👥 Group C:

Manoranjani Navamani Kathiresan Kulanthaivel

Brice Setra Robert

Assignment 2

1. Use a moving average filer over 100 samples to extract the large scale fading from the power measurements (Hint: have a look at the function smooth). Substract the large scale fading from the measurements to get the small scale fading. Plot the measurements as well as the two extracted fading processes vs time in the same figure

In [1]:
```
## Receive the signal
include("./data/measurements.jl");
```

In [2]:
```
# Vectorize the Matlab's data to Julia's format
println("A vector in Matlab is represented as a ",typeof(rx_power_dBm), " with size ", size(rx_power_dBm), " pres
Pᵣₓ = vec(rx_power_dBm)
println("Julia uses column vectors for vector representation, vec(.) function transforms row to column vector ",
```

```
A vector in Matlab is represented as a Matrix{Float64} with size (1, 10000) presented as a row vector to Julia
Julia uses column vectors for vector representation, vec(.) function transforms row to column vector Vector{Float6
4}
```

In [3]:
```
using Plots, Statistics, FFTW, Measures, LinearAlgebra
```

In [4]:
```
# The benefit of using Julia is its usage of Unicode characters
# allowing a more readable source code close to its Mathematical notation
# the below Julia source code brings some practical operations
# including the 𝔼 mean and inverse FFT that will be used throughout the Notebook
include("../modules/operations.jl");
```

In [5]:
```
N = 100 # Windows Size
```

Out[5]: 100

In signal processing, a moving average filter is used to smooth out variations in a signal by averaging neighboring data points. It's commonly used for noise reduction and to highlight underlying trends in the data.

The formula:

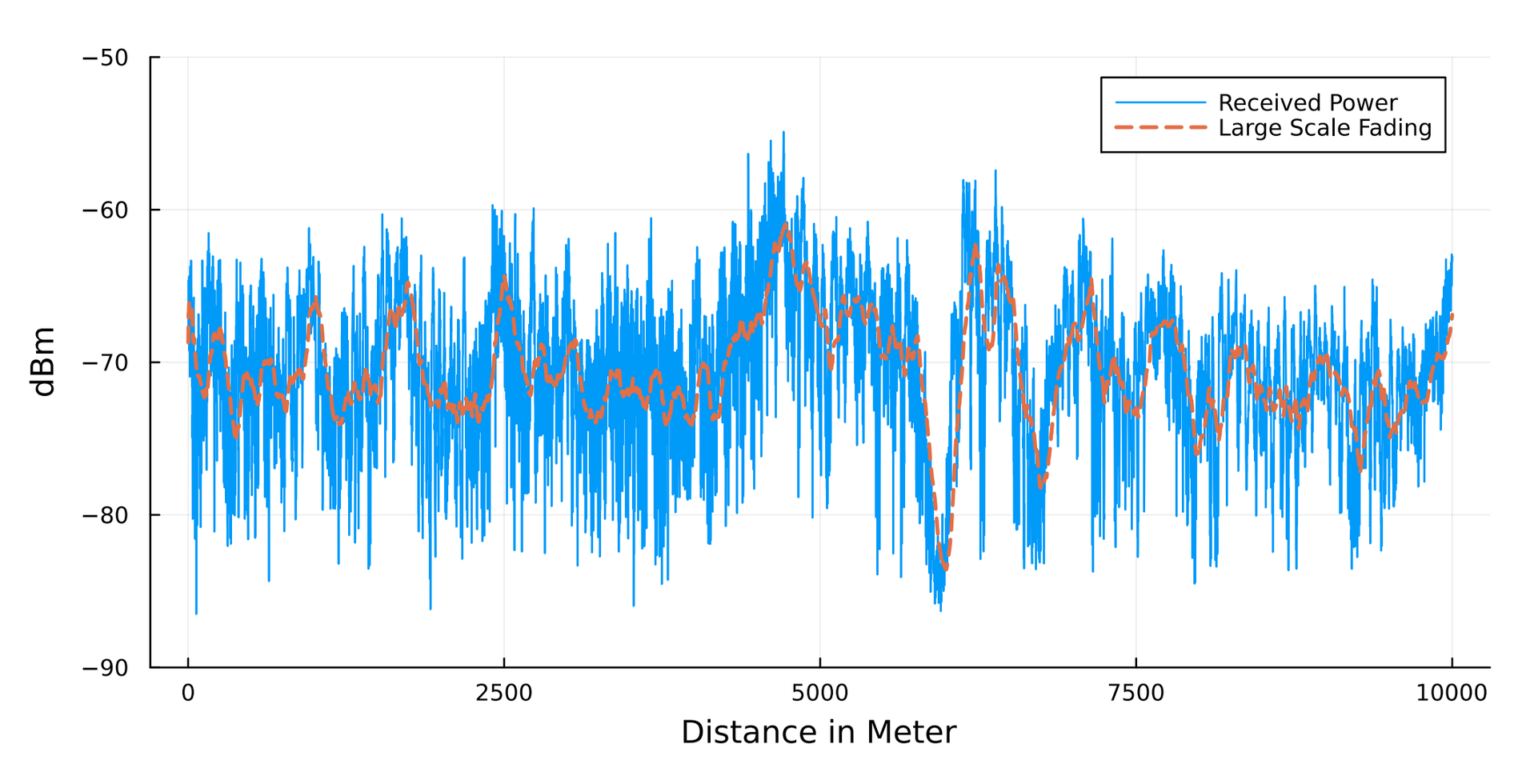
$$F[i] = \mathrm{E}\left(\mathrm{rx}[\max(1, i - N + 1) : i]\right)$$

