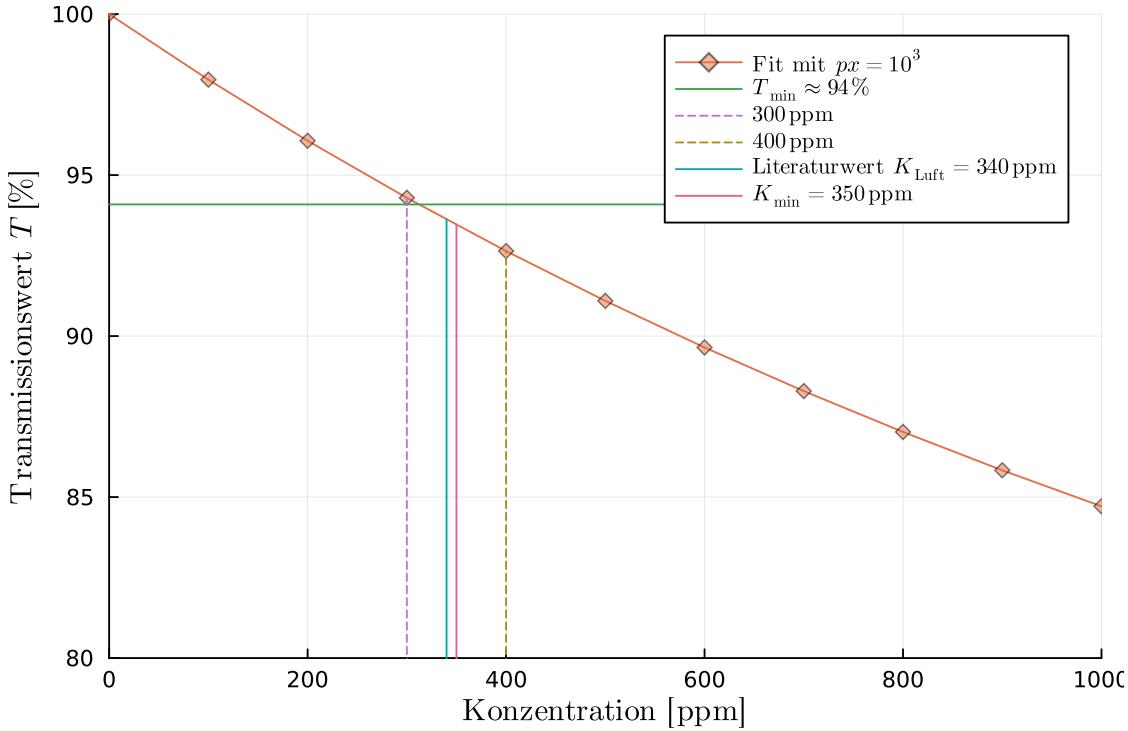
# Literaturwert:
lit = 340 # ppm

# Anliegende Punkte:
x1 = 300
x1_index = 4
x2 = 400
x2_index = 5

# Gerade zwischen erstem und zweitem anliegendem Punkt:
a = (y_array[x2_index] - y_array[x1_index]) / (x2 - x1)
litHeight = (lit - x1) * a + y_array[x1_index]
T_minLength = x1 + (T_min - y_array[x1_index]) / a
K_minHeight = (K_min - x1) * a + y_array[x1_index]

# Resultierende Plots:
plot!([0,2T_minLength], [T_min, T_min].*100, label=L"T_\mathrm{min} \approx
↪94\%, \%) # 2*Länge aus ästhetischen Gründen
plot!([300, 300], [0, y_array[4]].*100, linestyle = :dash, label=L"300 \enspace
↪\%, \mathrm{ppm}")
plot!([400, 400], [0, y_array[5]].*100, linestyle = :dash, label=L"400 \enspace
↪\%, \mathrm{ppm}")
plot!([lit, lit], [0,litHeight].*100, label=L"\mathrm{Literaturwert} \enspace \enspace
↪K_\mathrm{Luft} = 340 \%, \mathrm{ppm}")
plot!([K_min, K_min], [0,K_minHeight].*100, label=L"K_\mathrm{min} = 350 \enspace
↪\%, \mathrm{ppm}")
```

```
[22]:
```



```
savefig(fitZoom, "../media/B1.1/fitZoomPpm.pdf");
```

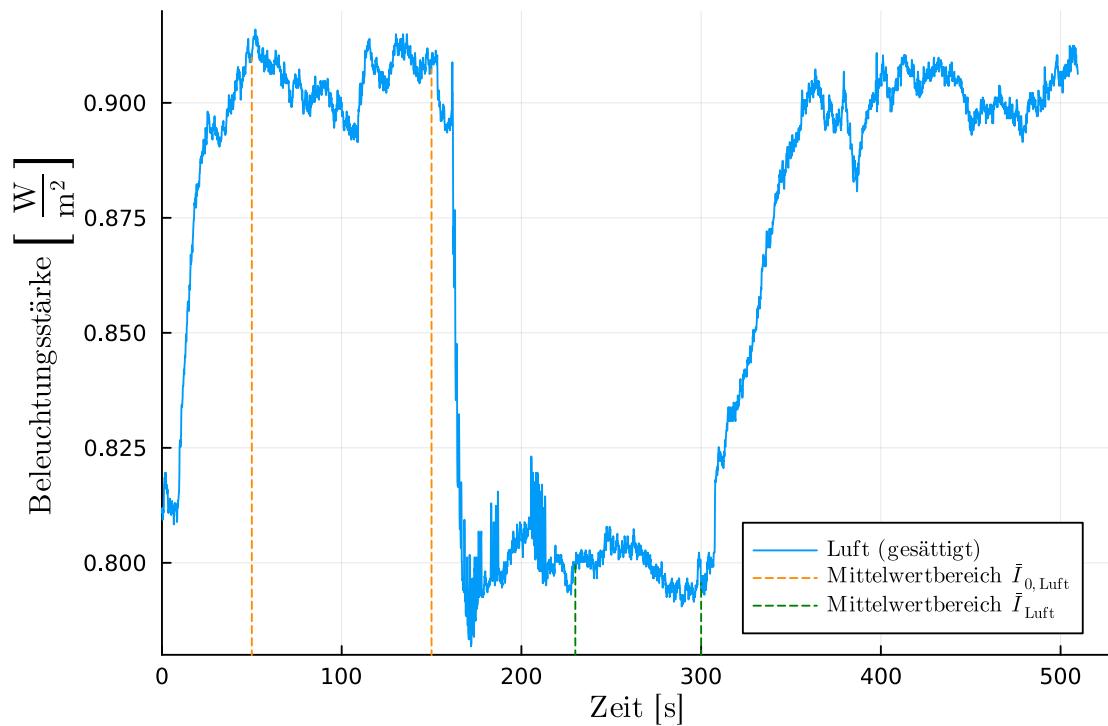
0.0.3 Auswertungsteil 4: Messung der CO₂-Konzentration in der Raumluft durch IR-Absorption

```
[23]: saettigungLuft = CSV.read("Konzentrationskalibrierung/Saettigung_Luft.csv", DataFrame);
```