captures the essence of a moving average filter operation. It computes the mean (average) of a sliding window of data points from the input signal rx, where the window size is defined by N. This mean value represents the output of the moving average filter at each index i, providing a smoothed version of the original signal.

In [6]:
```
# Moving Average Filter
F = [ 𝔼( Pᵣₓ[ max(1, i-N+1): i ] ) for i in 1:length(Pᵣₓ) ];
```

In [7]:
```
# Plotting
plot(Pᵣₓ
    , label="Received Power"
    , xlabel="Distance in Meter", ylabel="dBm"
    , linewidth=1
    , size = (800,400)
    , ylims = (-90, -50)
)
plot!(F
    , label="Large Scale Fading"
    , linewidth=2
    , linestyle=:dash
    , margin = 5mm
)
```

Out[7]:



Definitions:

1. **Received Power Signal** ($P_{rx}(t)$): This represents the power of the received signal at time t.
2. **Large Scale Fading Component** ($\mathcal{F}(t)$): This represents the average or smoothed power of the received signal, typically obtained by applying a moving average filter to $P_{rx}(t)$. It accounts for the slow variations due to path loss and shadowing.
3. **Small Scale Fading Component** ($Y_2(t)$): This represents the rapid fluctuations around the average signal power, primarily caused by multipath propagation effects.

Mathematical Representation:

Given:

- $P_{rx}(t)$: The received power signal at time t.
- $\mathcal{F}(t)$: The large scale fading component at time t.

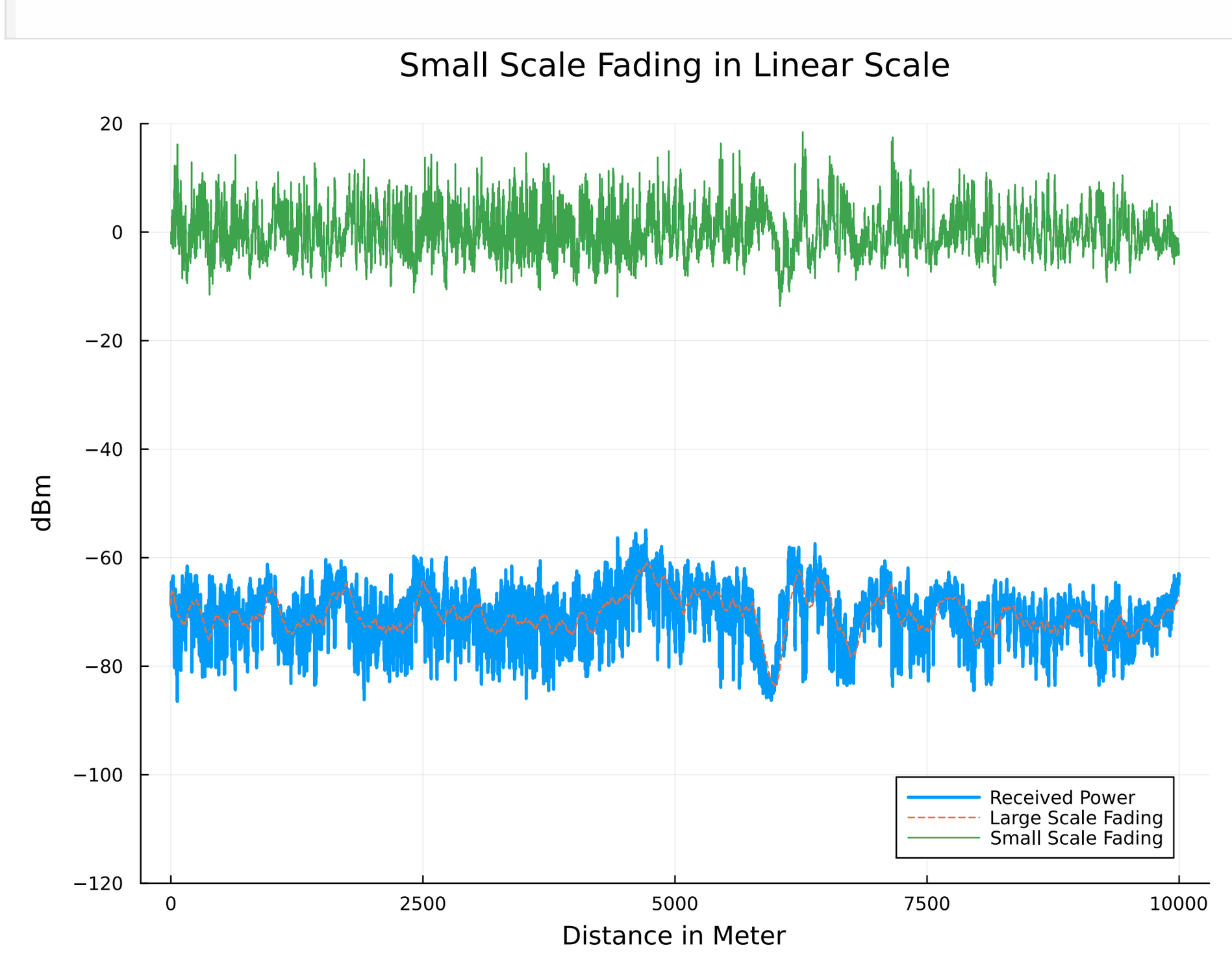
The small scale fading component $Y_2(t)$ is given by:

$$Y_2(t) = P_{rx}(t) - \mathcal{F}(t)$$

In [8]:
```
# Generate the Y2 data
Y₂ = F .- Pᵣₓ ;
```

In [9]:
```
using Plots

# Plot Received Power
plot(Pᵣₓ
    , title="Small Scale Fading in Linear Scale"
    , label="Received Power"
    , xlabel="Distance in Meter", ylabel="dBm"
    , linewidth=2, size=(800,600)
    , ylims = (-120,20)
    ,legend=:true, margin=5mm
)

# Add Large Scale Fading to the same plot
plot!(F, label="Large Scale Fading", linestyle=:dash)

# Plot Small Scale Fading in a new plot
plot!(Y₂, label="Small Scale Fading" )
```

Out[9]:

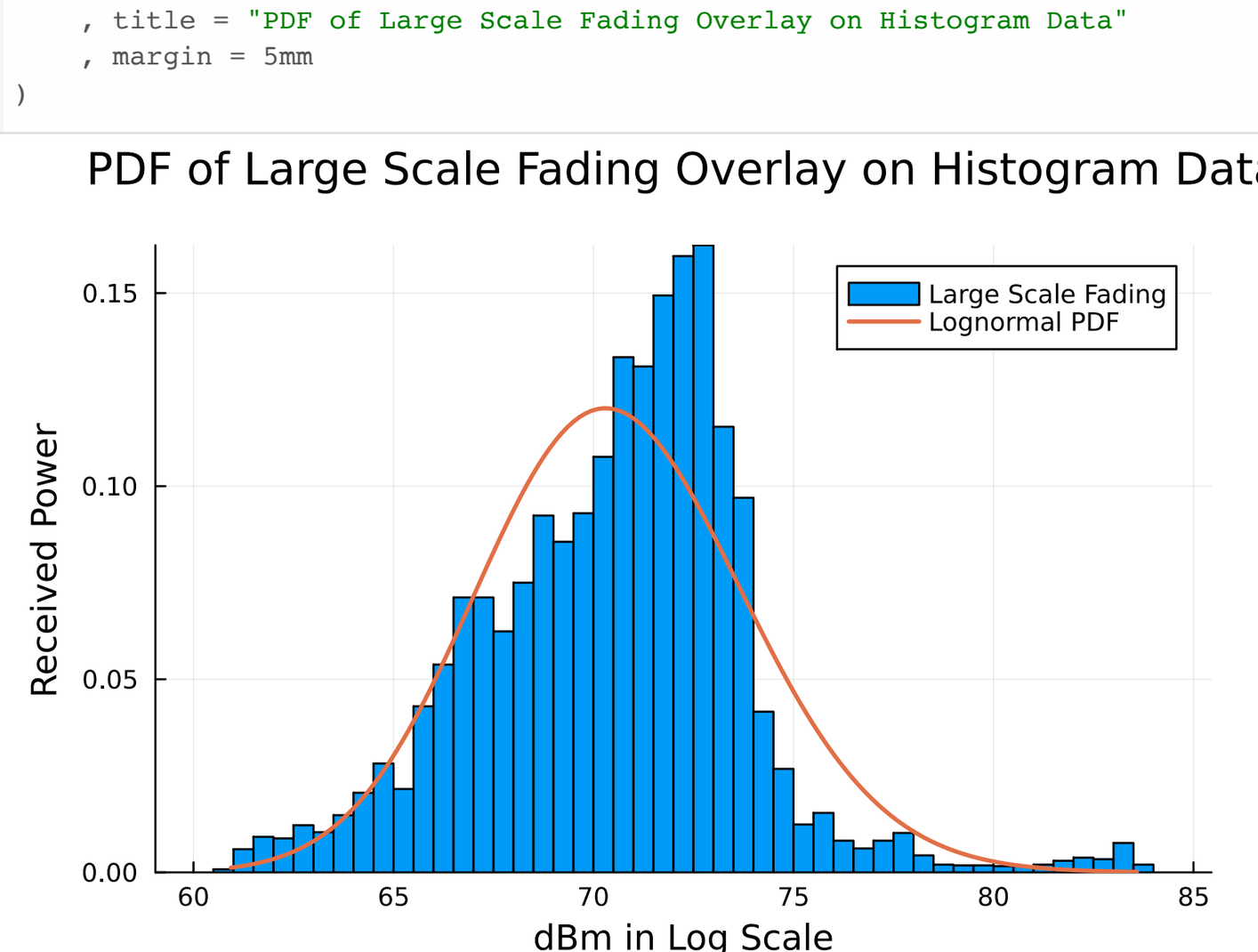


2. Plot a histogram of the large scale fading (on a log scale). Compute the mean and the variance and overlay the histogram with the pdf of a lognormal distribution of those parameters (Hint: have a look at function histfit). Discuss the result.

In [10]:
```
using Plots
using StatsBase
using Distributions
using Measures
```

In [11]:
```
# Create histogram
histogram(-F
    , bins= 100
    , xticks=:log10
    # , xscale=:ln
    , normed=true
    , label="Large Scale Fading"
   # , xticks = :false
)

# Fit lognormal distribution to data
 fit_lognormal = fit(LogNormal, -F)

# Overlay histogram with PDF of lognormal distribution
 X = exp.(range(log(minimum(-F)), log(maximum(-F)), length=150))
pdf_lognormal = pdf(fit_lognormal, X)
plot!(X, pdf_lognormal
    , linewidth=2
    , xticks=:log10
    , label="Lognormal PDF"
    , xlabel ="dBm in Log Scale"
    , ylabel = "Received Power"
    , title = "PDF of Large Scale Fading Overlay on Histogram Data"
    , margin = 5mm
)
```

Out[11]:



The plot shows the difference between the log-normal distribution's pdf and the histogram of large-scale fading. The log-normal pdf was generated using the fit function with mean and variance calculated from the large-scale fading data.

3. Plot a histogram of the small scale fading and and overlay the histogram with the pdf of a Rayleigh, a Ricean and a Nakagami distribution (Hint: have a look at function histfit). Discuss the results.

In [12]:
```
include("distributions/Nakagami.jl")
```

Out[12]: fit_mle (generic function with 62 methods)

In [13]:
```
using Plots
using Distributions
using SpecialFunctions  # Add this to access the gamma function
```

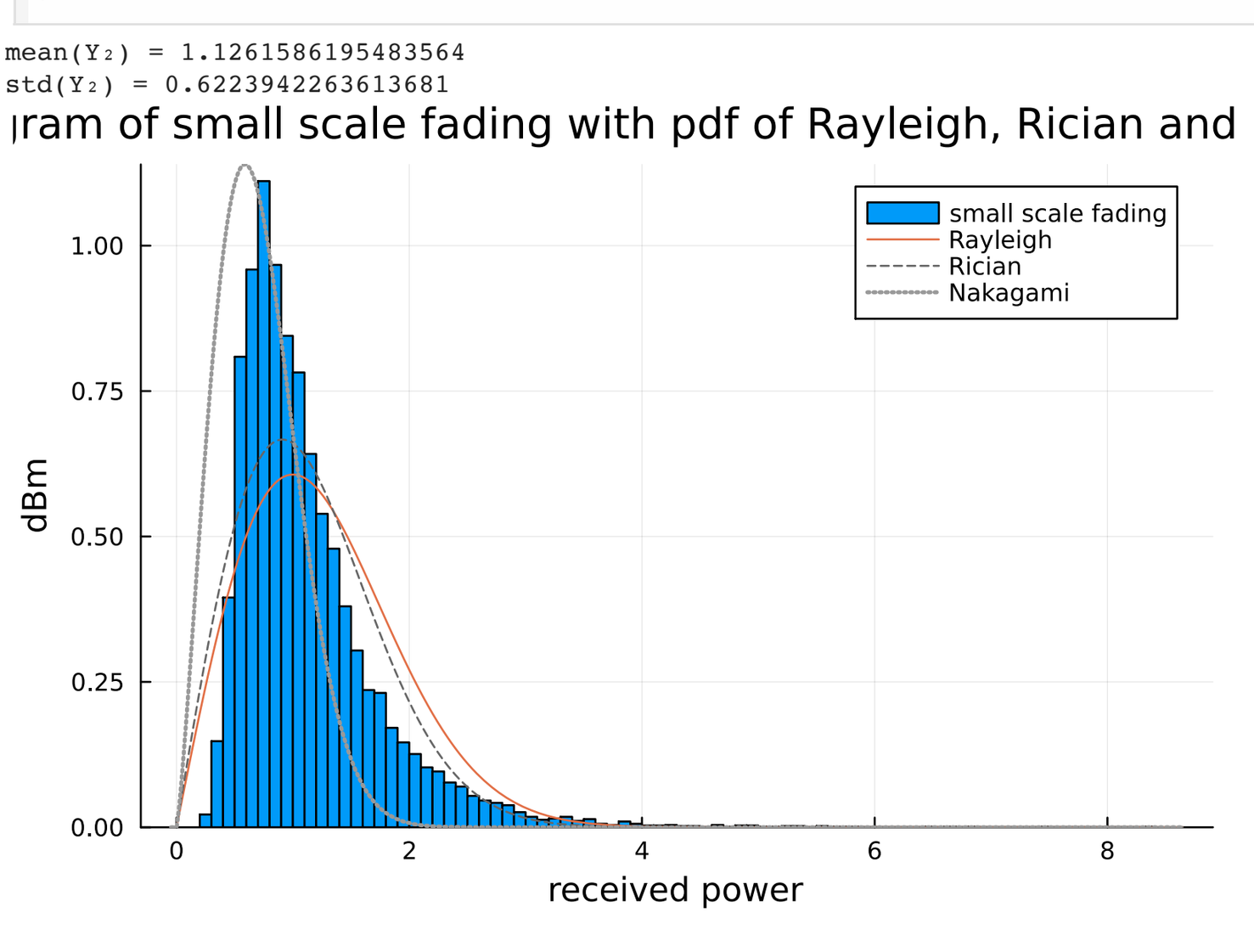
In [14]:
```
# Generate the Y2 data
Y₂ = F .- Pᵣₓ

# Step 1: Vectorize Y₂ (ensure it's a column vector)
Y₂ = vec(Y₂); @show typeof(Y₂), sizeof(Y₂)

# Step 2: Convert Y₂ from dB to linear scale
# Convert to linear and to amplitude
Y₂ = 10 .^(Y₂ / 20);
```

```
(typeof(Y₂), sizeof(Y₂)) = (Vector{Float64}, 80000)
```

In [15]:
```
# Plot the histogram
pₕ = histogram(Y₂
    , bins=100
    , normalize=true
    , label="small scale fading"
    , xlabel="received power", ylabel="dBm"
    , title="histogram of small scale fading with pdf of Rayleigh, Rician and Nakagami"
)


# Fit Rayleigh distribution
σ = 1.0
rayleigh_dist = Rayleigh(σ)
rayleigh_pdf = x -> pdf(rayleigh_dist, x)
pₗ = plot!(rayleigh_pdf
    , label="Rayleigh"
)

# Fit Rician distribution (manual fitting)
μᵣᵢ꜀ᵢₐₙ = mean(Y₂)
σᵣᵢ꜀ᵢₐₙ = std(Y₂)
rician_dist = Rician(μᵣᵢ꜀ᵢₐₙ, σᵣᵢ꜀ᵢₐₙ)
rician_fit = fit(Rician,Y₂)

rician_pdf = x -> pdf(rician_fit, x)
pᵣ = plot!(rician_pdf
    , label="Rician"
    , color=RGB(0.4, 0.4, 0.4)
    , linestyle=:dash
)

# Fit Nakagami distribution
# Julia does not have a built-in Nakagami distribution, so we simulate it
μₙₐₖₐᵧₐₘᵢ = mean(Y₂); @show mean(Y₂); # Shape parameter
ωₙₐₖₐᵧₐₘᵢ = std(Y₂); @show std(Y₂); # Spread parameter
nakagami_dist = Nakagami(μₙₐₖₐᵧₐₘᵢ, ωₙₐₖₐᵧₐₘᵢ)
nakagami_pdf = x -> pdf(nakagami_dist, x)
pₙ = plot!(nakagami_pdf
    , label="Nakagami"
    , color=RGB(0.6, 0.6, 0.6)
    , linestyle=:dot, linewidth=2.0
)
```

```
mean(Y₂) = 1.1261586195483564
std(Y₂) = 0.6223942263613681
```

Out[15]:



4. Using the original time-variant transfer function H2, calculate and plot the power delay profile making sure you label the axes correctly. Further compute the total power, the average mean delay, the average rms delay spread, and the coherence bandwidth. Explain how you calculated each value and discuss the results.

The Power Delay Profile (PDP) is a representation of the power of a signal as a function of time delay, often used in wireless communications to characterize multipath propagation. It describes how the power of the received signal is distributed over different time delays.