```
[24]: luft = plot(saettigungLuft.seconds, saettigungLuft.beleuchtungsstaerke, label = L"\mathrm{Luft} \enspace (\mathrm{gesättigt})",
    legend=:bottomright, xaxis = [0,530], yaxis = [0.78,0.92])
plot!([50,50], [0.7, saettigungLuft[500, :beleuchtungsstaerke]], linestyle=:
    :dash, linecolor=:darkorange,
    label = L"\mathrm{Mittelwertbereich} \enspace \bar{I}_\mathrm{Luft}")
plot!([150,150], [0.7, saettigungLuft[1500, :beleuchtungsstaerke]], linestyle=:
    :dash, linecolor=:darkorange, label = "")
plot!([230,230], [0.7, saettigungLuft[2300, :beleuchtungsstaerke]], linestyle=:
    :dash, linecolor=:green,
    label = L"\mathrm{Mittelwertbereich} \enspace \bar{I}_\mathrm{Luft}")
plot!([300,300], [0.7, saettigungLuft[3000, :beleuchtungsstaerke]], linestyle=:
    :dash, linecolor=:green, label = "")
xlabel!(L"\mathrm{Zeit} \enspace [\mathrm{s}]")
```

```
ylabel!(L"\mathrm{Beleuchtungsstärke} \enspace \rightarrow \left[\mathrm{\frac{W}{m^2}}\right]")
```

[24]:



```
savefig(luft, "../media/B1.1/luft.pdf");
```

[25]: `I_0_luft = mittelwert(saettigungLuft[500:1500,:beleuchtungsstaerke])`

[25]: (0.9052179020978987, 0.00017177961799133386)

[26]: `I_luft = mittelwert(saettigungLuft[2300:3000,:beleuchtungsstaerke])`

[26]: (0.799490841654779, 0.00014846463455764117)

[27]: `T_luft = I_luft[1]/I_0_luft[1]`

[27]: 0.8832026408248328

[28]: `# Gaußsche Fehlerfortpflanzung:`
`T_luft_err = sqrt((I_luft[2] / I_0_luft[1])^2 + (I_luft[1] * I_0_luft[2] / I_0_luft[1]^2)^2)`

[28]: 0.00023449863741111332

```
[29]: t1, t2, trans, t3, conc = t_rel_2(10^3)
df = DataFrame(T=trans, C=conc)
;

[30]: # Plot
#scatter(konzentrationen, transmissionswerte, label=L"\"Messwerte\"", xaxis=[0,0.001], yaxis=[0.8,1],
#           xerr = konzentrationen_err, yerr = transmissionswerte_err)
i = 1000
t1, t2, y_array, t3, x_array = t_rel_2(i)
luftZoom = plot([1], label="")
# Nur um Farben wie oben anzupassen
plot!(x_array.*10^6, y_array.*100, label=L"\\"mathrm{Fit} \enspace \mathrm{mit}\"
\enspace px = 10^3",
markeralpha=0.5, markershape=:diamond, xaxis=[0,1000], yaxis=[80,100])
xlabel!(L"\\"mathrm{Konzentration} \enspace [\\"mathrm{ppm}]")
ylabel!(L"\\"mathrm{Transmissionswert} \enspace [\%]")

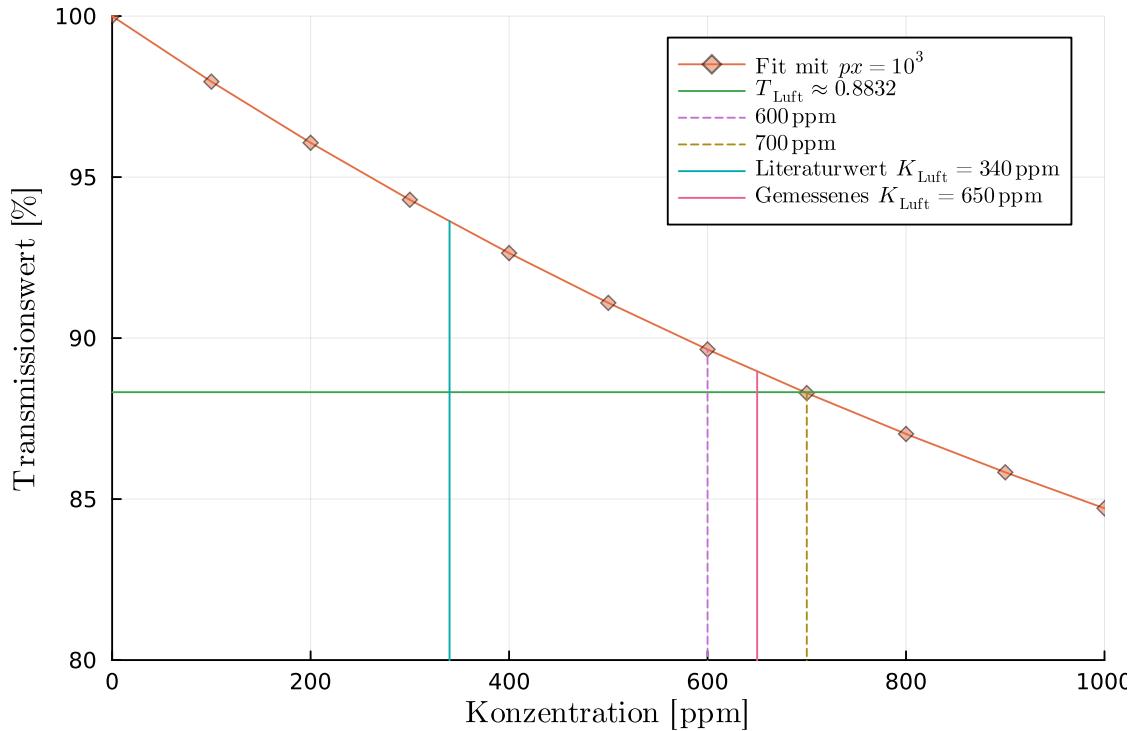
# Literaturwert:
lit = 340 # ppm

# Anliegende Punkte:
x1 = 600
x1_index = 7
x2 = 700
x2_index = 8

# Gerade zwischen erstem und zweitem anliegendem Punkt:
a_lit = (y_array[5] - y_array[4]) / (400 - 300)
litHeight = (lit - 300) * a_lit + y_array[4]
a = (y_array[x2_index] - y_array[x1_index]) / (x2 - x1)
T_luftLength = x1 + (T_luft - y_array[x1_index]) / a
luftHeight = (650 - x1) * a + y_array[x1_index]

# Resultierende Plots:
plot!([0, 2T_luftLength], [T_luft, T_luft].*100, label=L"\\"T_\mathrm{Luft}\"
\approx 0.8832" # 2*Länge aus ästhetischen Gründen
plot!([x1, x1], [0, y_array[x1_index]].*100, linestyle = :dash, label=L"600\"
\,\mathrm{ppm}")
plot!([x2, x2], [0, y_array[x2_index]].*100, linestyle = :dash, label=L"700\"
\,\mathrm{ppm}")
plot!([lit, lit], [0, litHeight].*100, label=L"\\"mathrm{Literaturwert} \enspace \enspace\"
\mathrm{Luft} = 340 \,\mathrm{ppm}")
plot!([650, 650], [0, luftHeight].*100, label = L"\\"mathrm{Gemessenes} \enspace \enspace\"
\mathrm{Luft} = 650 \,\mathrm{ppm}")
```

[30]:



```
savefig(luftZoom, "../media/B1.1/luftZoomPpm.pdf");
```

[31]: `T_luft_ppm = (700 + 600) / 2 # ppm (Literaturwert ~ 420 ppm bzw. 340 ppm)`

[31]: 650.0

[32]: `T_luft_ppm_err = (700 - 600) / 2 # Da dies genau die halbe Intervalllänge ist`

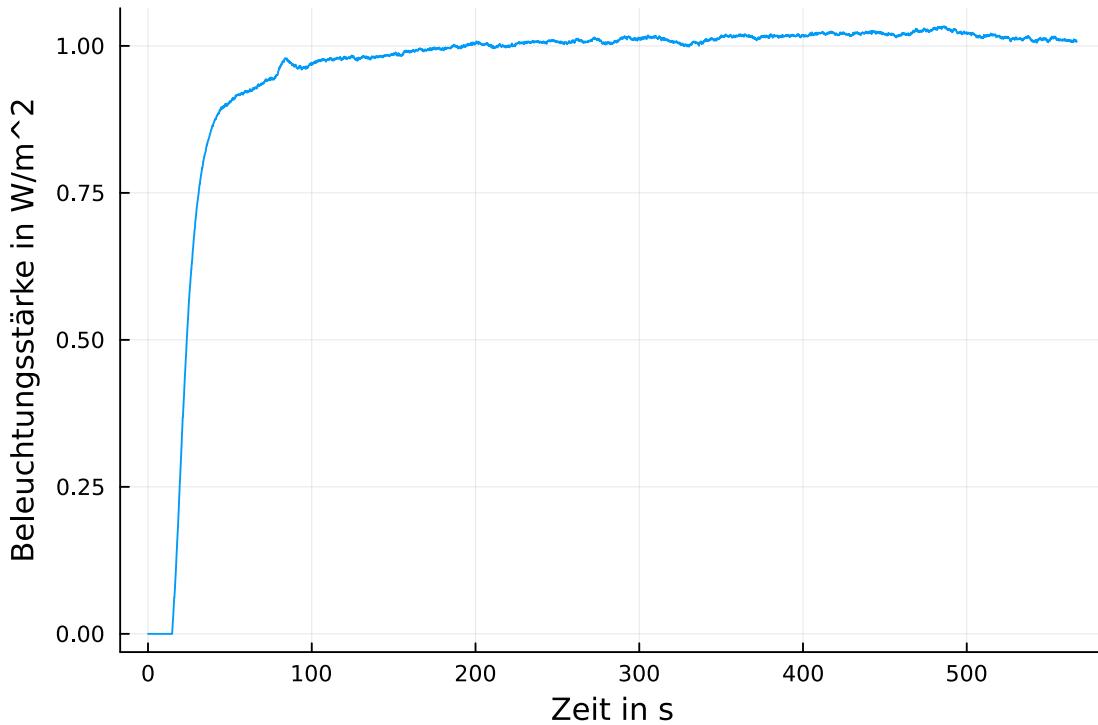
[32]: 50.0

0.0.4 Auswertungsteil 5: Driftkorrektur

[33]: `anschalt = CSV.read("IR-Transmission/Anschaltvorgang.csv", DataFrame);`

[34]: `plot(anschalt.seconds, anschalt.beleuchtungsstaerke, xlabel="Zeit in s",
ylabel="Beleuchtungsstärke in W/m^2", label="")`

[34]:



```
[35]: data = DataFrame(seconds = anschalt.seconds, beleuchtungsstaerke = anschalt.
    ↪beleuchtungsstaerke)

# Funktion zur Berechnung der Koeffizienten der linearen Anpassung
function linear_fit(x, y)
    X = [ones(length(x)) x] # Designmatrix
    = X \ y # Lineare Regression
    return
end

# Funktion zur Berechnung der Steigung in einem Intervall
function calculate_slope(data::DataFrame, start_idx::Int, end_idx::Int, x_col::Symbol, y_col::Symbol)
    # Daten im angegebenen Intervall extrahieren
    subdata = data[start_idx:end_idx, :]
    # Lineare Anpassung durchführen
    = linear_fit(subdata[:, x_col], subdata[:, y_col])
    intercept, slope = [1], [2]
    return slope
end

# Intervall definieren (ersetzen Sie dies durch Ihre tatsächlichen Intervalle)
start_idx = 1100
```

```

end_idx = 2000
# Steigung im Intervall berechnen
slope = calculate_slope(data, start_idx, end_idx, :seconds, :
    ↪beleuchtungsstaerke)
println("Die Steigung im Intervall $start_idx bis $end_idx ist $slope")

# Daten und angepasste Gerade im Intervall plotten
subdata = data[start_idx:end_idx, :]
= linear_fit(subdata[:, :seconds], subdata[:, :beleuchtungsstaerke])
intercept, slope = [1], [2]

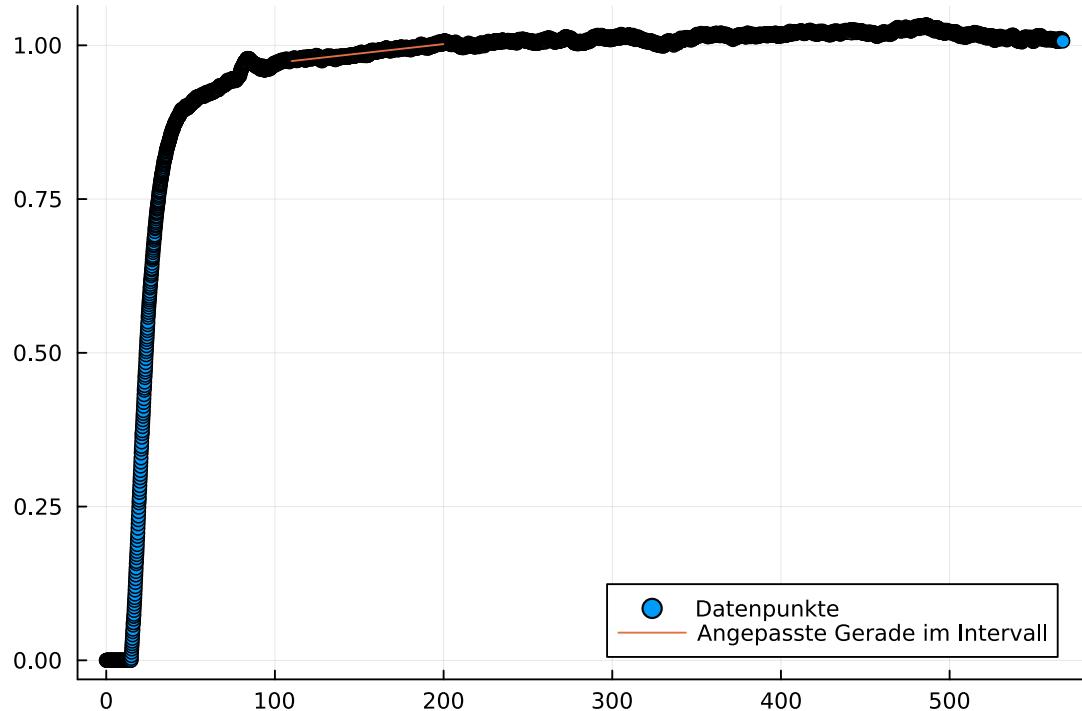
# Angepasstes Polynom definieren
p = x -> intercept + slope * x