The mathematical symbol for the Power Delay Profile can vary depending on the context, but it is commonly denoted as $P(\tau)$, where τ represents the delay time.

Mathematical Definition:

If $h(t)$ represents the channel impulse response, the Power Delay Profile $P(\tau)$ is defined as:

$$P(\tau) = \mathbb{E}[|h(\tau)|^2]$$

Here, $\mathbb{E}$ denotes the expectation operator, and $|h(\tau)|^2$ represents the power of the channel impulse response at delay τ.

Example in Mathematical Expression:

Consider a channel impulse response $h(t, \tau)$ where t is the time and τ is the delay. The Power Delay Profile can be expressed as:

$$P(\tau) = \mathbb{E}[|h(t, \tau)|^2]$$

In discrete form, if $h[n]$ represents the sampled impulse response, the PDP can be written as:

$$P[n] = \frac{1}{N} \sum_{k=1}^{N} |h[k, n]|^2$$

where N is the number of samples.

In [16]:
```
# Read the Channel Input Response H2 from the Matlab's file
# a Matlab's matrix stays a Julia matrix when imported
H₂  = H2; @show typeof(H₂), size(H₂);

# We use Matlab notation, i.e., indexing starts at 1!
# Tha is also the Julia's default notation, one-index based
```

```
(typeof(H₂), size(H₂)) = (Matrix{ComplexF64}, (50, 10000))
```

Mathematical Expressions and Calculations

Let's break down each of the required steps and provide the corresponding mathematical expressions for each calculation.

1. Power Delay Profile (PDP)

The Power Delay Profile (PDP) $P(\tau)$ is obtained from the inverse Fourier transform of the channel's frequency response $H(f)$. Given the original time-variant transfer function $H_2(f)$:

$$P(\tau) = \mathbb{E}[|h(\tau)|^2]$$

where $h(\tau)$ is the inverse Fourier transform of $H(f)$:

$h(\tau) = \mathrm{IFFT}\{H_2(f)\}$ or $h(\tau) = \mathscr{F}^{-1}\{H_2(f)\}$

The PDP is then:

$$P(\tau) = \frac{1}{N} \sum_{k=1}^{N} |h_k(\tau)|^2$$

2. Total Power

The total power P_{total} is the sum of the PDP values:

$$P_{\text{total}} = \sum_{\tau} P(\tau)$$

3. Average Mean Delay

The average mean delay $\bar{\tau}$ is calculated by weighting the delays τ by the corresponding power values and normalizing by the total power:

$$\bar{\tau} = \frac{\sum_{\tau} \tau \cdot P(\tau)}{P_{\text{total}}}$$

4. Average RMS Delay Spread

The RMS delay spread τ_{rms} measures the spread of the delay values around the mean delay:

$$\tau_{\text{rms}} = \sqrt{\frac{\sum_{\tau} (\tau - \bar{\tau})^2 \cdot P(\tau)}{P_{\text{total}}}}$$

Alternatively, this can be calculated directly without subtracting the mean delay first:

$$\tau_{\text{rms}} = \sqrt{\frac{\sum_{\tau} \tau^2 \cdot P(\tau)}{P_{\text{total}}} - \bar{\tau}^2}$$

5. Coherence Bandwidth

The coherence bandwidth B_c is inversely related to the RMS delay spread. It represents the frequency range over which the channel can be considered "flat":

$$B_c \approx \frac{1}{\tau_{\text{rms}}}$$

(Note) When using LaTeX on Github and Gitlab, add three-character space in front of the formula.

Julia Code to Calculate and Plot

Here's how you can implement these calculations and plot the results in Julia:

In [17]:
```
using FFTW
using Unitful
using LinearAlgebra, Plots, Statistics
```

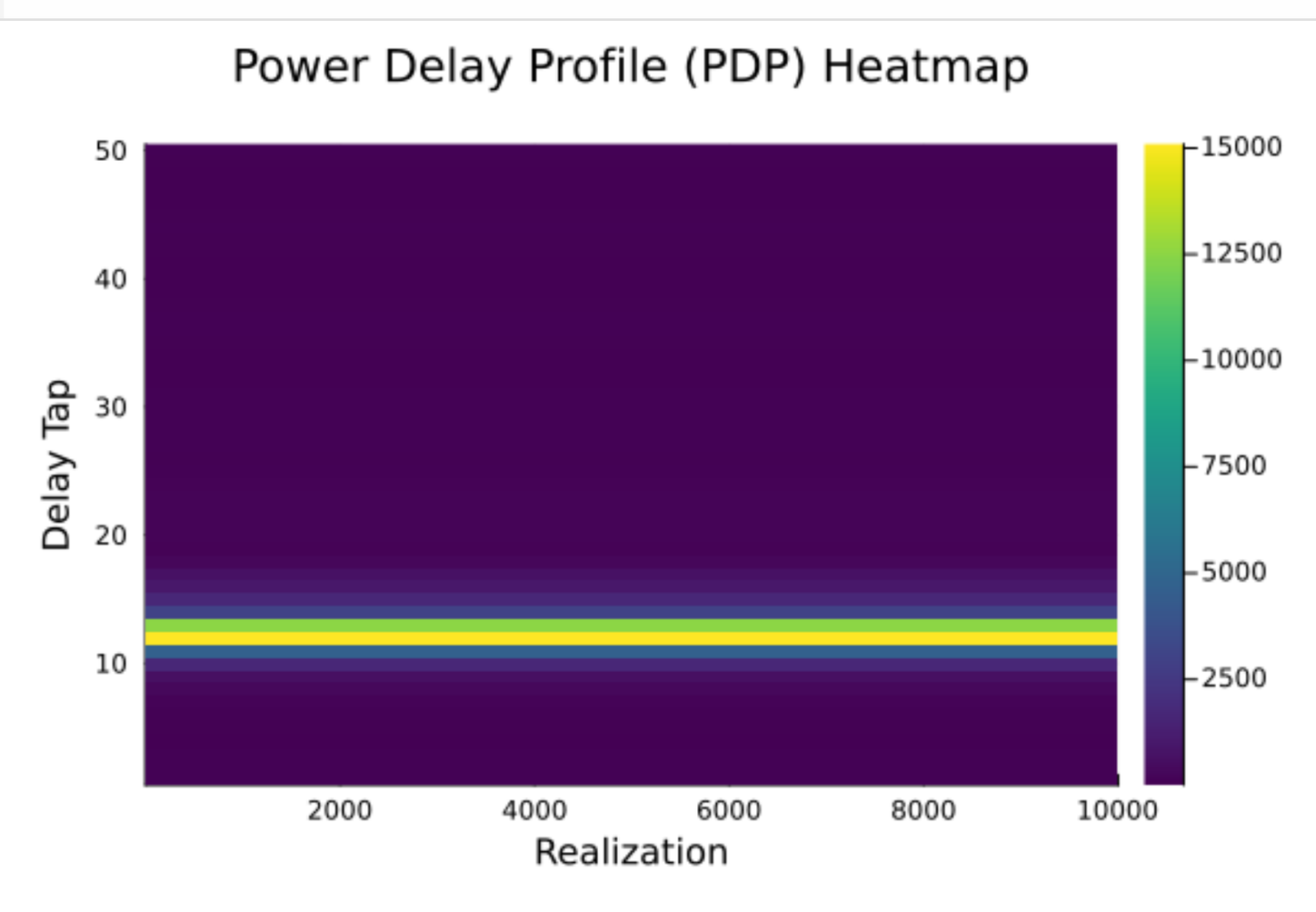
In [18]:
```
using FFTW, LinearAlgebra, Plots, Statistics, LaTeXStrings, Measures
```

In [19]:
```
# Step 1: Compute the inverse FFT along each column to get the time-domain impulse response
hₜ = ℱ⁻¹(H₂, 1);
```

In [20]:
```
# Step 2: Dimensions: Number of rows (delay taps), Number of columns (realizations)
K, N = size(H₂); @show K, N

# Step 3: Compute the Power Delay Profile (PDP) using for comprehension
Pₜ = [1/N * ∑(abs.(hₜ[i, :]).^2) for i in 1:K];
```

```
(K, N) = (50, 10000)
```