# Plot
scatter(data.seconds, data.beleuchtungsstaerke, label="Datenpunkte")
plot!(x -> p(x), minimum(subdata.seconds):0.1:maximum(subdata.seconds), □
    ↪label="Anangepasste Gerade im Intervall")

```

Die Steigung im Intervall 1100 bis 2000 ist 0.0003061580554742107

[35]:



[36]: function g3(x)

```
if 0<=x<=310
```

```
return 1.179*10^(-4)*x
```

```

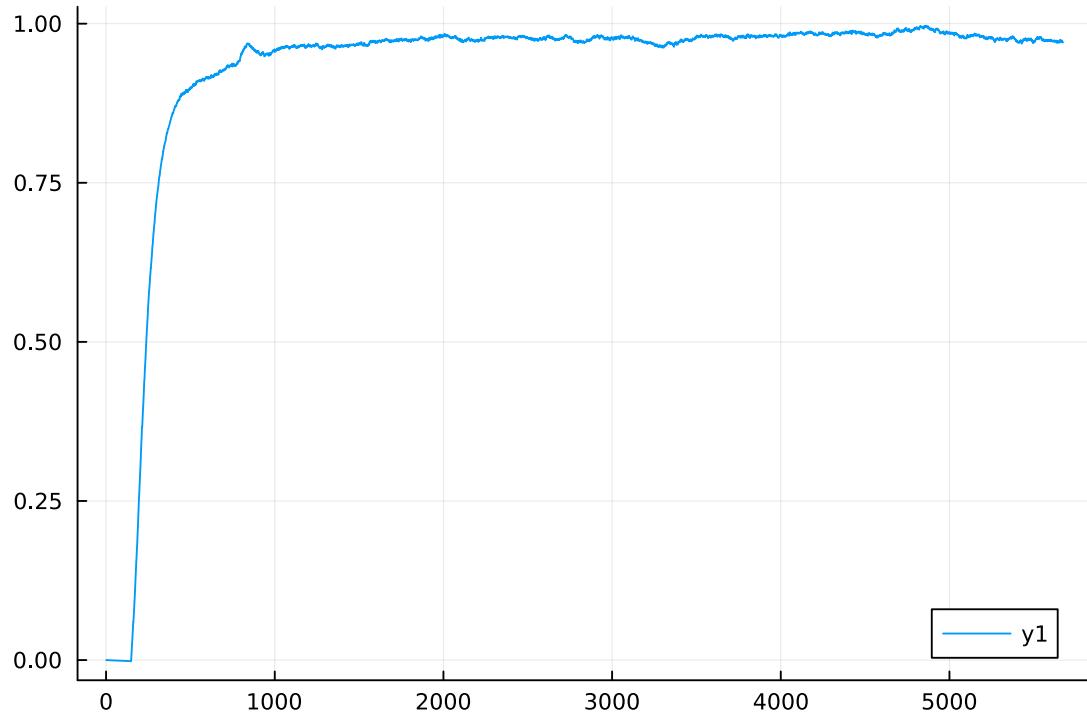
elseif x>310
    return 0.0365
else 0
end
end

```

[36]: g3 (generic function with 1 method)

[37]: # Berechne die angepassten Werte mit der Geradenfunktion g3
g_3 = g3.(anschalt.seconds)
drift3 = anschalt.beleuchtungsstaerke .- g_3
plot(drift3)

[37]:



[38]: data = DataFrame(seconds=anschalt.seconds, beleuchtungsstaerke=drift3)

```

function linear_fit(x, y)
    X = [ones(length(x)) x] # Designmatrix
    = X \ y # Lineare Regression
    return
end

# Funktion zur Berechnung der Steigung in einem Intervall

```

```

function calculate_slope(data, start_idx::Int, end_idx::Int, x_col::Symbol, □
    ↪y_col::Symbol)
    # Daten im angegebenen Intervall extrahieren
    subdata = data[start_idx:end_idx, :]
    # Lineare Anpassung durchführen
    = linear_fit(subdata[:, x_col], subdata[:, y_col])
    intercept, slope = [1], [2]
    return slope
end

# Intervall definieren (ersetzen Sie dies durch Ihre tatsächlichen Intervalle)
start_idx = 1100
end_idx = 2000
# Steigung im Intervall berechnen
slope = calculate_slope(data, start_idx, end_idx, :seconds, :
    ↪beleuchtungsstaerke)
println("Die Steigung im Intervall $start_idx bis $end_idx ist $slope")

# Daten und angepasste Gerade im Intervall plotten
subdata = data[start_idx:end_idx, :]
= linear_fit(subdata[:, :seconds], subdata[:, :beleuchtungsstaerke])
intercept, slope = [1], [2]

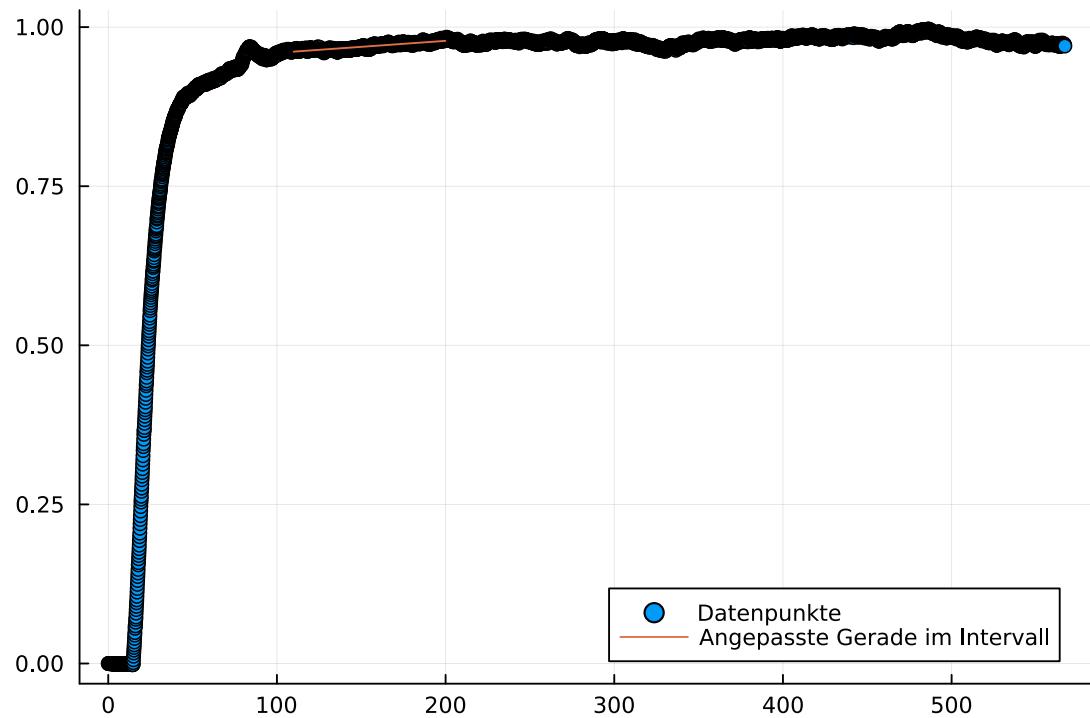
# Angepasstes Polynom definieren
p = x -> intercept + slope * x

# Plot
scatter(data.seconds, data.beleuchtungsstaerke, label="Datenpunkte")
plot!(x -> p(x), minimum(subdata.seconds):0.1:maximum(subdata.seconds), □
    ↪label="Angepasste Gerade im Intervall")

```

Die Steigung im Intervall 1100 bis 2000 ist 0.0001882580554742096

[38]:

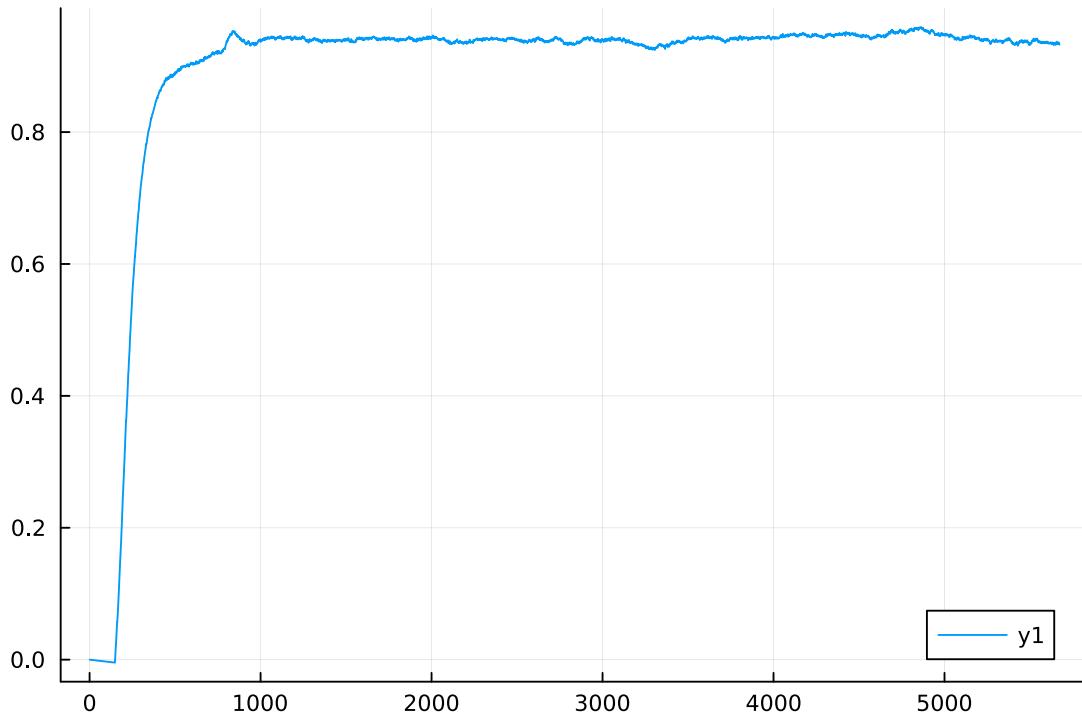


```
[39]: function g2(x)
    if 0<=x<=200
        return 1.8826*10^(-4)*x
    elseif x>200
        return 0.037652
    else 0
    end
end
```

```
[39]: g2 (generic function with 1 method)
```

```
[40]: g_2 = g2.(anschalt.seconds)
drift2 = drift3 .- g_2
plot(drift2)
```

```
[40]:
```



```
[41]: data = DataFrame(seconds=anschalt.seconds, beleuchtungsstaerke=drift2)

function linear_fit(x, y)
    X = [ones(length(x)) x] # Designmatrix
    = X \ y # Lineare Regression
    return
end

# Funktion zur Berechnung der Steigung in einem Intervall
function calculate_slope(data, start_idx::Int, end_idx::Int, x_col::Symbol, ▾y_col::Symbol)
    # Daten im angegebenen Intervall extrahieren
    subdata = data[start_idx:end_idx, :]
    # Lineare Anpassung durchführen
    = linear_fit(subdata[:, x_col], subdata[:, y_col])
    intercept, slope = [1], [2]
    return slope
end

# Intervall definieren (ersetzen Sie dies durch Ihre tatsächlichen Intervalle)
start_idx = 960
end_idx = 1100
# Steigung im Intervall berechnen
```

```

slope = calculate_slope(data, start_idx, end_idx, :seconds, :
    ↪beleuchtungsstaerke)
println("Die Steigung im Intervall $start_idx bis $end_idx ist $slope")

# Daten und angepasste Gerade im Intervall plotten
subdata = data[start_idx:end_idx, :]
= linear_fit(subdata[:, :seconds], subdata[:, :beleuchtungsstaerke])
intercept, slope = [1], [2]

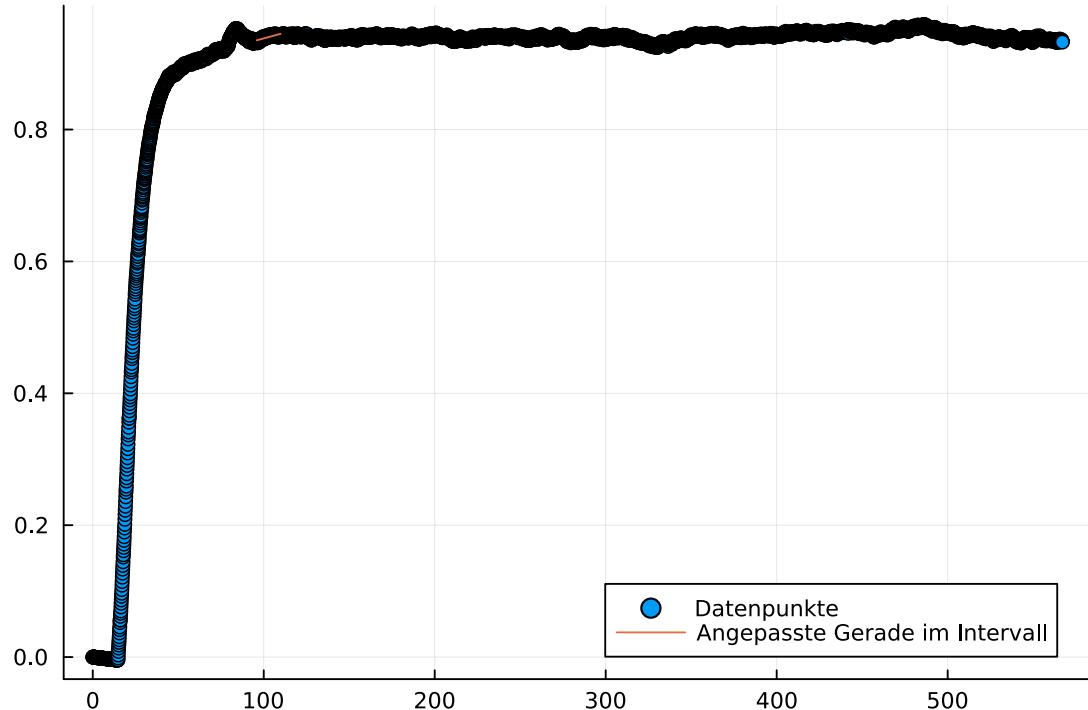
# Angepasstes Polynom definieren
p = x -> intercept + slope * x

# Plot
scatter(data.seconds, data.beleuchtungsstaerke, label="Datenpunkte")
plot!(x -> p(x), minimum(subdata.seconds):0.1:maximum(subdata.seconds), □
    ↪label="Anangepasste Gerade im Intervall")

```

Die Steigung im Intervall 960 bis 1100 ist 0.0007057617432253055

[41]:



[42]:

```

function g1(x)
    if 0<=x<=110
        return 7.0576*10^(-4)*x
    elseif x>110
        return 7.0576*10^(-4)*110

```

```

else 0
end
end

```

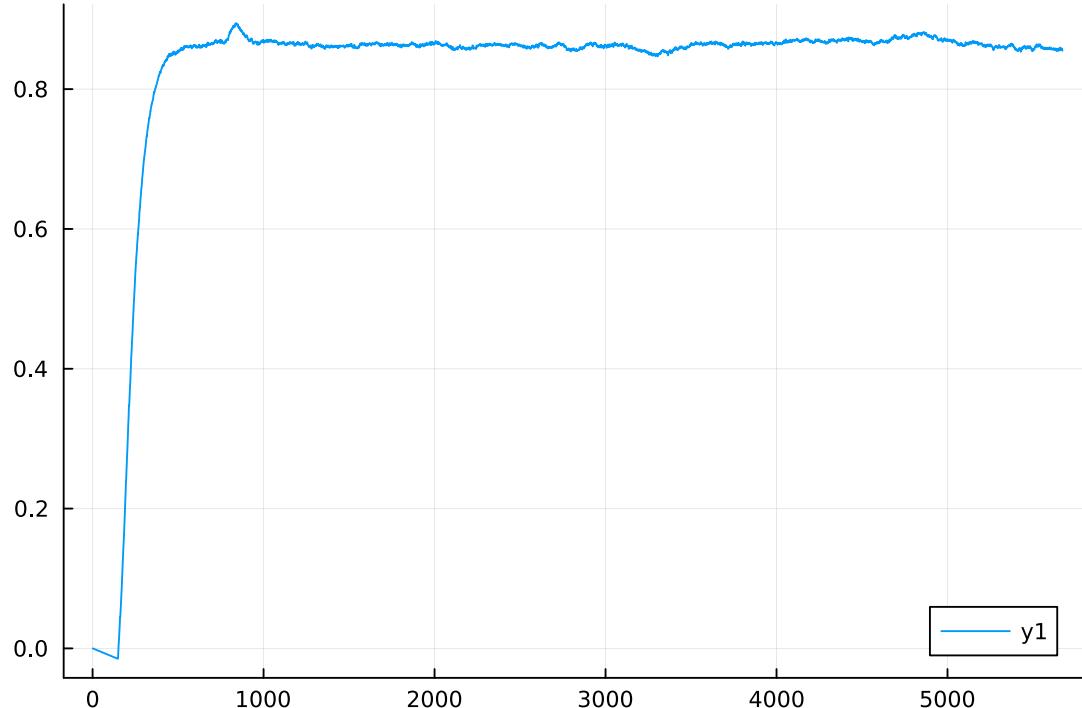
[42]: g1 (generic function with 1 method)

```

[43]: g_1 = g1.(anschalt.seconds)
drift1 = drift2 .- g_1
plot(drift1)

```

[43]:



[44]: b = minimum(drift1)

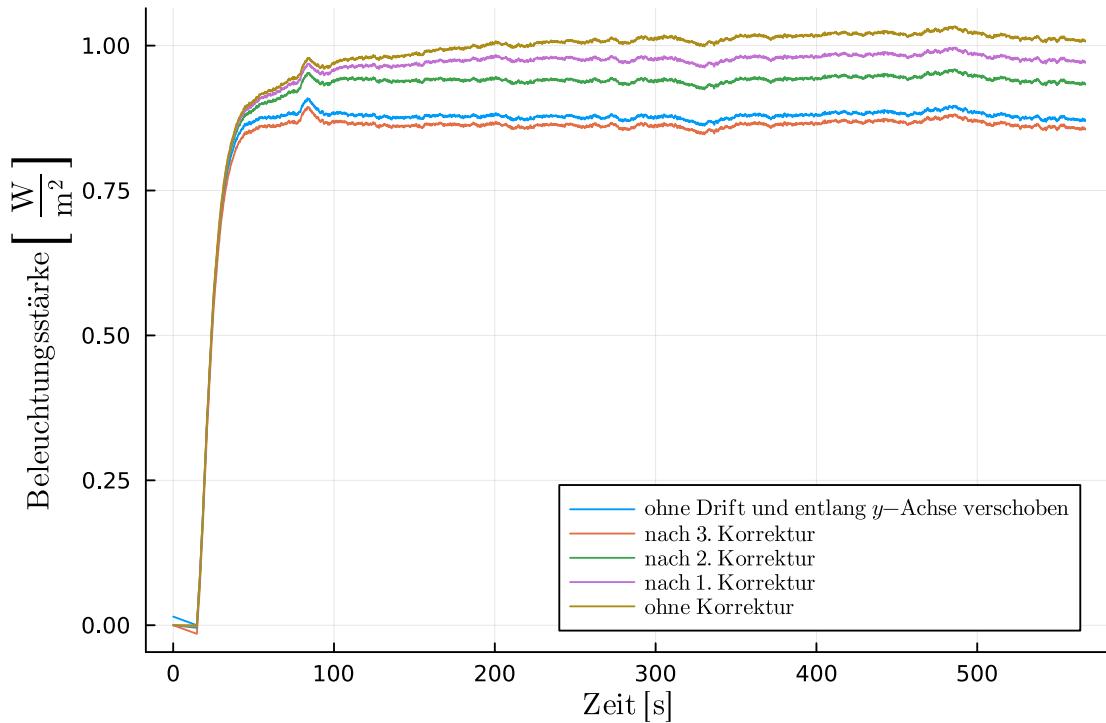
[44]: -0.014875224

```

[45]: ohne_drift = drift1 .- b
plot(anschalt.seconds,ohne_drift, xlabel=L"\mathrm{Zeit}\ [\mathrm{s}]",
      ylabel=L"\mathrm{Beleuchtungsstärke}\ \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]",
      label=L"\mathrm{ohne}\ Drift\ und\ entlang\ } y\mathrm{achse}\ verschoben")
plot!(anschalt.seconds,drift1, label=L"\mathrm{nach}\ } 3.\mathrm{ Korrektur}")
plot!(anschalt.seconds,drift2, label=L"\mathrm{nach}\ } 2.\mathrm{ Korrektur}")
plot!(anschalt.seconds,drift3, label=L"\mathrm{nach}\ } 1.\mathrm{ Korrektur}")
plot!(anschalt.seconds,anschalt.beleuchtungsstaerke, label=L"\mathrm{ohne}\ \hookrightarrow\mathrm{Korrektur}")

```

[45]:



```
savefig("../media/B1.1/Drift.pdf");
```

```
[46]: data = DataFrame(x=anschalt.seconds, y=anschalt.beleuchtungsstaerke);
```

```
[47]: start_sec = 155.5
end_sec = 155.5

filtered_data = filter(row -> start_sec <= row.x <= end_sec, data)
mean_value = mean(filtered_data.y)
println("Der Mittelwert der Beleuchtungsstärke im Bereich $start_sec bis
↪$end_sec Sekunden ist: $mean_value")
```

Der Mittelwert der Beleuchtungsstärke im Bereich 155.5 bis 155.5 Sekunden ist:
0.98685

```
[48]: start_sec =310
end_sec = 567.2
filtered_data = filter(row -> start_sec <= row.x <= end_sec, data)
mean_value1 = mean(filtered_data.y)
println("Der Mittelwert der Beleuchtungsstärke im Bereich $start_sec bis
↪$end_sec Sekunden ist: $mean_value1")
```

Der Mittelwert der Beleuchtungsstärke im Bereich 310 bis 567.2 Sekunden ist:
1.0166415746500779

```
[49]: mittel = mean_value/mean_value1
```

```
[49]: 0.9706960885793678
```

```
[50]: stationarer_wert = mean(ohne_drift[1000:5500])
```

```
[50]: 0.8789088146677182
```

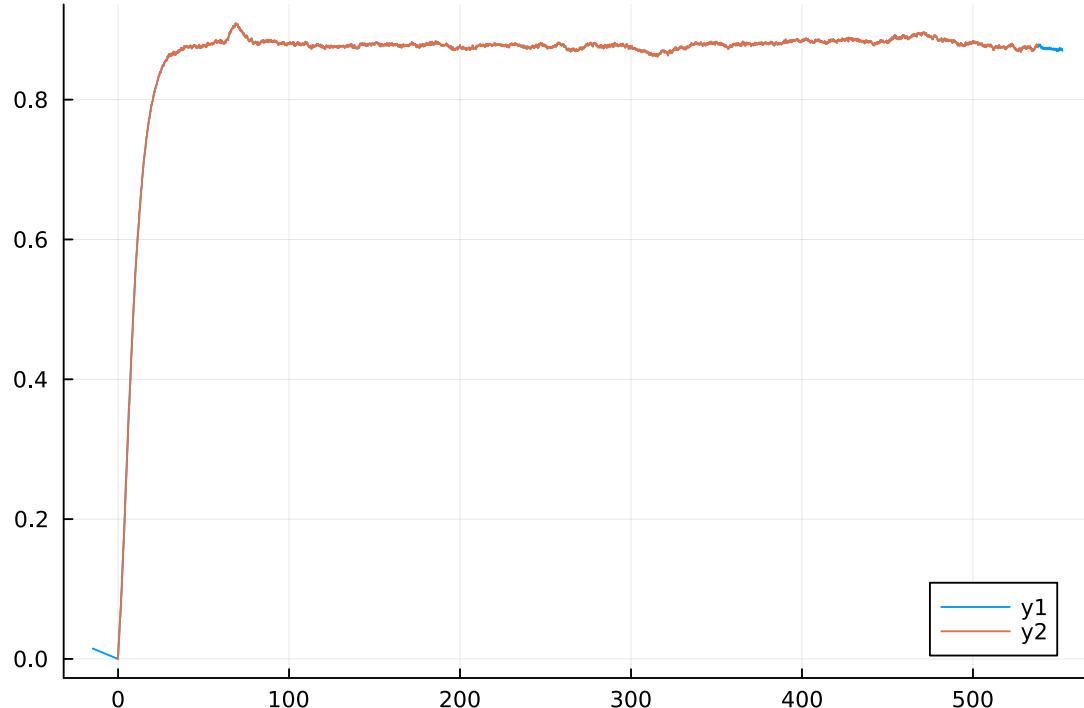
0.0.5 fitting

```
[51]: zeitverschiebung = 148 # 10th of second
zeitintervall_ende = length(ohne_drift) - zeitverschiebung

verschobene_zeit = anschalt.seconds[zeitverschiebung:zeitintervall_ende] .- zeitverschiebung/10
verschoben_ohne_drift = ohne_drift[zeitverschiebung:zeitintervall_ende]

plot(anschalt.seconds .- zeitverschiebung/10, ohne_drift)
plot!(verschobene_zeit, verschoben_ohne_drift)
```

```
[51]:
```



```
[52]: rounded_string(value) = rpad(round(value, digits=3), length(string(round(value))), "0")
```

[52]: rounded_string (generic function with 1 method)

curve_fit benötigt (mindestens) zwei Parameter zum Fitten. Da nur einer benötigt wird, wir dieser als Summe beider notwendigen Parameter dargestellt.

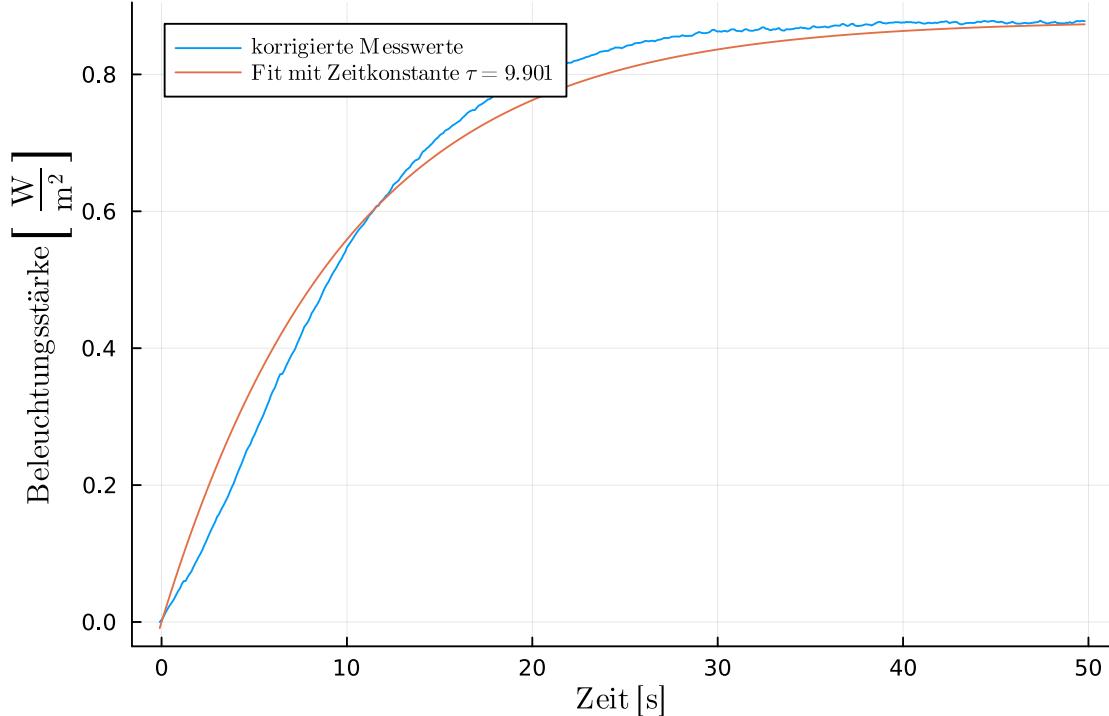
```
[53]: fit_range = 1:500
fit_func(x, p) = stationarer_wert .* (1 .- exp.(-x./(p[1]+p[2])))

result = curve_fit(fit_func, verschobene_zeit[fit_range], ↴
    ↪verschoben_ohne_drift[fit_range], [0.8, 8]);
zeitkonstante = rounded_string(sum(result.param))
@show zeitkonstante

plot(verschobene_zeit[fit_range], verschoben_ohne_drift[fit_range], ↴
    ↪label=L"\mathrm{korrigierte\ Messwerte}",
    xlabel=L"\mathrm{Zeit}\ [\mathrm{s}]",
    ↪ylabel=L"\mathrm{Beleuchtungsstärke}\ \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]")
plot!(verschobene_zeit[fit_range], fit_func(verschobene_zeit[fit_range], result. ↪
    ↪param),
    label="\$\\mathrm{Fit}\\ mit\\ Zeitkonstante\\ }\\tau=$zeitkonstante\$")
```

zeitkonstante = "9.901"

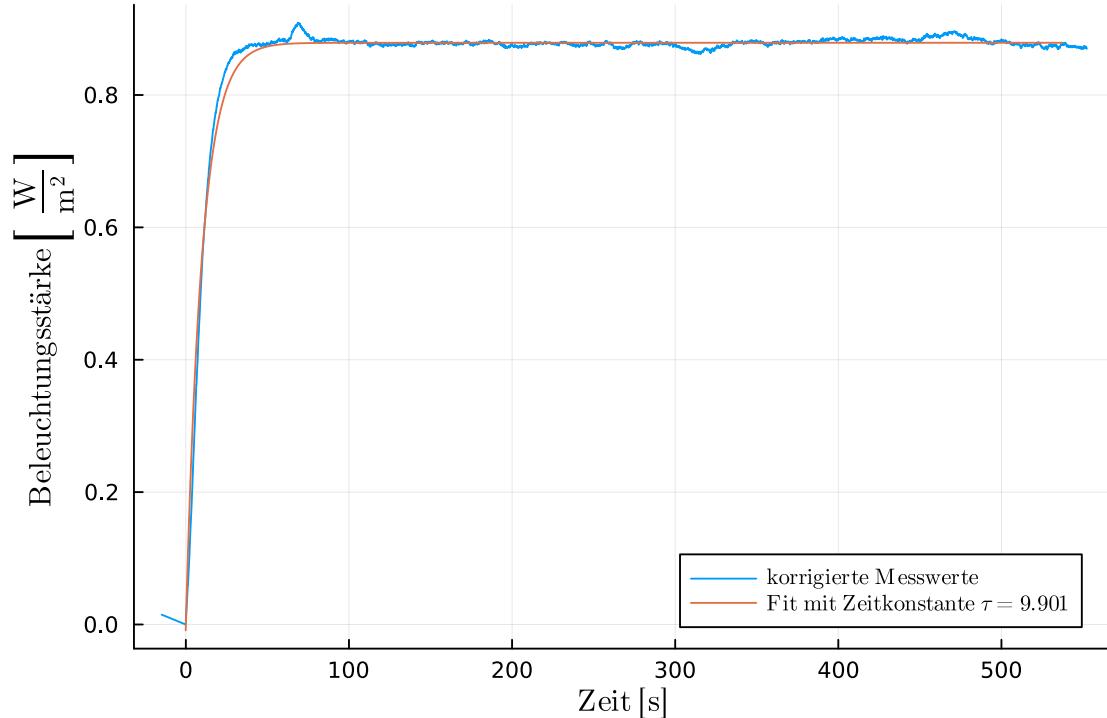
[53]:



```
savefig("../media/B1.1/Zeitkonstante-zoomed.pdf");
```

```
[54]: plot(anschalt.seconds .- zeitverschiebung/10, ohne_drift,
    ↪label=L"\mathrm{korrigierte\ Messwerte}",
    xlabel=L"\mathrm{Zeit}\ [\mathrm{s}]",
    ↪ylabel=L"\mathrm{Beleuchtungsstärke}\ \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]")
plot!(verschobene_zeit, fit_func(verschobene_zeit, result.param),
    label="\$\\mathrm{Fit}\\ mit\\ Zeitkonstante\\ \tau=\$zeitkonstante\$")
```

[54]:



```
savefig("../media/B1.1/Zeitkonstante.pdf");
```

0.0.6 Mittelwerte

```
[55]: start_sec = 100
end_sec = 100
filtered_data = filter(row -> start_sec <= row.x <= end_sec, data)
mean_value = mean(filtered_data.y)
println("Der Mittelwert der Beleuchtungsstärke im Bereich $start_sec bis
↪$end_sec Sekunden ist: $mean_value")
```

Der Mittelwert der Beleuchtungsstärke im Bereich 100 bis 100 Sekunden ist:
0.97002

```
[56]: mean_value/0.8797
```

```
[56]: 1.1026713652381492
```

```
[57]: # Beispiel-Daten
t_data = verschobene_zeit # Zeit-Daten
y_data = verschoben_ohne_drift # Messwerte

# Modellfunktion
model(t, a) = 0.8797 .* (1 .- exp.(-t ./ a))

# Anfangsschätzung für Parameter a
initial_guess = [3.0]

# Kurvenanpassung
kurve = curve_fit(model, t_data, y_data, initial_guess)

# Ergebnisse ausgeben
println("Angepasster Parameter a: ", kurve.param[1])
```

Angepasster Parameter a: 9.91289040187024

```
[58]: # Kovarianzmatrix
covariance_matrix = estimate_covar(kurve)

# Fehler für a ist die Quadratwurzel des entsprechenden Diagonalelements der ↴Kovarianzmatrix
standard_errors = sqrt.(diag(covariance_matrix))

# Fehler für a
error_a = standard_errors[1]

# Ergebnisse ausgeben
println("Fehler für a: ", error_a)
```

Fehler für a: 0.02594125381611405