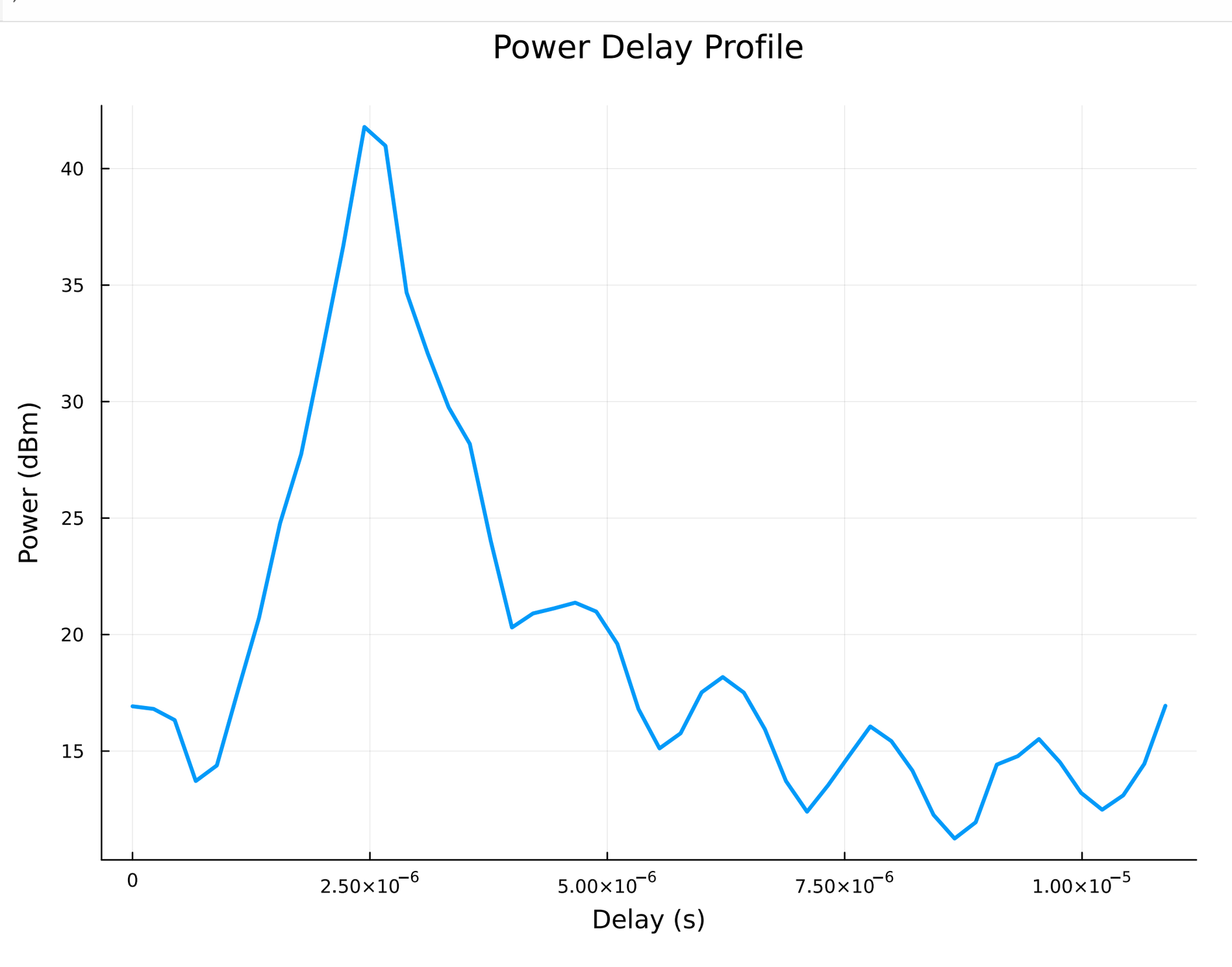
In [21]:
```
Pₜ_matrix = repeat(Pₜ, 1, N)

hp = heatmap(Pₜ_matrix
    , xlabel="Realization", ylabel="Delay Tap"
    , title="Power Delay Profile (PDP) Heatmap"
    , color=:viridis
    , margin = 5mm
)
savefig(hp, "images/pdp_heatmap.png");
```



In [22]:
```
# Define the delay time vector (assuming sampling interval of 222 ns)
Δt = uconvert(u"s", 222u"ns");
τ = Δt * (0:(K - 1))

# Convert τ to microseconds for plotting
τₘₛ = τ / u"μs"

# Calculate PDP in dBm for plotting
Pₗₙ₁₀ = 10 .* log10.(Pₜ);
```

In [23]:
```
# Plot the results
plot(τ, Pₗₙ₁₀, xlabel="Delay", ylabel="Power (dBm)", title="Power Delay Profile", label="PDP"
    , linewidth = 2.5
    # , xlims = (-0.1e-5 , 1.2e-5), ylims = (10 , 45 )
    , size = (800,600)
    , legend = :false
    , margin = 5mm

)
```

Out[23]:



In [24]:
```
# Convert PDP from dBm to linear scale for calculations
Pₗᵢₙₑₐᵣ = 10 .^ (Pₗₙ₁₀ / 10)

# Calculate total power
Pₜₒₜₐₗ = ∑(Pₗᵢₙₑₐᵣ)

# Calculate average mean delay
τ̄ = ∑(τ .* Pₗᵢₙₑₐᵣ) / Pₜₒₜₐₗ

# Calculate average RMS delay spread
τᵣₘₛ = √(∑((τ .^ 2) .* Pₗᵢₙₑₐᵣ) / Pₜₒₜₐₗ - τ̄ ^ 2)

# Calculate coherence bandwidth
B꜀ = 1 / τᵣₘₛ; # Typed \biB\_(
```

In [25]:
```
# Print results
println("Total Power: ", round(Pₜₒₜₐₗ, digits = 0))
println("Average Mean Delay: ", τ̄)
println("Average RMS Delay Spread: ", τᵣₘₛ)
println("Coherence Bandwidth: ", B꜀)
```

```
Total Power: 43140.0
Average Mean Delay: 2.6734898922932467e-6 s
Average RMS Delay Spread: 8.788280501017051e-7 s
Coherence Bandwidth: 1.137879019546852e6 s⁻¹
```

In [ ]: