Wadi Moughanim - Lecture 1 (2023/10/05): Exercise 1.3

Exercise 1.3 (Domains of Attraction: Illustration)

1. Illustrate the Phenomenon of Weak Convergence

Illustrate the phenomenon of weak convergence stated in Fisher and Tippett's theorem through the convergence of histograms (built from a random sample) of maxima towards histograms of the limit, for a negative shape parameter.

- **Choose a Distribution:** Choose a textbook distribution F in the Weibull domain of attraction and find appropriate norming sequences an, bn such that (MDA) holds.

We aim to show that the uniform distribution on the interval (0,1) belongs to the Weibull domain of attraction. We consider the cumulative distribution function (CDF) of $U(0,1)$,

$$F_{\text{unif}}(x) = x \cdot 1_{(0,1)}(x),$$

We proceed by choosing sequences a_n and b_n such that

$$a_n x + b_n \to \infty \quad \text{as} \quad n \to \infty.$$

Then :

$$F^n(a_n x + b_n) = (a_n x + b_n)^n,$$

valid for all $x \leq \frac{1-b_n}{a_n}$

Let's choose $b_n = 1$ and allow a_n to be any non-zero sequence and $x < 0$:

$$F^n(a_n x + b_n) = \exp(n \log(1 + a_n x)).$$

For a specific choice of $a_n = \frac{1}{n}$, the expression simplifies to

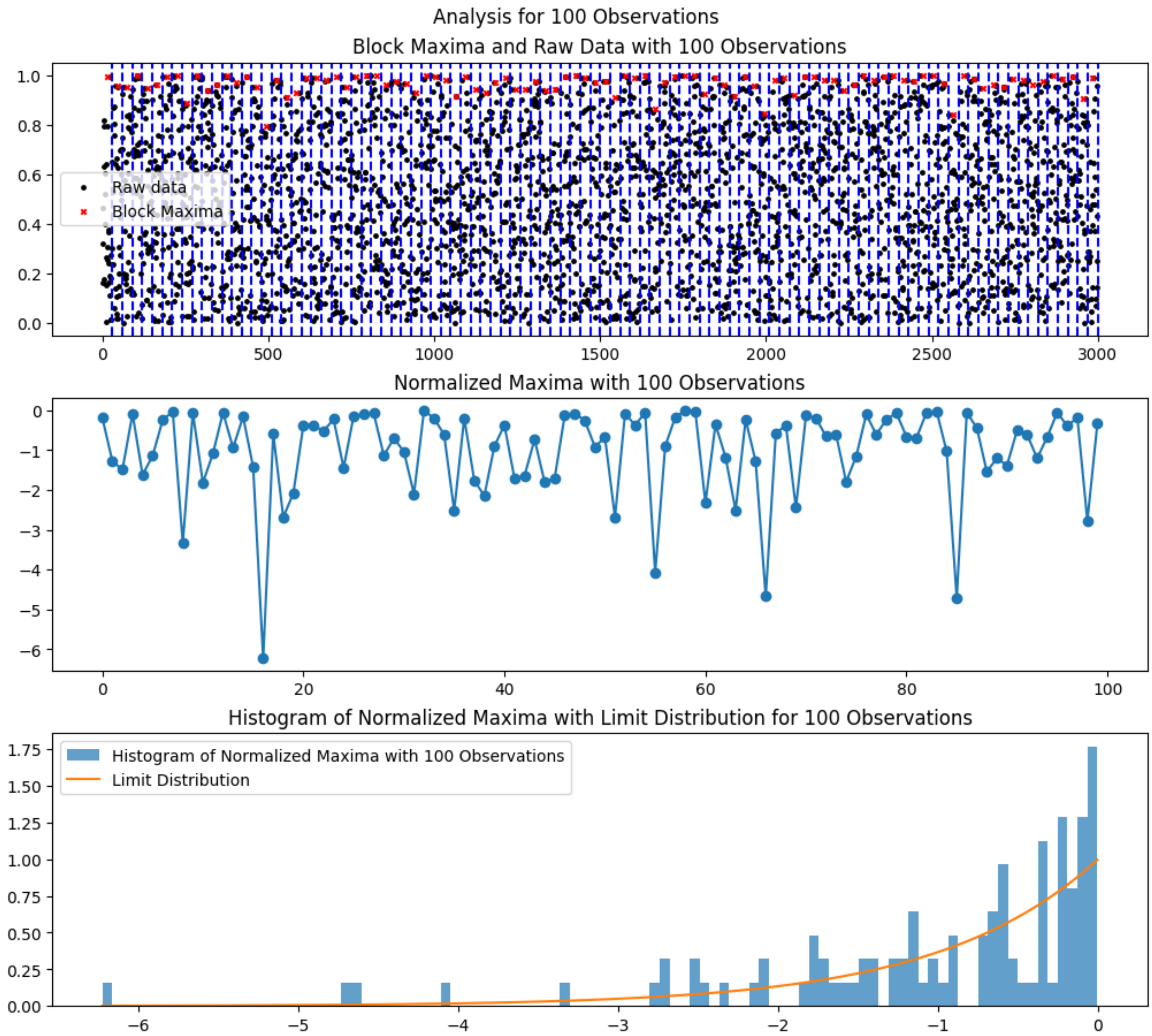
$$F^n\left(\frac{x}{n} + 1\right) \approx \exp(x) 1_{x<0}$$

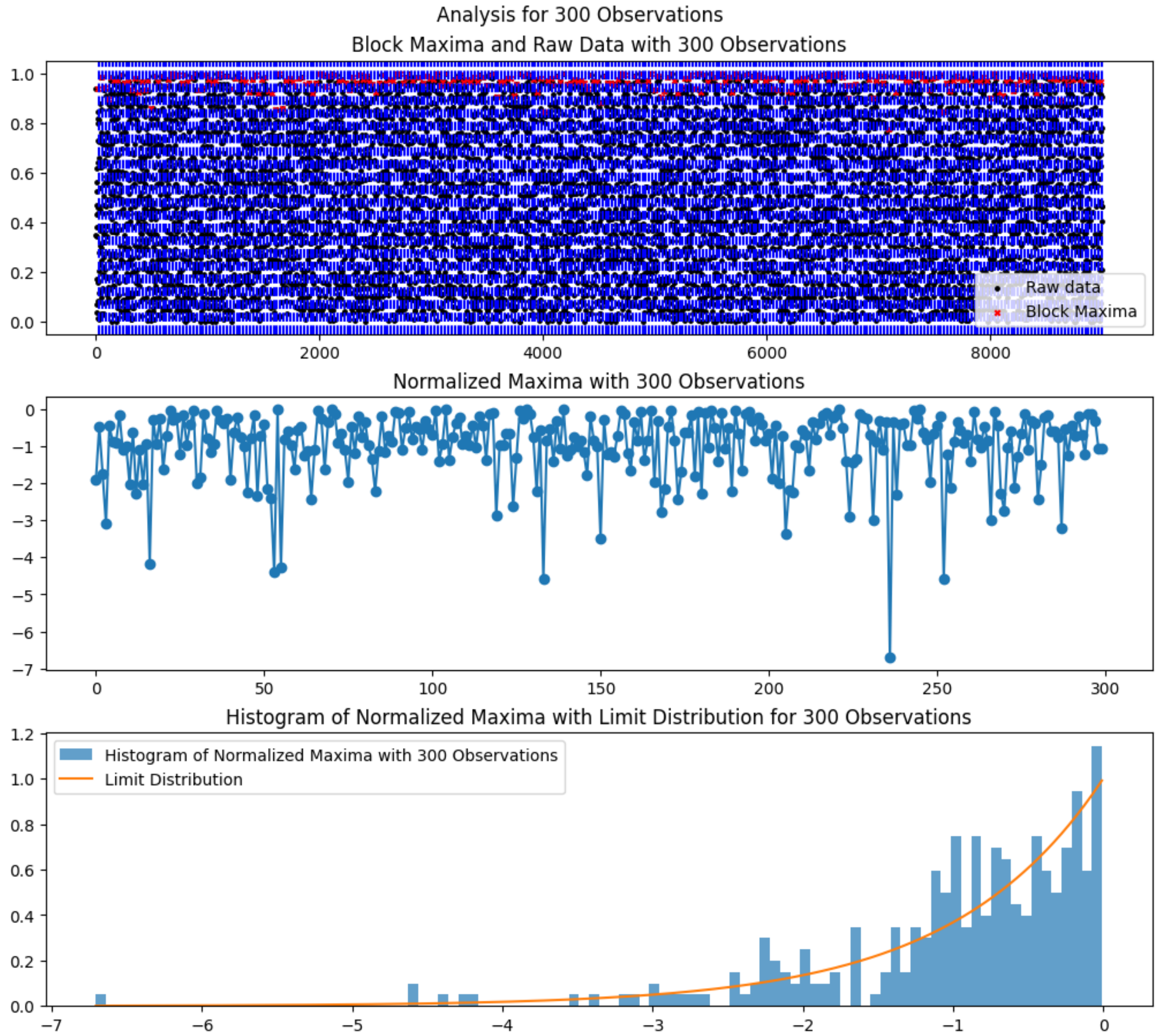
This is a particular case of the Weibull distribution with $\alpha = -1$. Hence, with the sequences $a_n = \frac{1}{n}$ and $b_n = 1$.

- **Write a Short Code:** Write a short code allowing to:
  - Generate M blocks of size n of independent random variables distributed according to F and normalize the block maxima.
  - Plot a histogram of the M normalized maxima and superimpose the histogram for the limit distribution in a visually illustrative manner.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

n = 30  # Size of each block

# Generate M blocks each of size n from U(0,1)
def generate_blocks(M, n):
    return np.random.rand(M, n)

# Compute block maxima
def compute_maxima(blocks):
    return np.max(blocks, axis=1)

# Normalize the block maxima
def normalize_maxima(maxima, n):
    a_n = 1.0 / n
    b_n = 1
    return (maxima - b_n)/a_n

def plot_analysis(blocks, block_maxima, normalized_maxima, M, nbins):
    fig, axs = plt.subplots(3, 1, figsize=(10, 9), constrained_layout=Tru

    # Raw Data and Block Maxima
    block_ends = np.arange(n, M*n+1, n)
    raw_data = blocks.flatten()
    axs[0].scatter(range(len(raw_data)), raw_data, s=6, c='black', label=
    axs[0].scatter(block_ends - n/2, block_maxima, s=10, c='red', marker=
    for end in block_ends:
        axs[0].axvline(x=end, color='blue', linestyle='--')
    axs[0].set_title(f"Block Maxima and Raw Data with {M} Observations")
    axs[0].legend()

    # Normalized Maxima
    axs[1].plot(normalized_maxima, label=f"Normalized Maxima with {M} Obs
    axs[1].set_title(f"Normalized Maxima with {M} Observations")

    # Histogram and Limit Distribution
    axs[2].hist(normalized_maxima, bins=nbins, density=True, label=f"Hist
    x_vals = np.linspace(min(normalized_maxima), max(normalized_maxima),
    y_vals = np.exp(x_vals)
    axs[2].plot(x_vals, y_vals, label='Limit Distribution')
    axs[2].set_title(f"Histogram of Normalized Maxima with Limit Distribu
    axs[2].legend()

    fig.suptitle(f"Analysis for {M} Observations")
    plt.show()

# 1
M1 = 100
blocks1 = generate_blocks(M1, n)
block_maxima1 = compute_maxima(blocks1)
normalized_maxima1 = normalize_maxima(block_maxima1, n)
plot_analysis(blocks1, block_maxima1, normalized_maxima1, M1, nbins=100)

# 2
M2 = 300
blocks2 = generate_blocks(M2, n)
block_maxima2 = compute_maxima(blocks2)
```

```
normalized_maxima2 = normalize_maxima(block_maxima2, n)
plot_analysis(blocks2, block_maxima2, normalized_maxima2, M2, nbins=100)
```



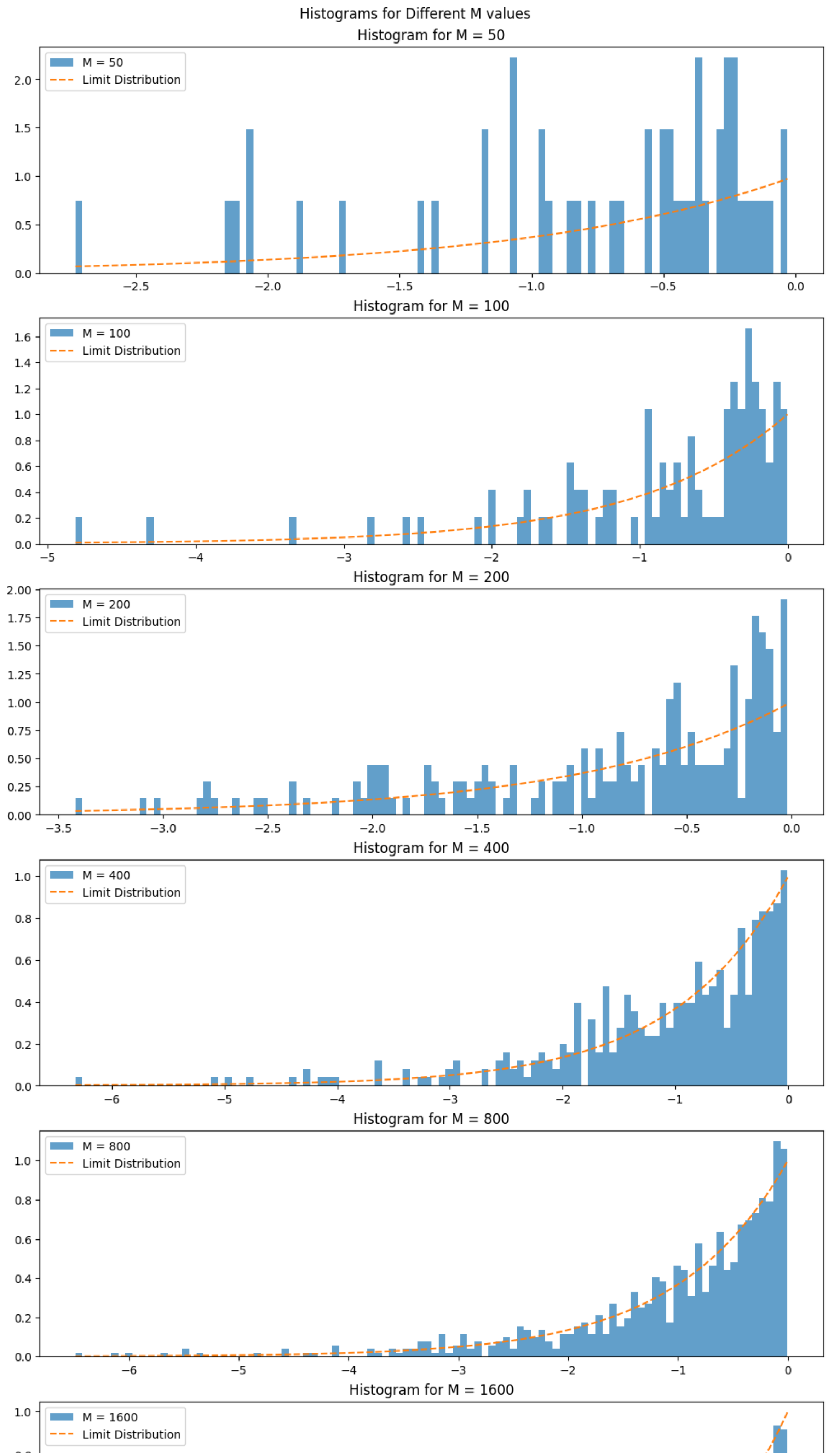


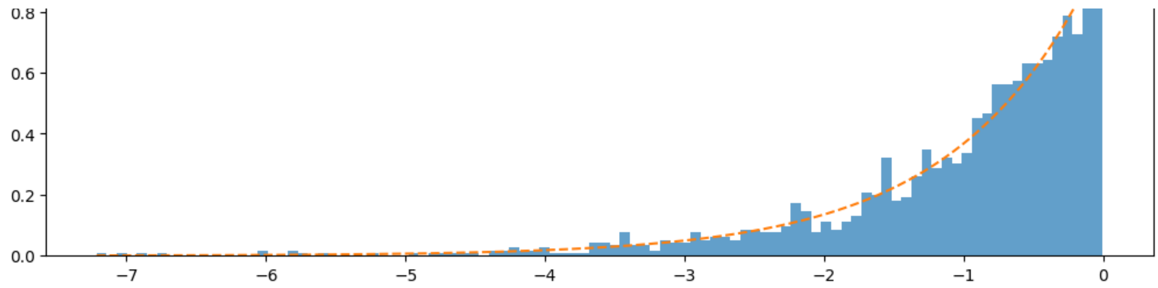
- **Vary M and n:** Let M and n vary so as to illustrate weak convergence of maxima as M → ∞. Explain the role of M and n in what you observe. Summarize the results in a figure including ≈ 6 such histograms with different values of M and a single (appropriate) value of n.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

n_values = [10, 20, 40, 80, 160, 320]
M_values = [50, 100, 200, 400, 800, 1600]

# Create a new figure with 6 subplots
fig, axs = plt.subplots(len(M_values), 1, figsize=(10, 20), constrained_l

# Populate the summary figure
for idx, (M, n) in enumerate(zip(M_values, n_values)):
    blocks = generate_blocks(M, n)
    block_maxima = compute_maxima(blocks)
    normalized_maxima = normalize_maxima(block_maxima, n)
    x_vals = np.linspace(min(normalized_maxima), max(normalized_maxima),
    y_vals = np.exp(x_vals)

    # Histogram
    axs[idx].hist(normalized_maxima, bins=100, density=True, label=f'M =
    # Limit Distribution
    axs[idx].plot(x_vals, y_vals, label='Limit Distribution', linestyle='
```

```
    axs[idx].set_title(f"Histogram for M = {M}")
    axs[idx].legend()

fig.suptitle("Histograms for Different M values")
plt.show()
```


Histograms for Different M values
Histogram for M = 50
M = 50
Limit Distribution
Histogram for M = 100
M = 100
Limit Distribution
Histogram for M = 200
M = 200
Limit Distribution
Histogram for M = 400
M = 400
Limit Distribution
Histogram for M = 800
M = 800
Limit Distribution
Histogram for M = 1600
M = 1600
Limit Distribution



2. Uniform Convergence of c.d.f's

Show (graphically and numerically) uniform convergence of c.d.f.'s. Explain why (i.e., prove that) weak convergence of normalized maxima indeed implies uniform convergence of c.d.f's.

The cdf. F_n of the normalized maxima converges to F at all cadlag points of F. Given that F is cadlag everywhere in the 3 cases Fréchet, Weibull and Gumbel, F_n converges to F for every $x \in \mathbb{R}$.

As F is increasing and for any $\epsilon > 0$, we take points $-\infty = x_0 < x_1 < \cdots < x_k = +\infty$ such that $\forall i \in= 0, \ldots, k-1$ $F(i+1) - F(i) \leq \epsilon$

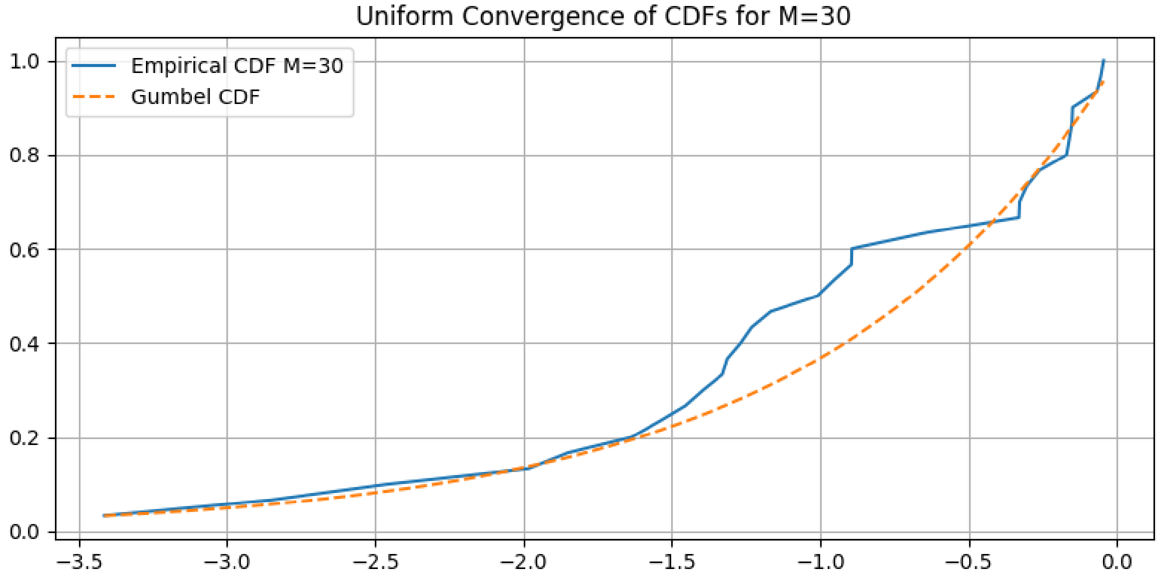
For any sub-interal $[x_i, x_{i+1}]$ of $[x_0, x_k]$, we have $F(x_{i+1}) - F(x_i) \leq \epsilon$. Thus as both F_n and F are increasing and bounded, we have $sup_{x\in[x_i,x_{i+1}]}|F_n(x) - F(x)| \leq max(|F_n(x_i) - F(x_i)|, |F_n(x_{i+1}) - F(x_{i+1})|) + \epsilon$.

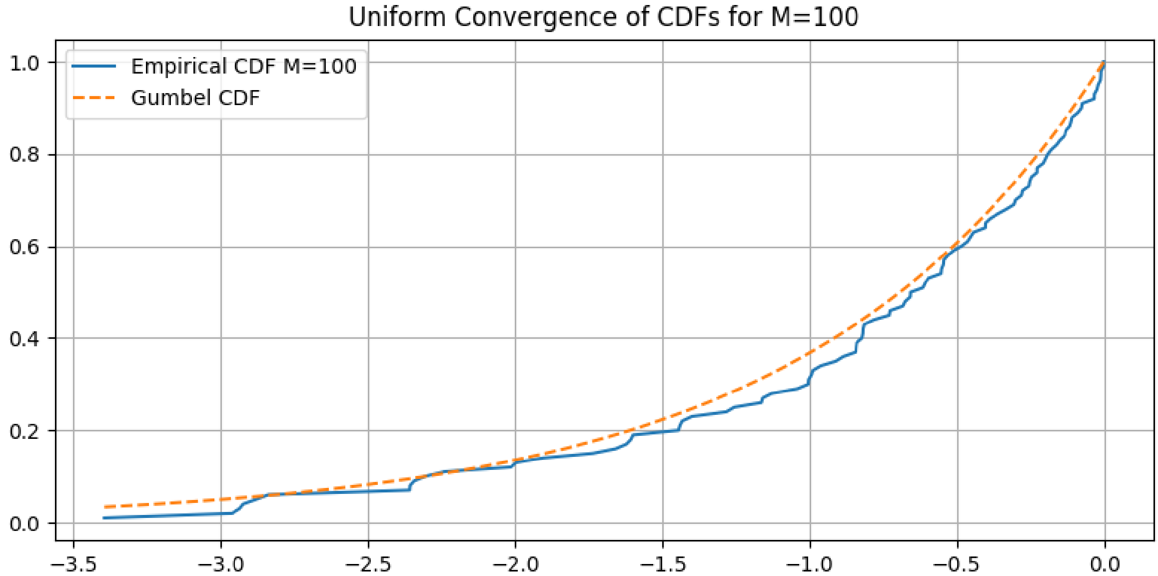
$$Thus: \sup_{x\in\mathbb{R}} |F_n(x) - F(x)| \leq \max_{j=0,1,...,K} |F_n(x_j) - F(x_j)| + \epsilon$$

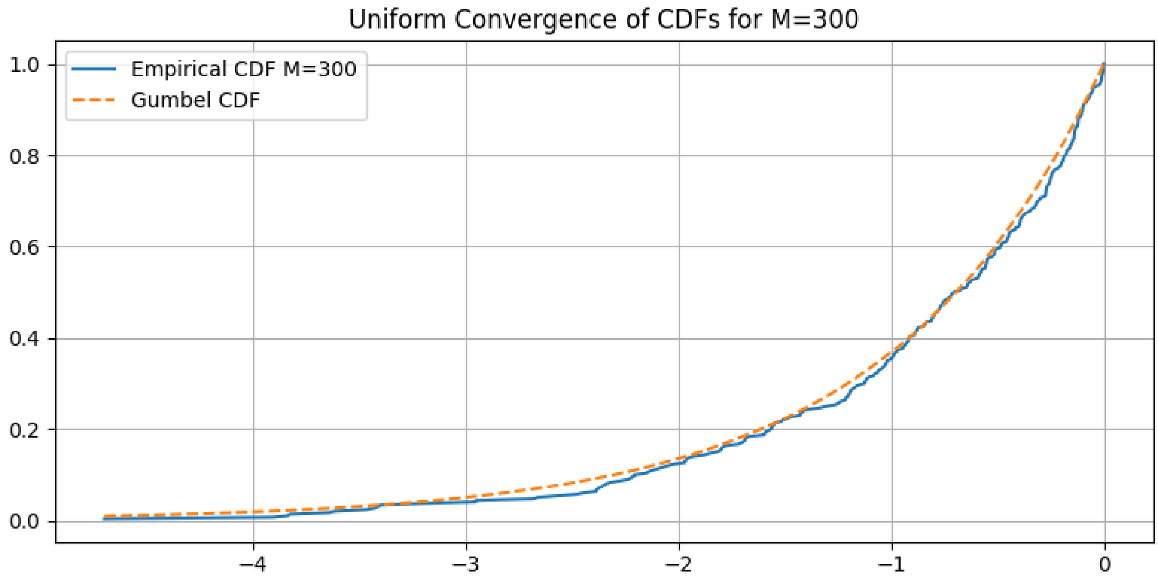
As n -> ∞, the term $\max_{j=0,1,...,K} |F_n(x_j) - F(x_j)|$ goes to 0 due to weak convergence. Thus, for sufficiently large n, the supremum of the absolute differences can be made smaller than any given ϵ, implying uniform convergence.

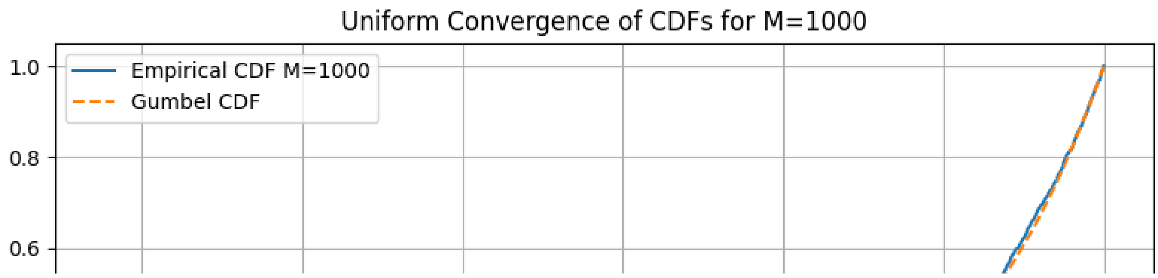
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
n = 30  # Size of each block

def generate_blocks(M, n):
    return np.random.rand(M, n)

def compute_maxima(blocks):
    return np.max(blocks, axis=1)

def normalize_maxima(maxima, n):
    a_n = 1.0 / n
    b_n = 1
    return (maxima - b_n) / a_n

def empirical_cdf(data):
    sorted_data = np.sort(data)
```

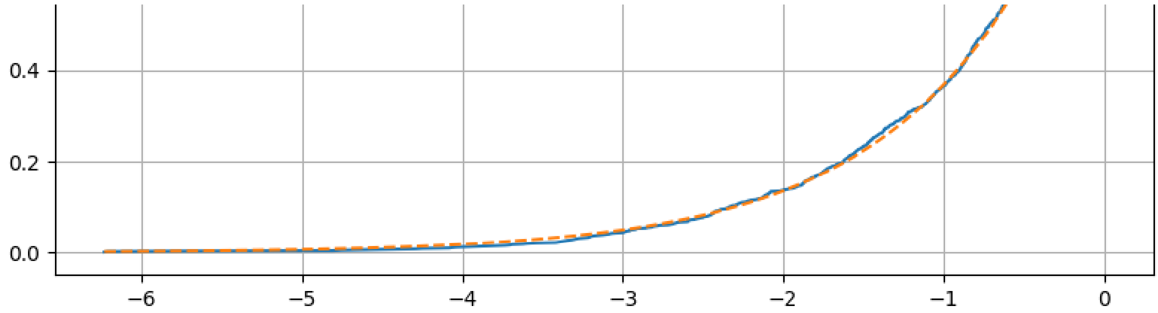
```
    y = np.arange(1, len(data) + 1) / len(data)
    return sorted_data, y

def weibull_cdf(x):
    return np.exp(x)

def plot_convergence(M_values):
    fig, axs = plt.subplots(len(M_values), 1, figsize=(8, 4*len(M_values)

    for i, M in enumerate(M_values):
        blocks = generate_blocks(M, n)
        maxima = compute_maxima(blocks)
        normalized_maxima = normalize_maxima(maxima, n)

        x_empirical, y_empirical = empirical_cdf(normalized_maxima)

        x_vals = np.linspace(min(normalized_maxima), max(normalized_maxim
        y_vals = weibull_cdf(x_vals)

        axs[i].plot(x_empirical, y_empirical, label=f'Empirical CDF M={M}
        axs[i].plot(x_vals, y_vals, '--', label='Gumbel CDF')
        axs[i].legend()
        axs[i].grid(True)
        axs[i].set_title(f'Uniform Convergence of CDFs for M={M}')

    plt.tight_layout()
    plt.show()

M_values = [30, 100, 300, 1000, 3000, 10000]
plot_convergence(M_values)
```

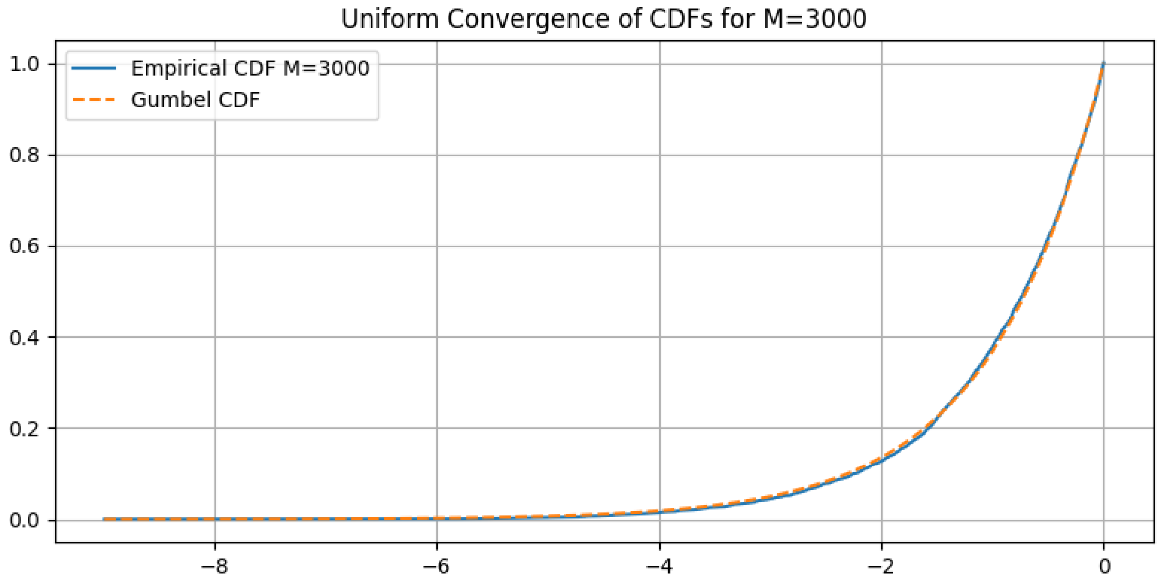

Uniform Convergence of CDFs for M=30
Empirical CDF M=30
Gumbel CDF
1.0
0.8
0.6
0.4
0.2
0.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0

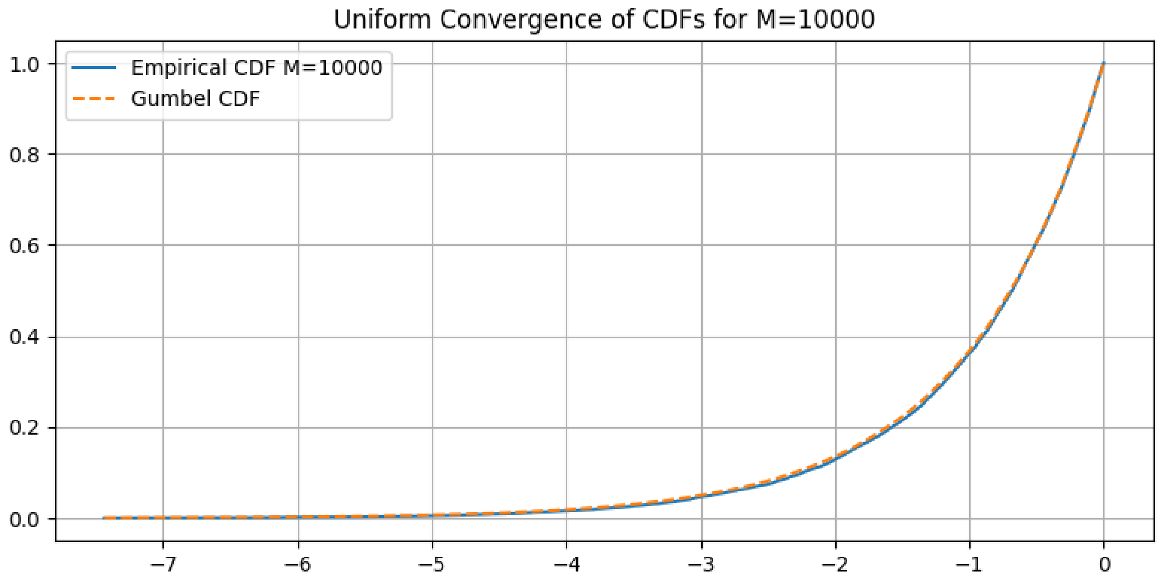

Uniform Convergence of CDFs for M=100
Empirical CDF M=100
Gumbel CDF
1.0
0.8
0.6
0.4
0.2
0.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0


Uniform Convergence of CDFs for M=300
Empirical CDF M=300
Gumbel CDF
1.0
0.8
0.6
0.4
0.2
0.0
−4
−3
−2
−1
0


Uniform Convergence of CDFs for M=1000
Empirical CDF M=1000
Gumbel CDF
1.0
0.8
0.6







3. Translated Pareto Distribution

Change the input distribution and work with a translated Pareto distribution, $P(X > x) = \left(\frac{x-\beta}{u}\right)^{-\alpha}$, for some $\alpha > 0, \beta, u \in \mathbb{R}, x \geq u + \beta$. Draw similar outputs as in the previous questions and compare the rate of convergence.

$$F(x) = 1 - \left(\frac{x-\beta}{u}\right)^{-\alpha} \quad \text{for} \quad x > u + \beta,$$

we are looking to find sequences a_n and b_n that normalize maxima.

$$P(X > x) = \left(\frac{x - \beta}{u}\right)^{-\alpha}.$$

Given the normalization $F_n(a_n x + b_n)$, and setting $b_n = \beta$:

$$F_n(a_n x + \beta) = \left(1 - \left(\frac{a_n x + \beta - \beta}{u}\right)^{-\alpha}\right)^n$$

$$F_n(a_n x + \beta) = \left(1 - \left(\frac{a_n x}{u}\right)^{-\alpha}\right)^n.$$

For large n, this behaves as:

$$F_n(a_n x + \beta) \approx \exp\left(n \log\left(1 - \left(\frac{a_n x}{u}\right)^{-\alpha}\right)\right).$$

For the normalization to converge to the Fréchet distribution, we aim for a structure that's proportional to $\exp(-x^{-\alpha})$.

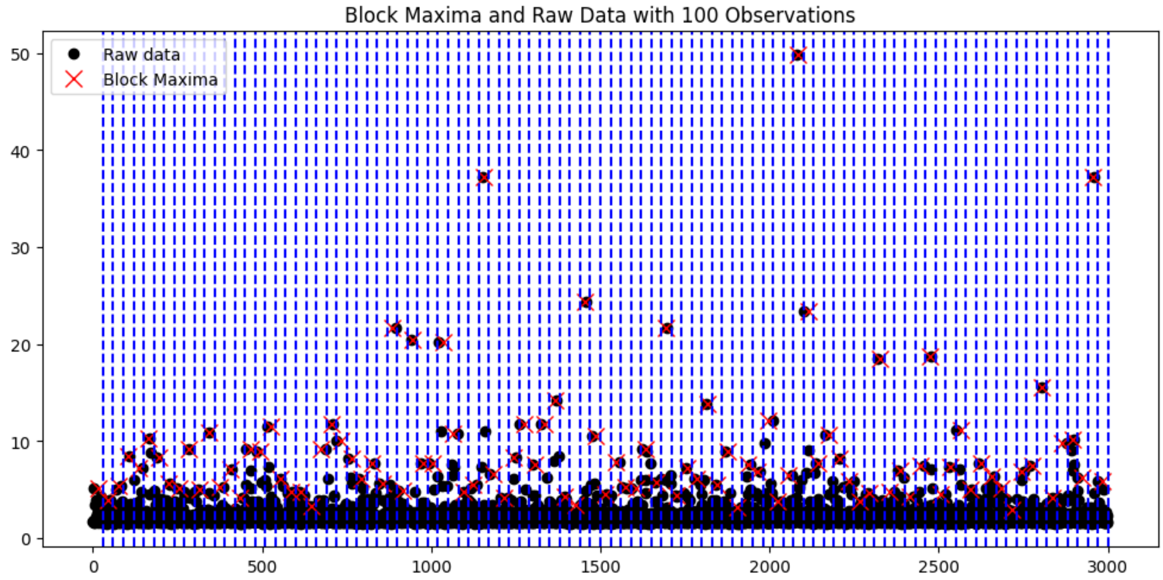
To attain this, let's choose $a_n = un^{1/\alpha}$. Plugging in this value:

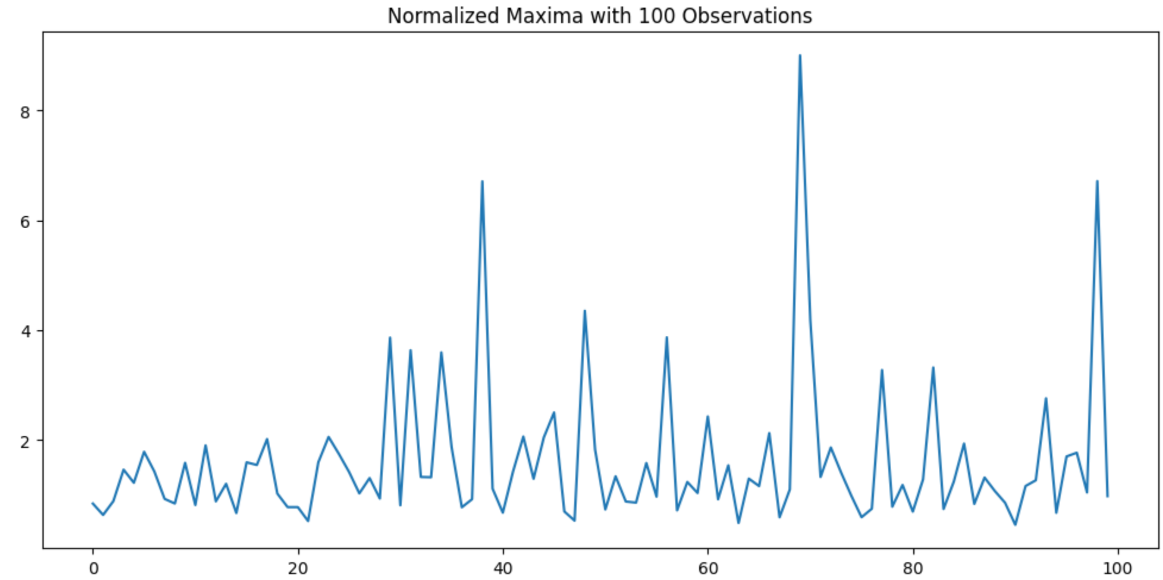
$$F_n(un^{1/\alpha} x + \beta) \approx \exp\left(n \log\left(1 - \left(\frac{n^{1/\alpha} x}{u}\right)^{-\alpha}\right)\right).$$

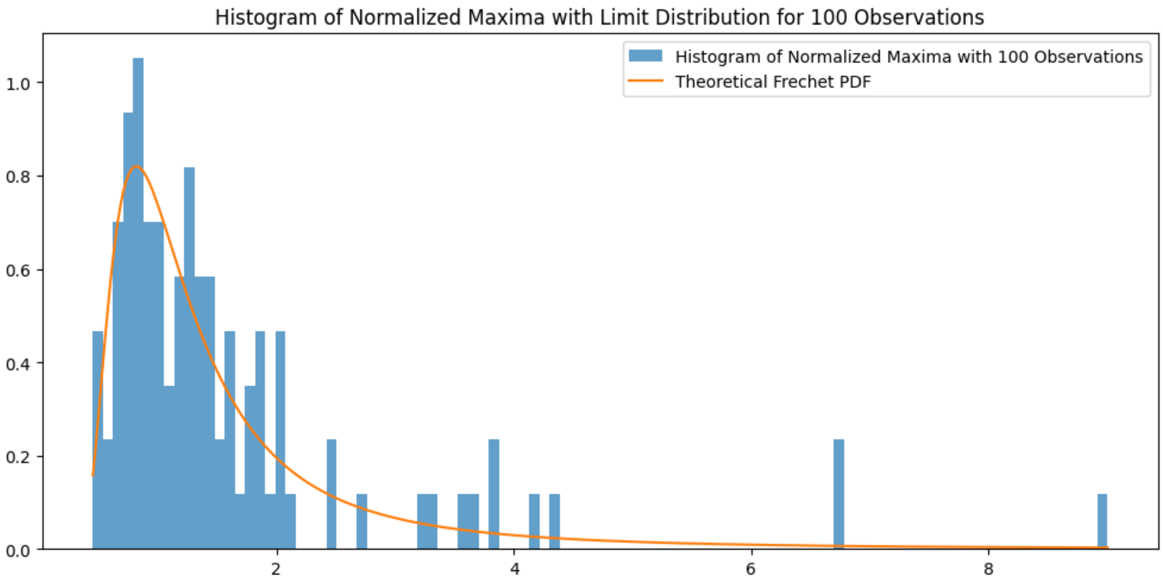
For large n, the term inside the logarithm gives us the desired structure, $\exp(-x^{-\alpha})$, which is the Fréchet distribution.

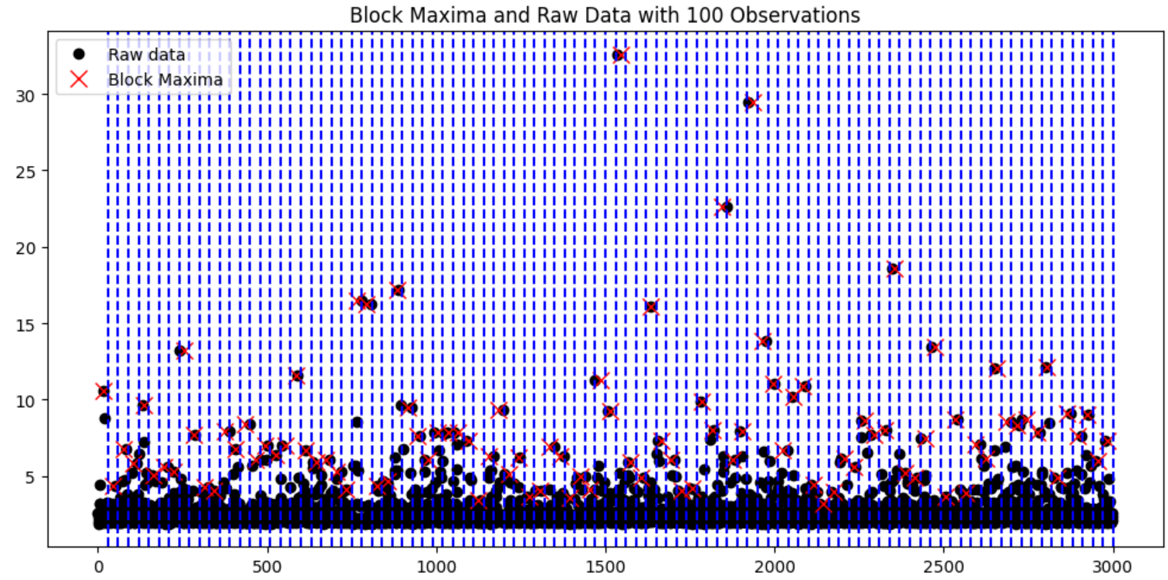
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
n = 30  # Size of each block

# List of parameter sets
parameter_sets = [
    {"alpha": 2.0, "beta": 0.5, "u": 1.0},
    {"alpha": 2.5, "beta": 0.7, "u": 1.2},
    {"alpha": 3.0, "beta": 0.9, "u": 1.5}
]

def generate_blocks(M, n, alpha, beta, u):
    uniform_samples = np.random.rand(M, n)
    pareto_samples = beta + u * ((1 - uniform_samples)**(-1/alpha))
    return pareto_samples

def compute_maxima(blocks):
    return np.max(blocks, axis=1)

def normalize_maxima(maxima, n):
    a_n = u * n**(1/alpha)
    b_n = beta
    return (maxima - b_n) / a_n

def plot_analysis(blocks, block_maxima, normalized_maxima, M, nbins, alph
    block_ends = np.arange(n, M*n+1, n)
```

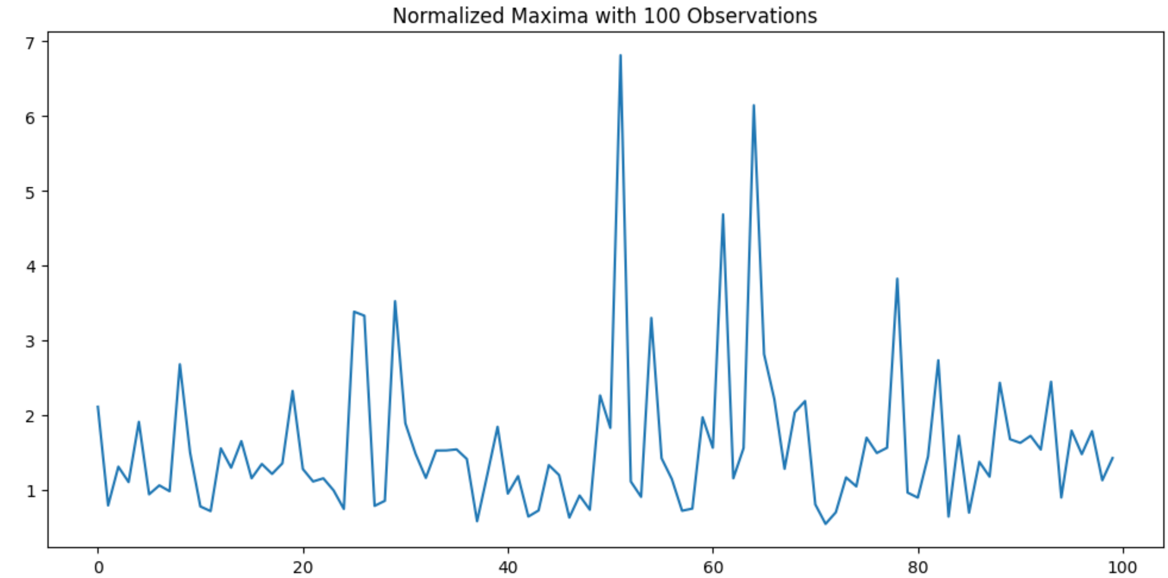
```
    raw_data = blocks.flatten()

    fig, axs = plt.subplots(3, 1, figsize=(10, 15))

    axs[0].plot(raw_data, 'ko', markersize=6, label='Raw data')
    axs[0].plot(block_ends-n/2, block_maxima, 'rx', markersize=10, label=
    for end in block_ends:
        axs[0].axvline(x=end, color='blue', linestyle='--')
    axs[0].legend()
    axs[0].set_title(f"Block Maxima and Raw Data with {M} Observations")

    axs[1].plot(normalized_maxima, label=f"Normalized Maxima with {M} Obs
    axs[1].set_title(f"Normalized Maxima with {M} Observations")

    axs[2].hist(normalized_maxima, bins=nbins, density=True, label=f"Hist
    x_vals = np.linspace(min(normalized_maxima), max(normalized_maxima),
    y_vals = alpha * x_vals**(-alpha - 1) * np.exp(-x_vals**(-alpha))
    axs[2].plot(x_vals, y_vals, label='Theoretical Frechet PDF')
    axs[2].set_title(f"Histogram of Normalized Maxima with Limit Distribu
    axs[2].legend()

    plt.tight_layout()
    plt.show()

for params in parameter_sets:
    alpha, beta, u = params["alpha"], params["beta"], params["u"]
    blocks = generate_blocks(M1, n, alpha, beta, u)
    block_maxima = compute_maxima(blocks)
    normalized_maxima = normalize_maxima(block_maxima, n)
    plot_analysis(blocks, block_maxima, normalized_maxima, M1, nbins=100,
```

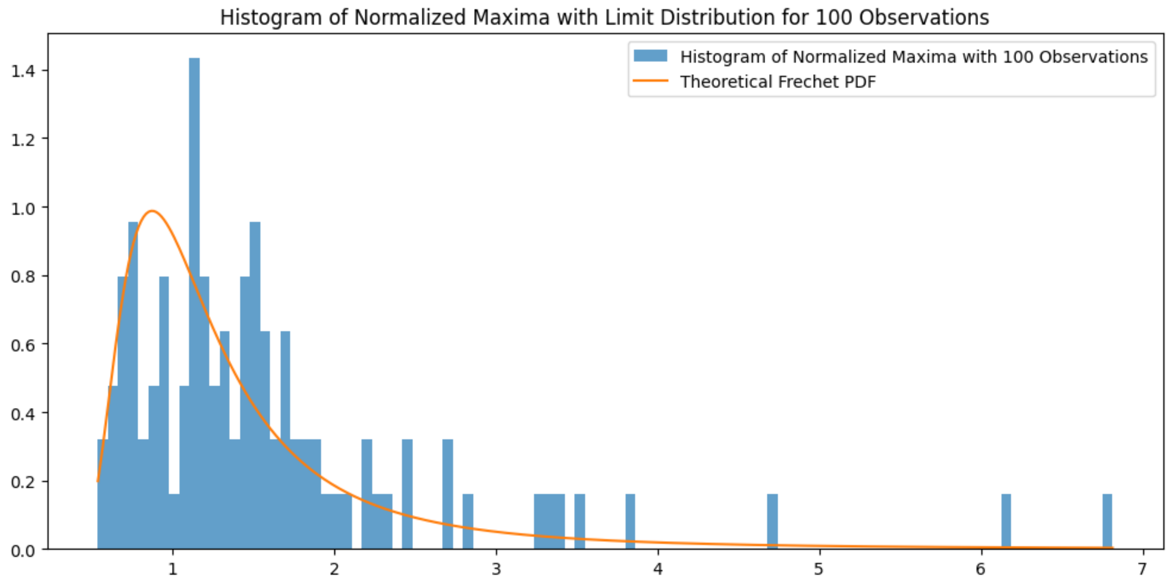

Block Maxima and Raw Data with 100 Observations
Raw data
Block Maxima
0
10
20
30
40
50
0
500
1000
1500
2000
2500
3000

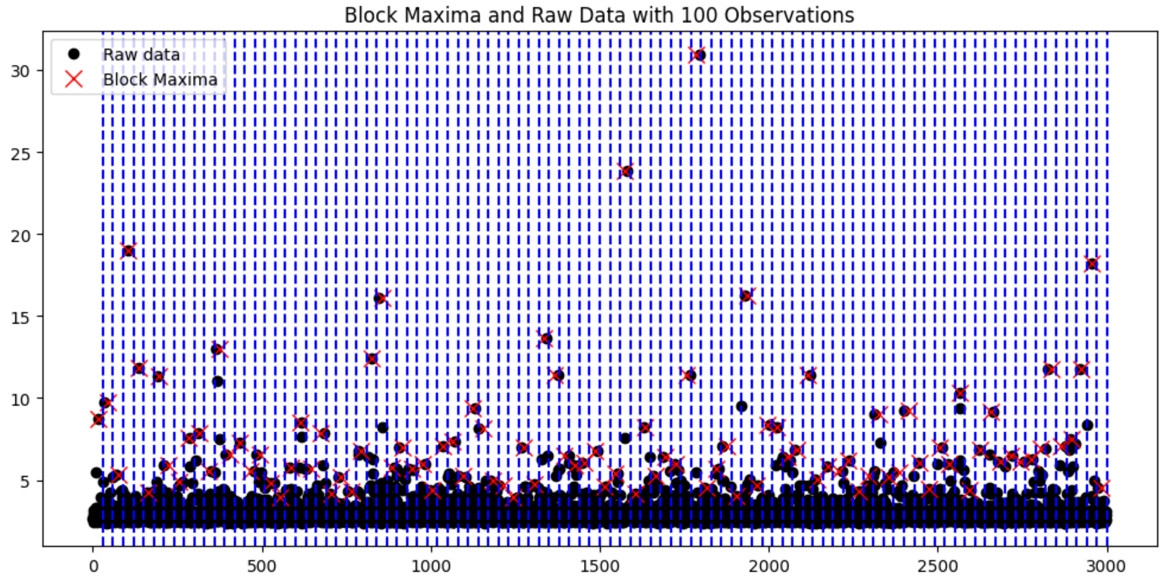

Normalized Maxima with 100 Observations
2
4
6
8
0
20
40
60
80
100

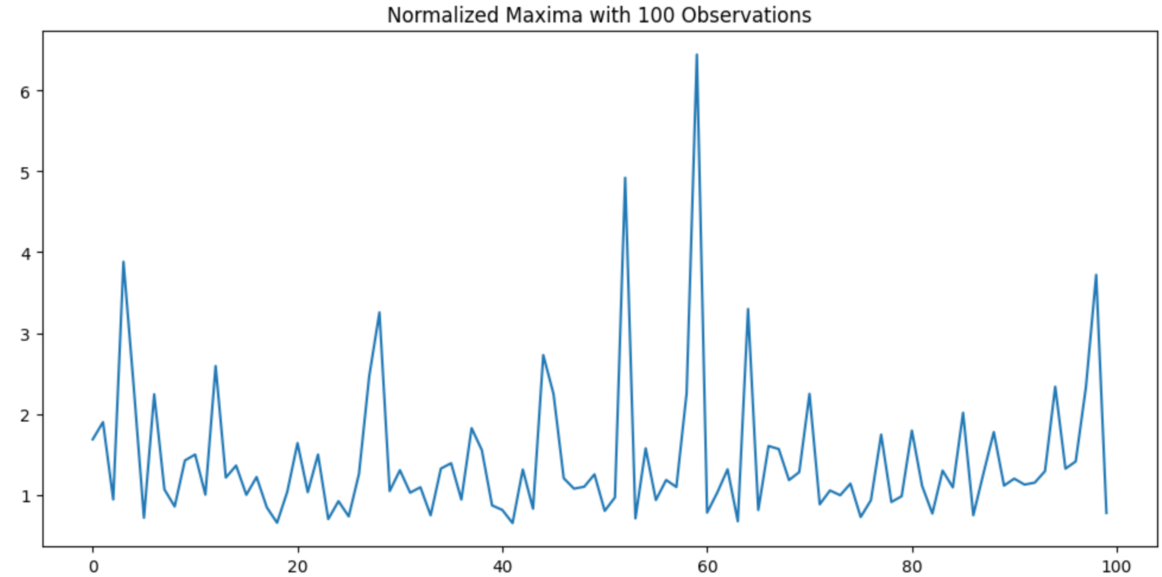

Histogram of Normalized Maxima with Limit Distribution for 100 Observations
Histogram of Normalized Maxima with 100 Observations
Theoretical Frechet PDF
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8

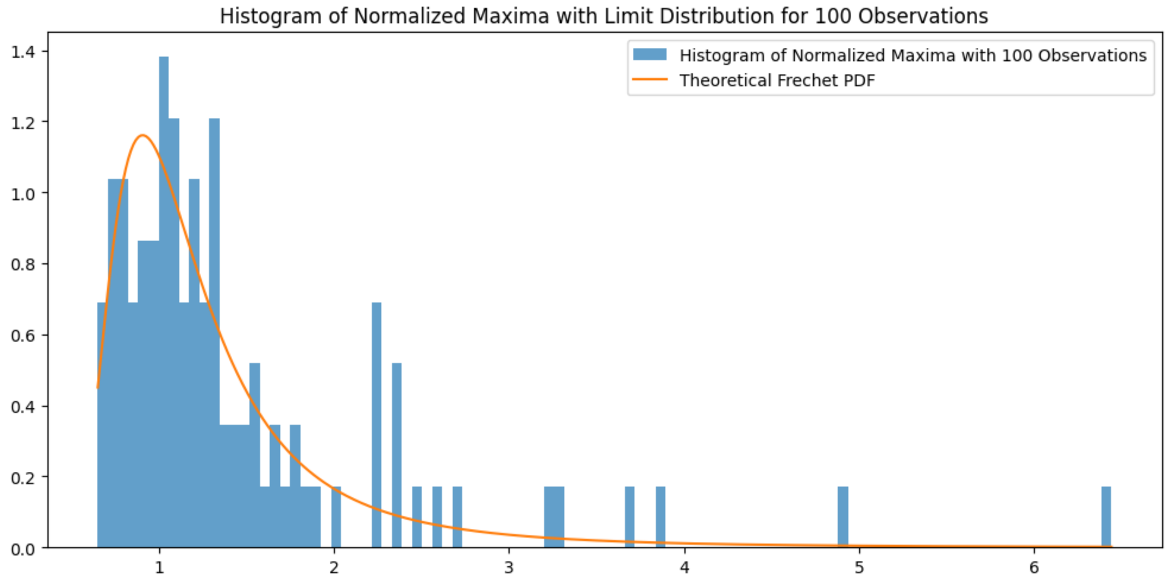

Block Maxima and Raw Data with 100 Observations
Raw data
Block Maxima


Normalized Maxima with 100 Observations


Histogram of Normalized Maxima with Limit Distribution for 100 Observations
Histogram of Normalized Maxima with 100 Observations
Theoretical Frechet PDF



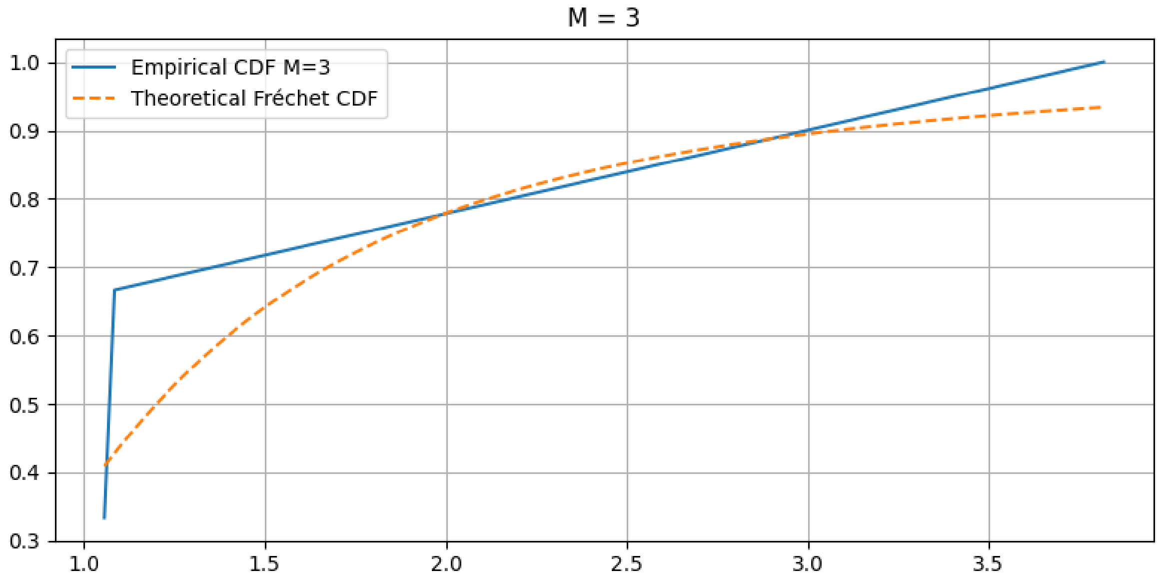


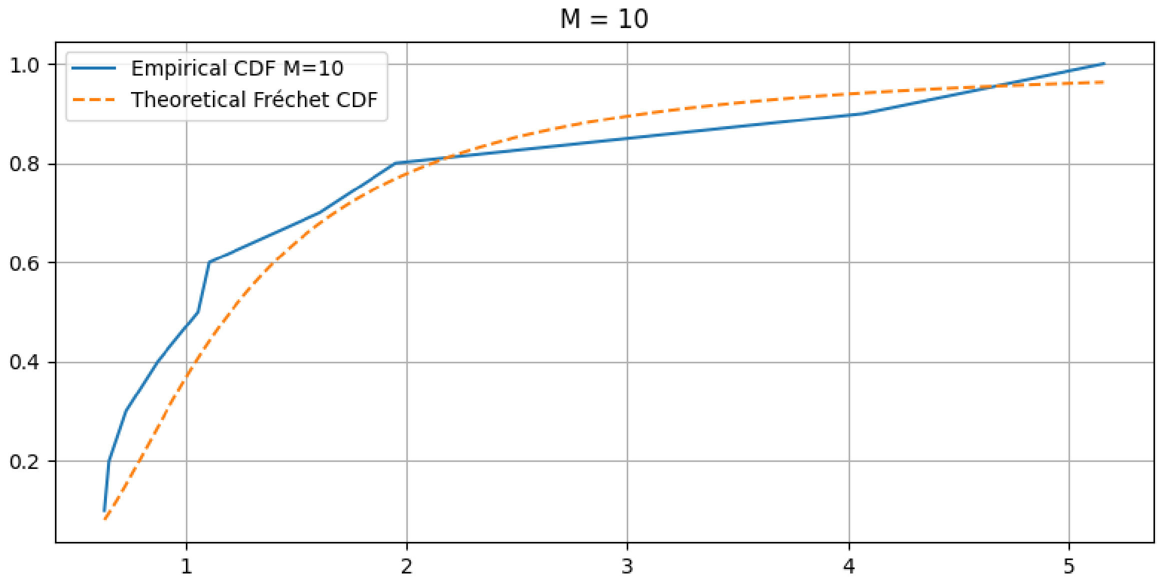


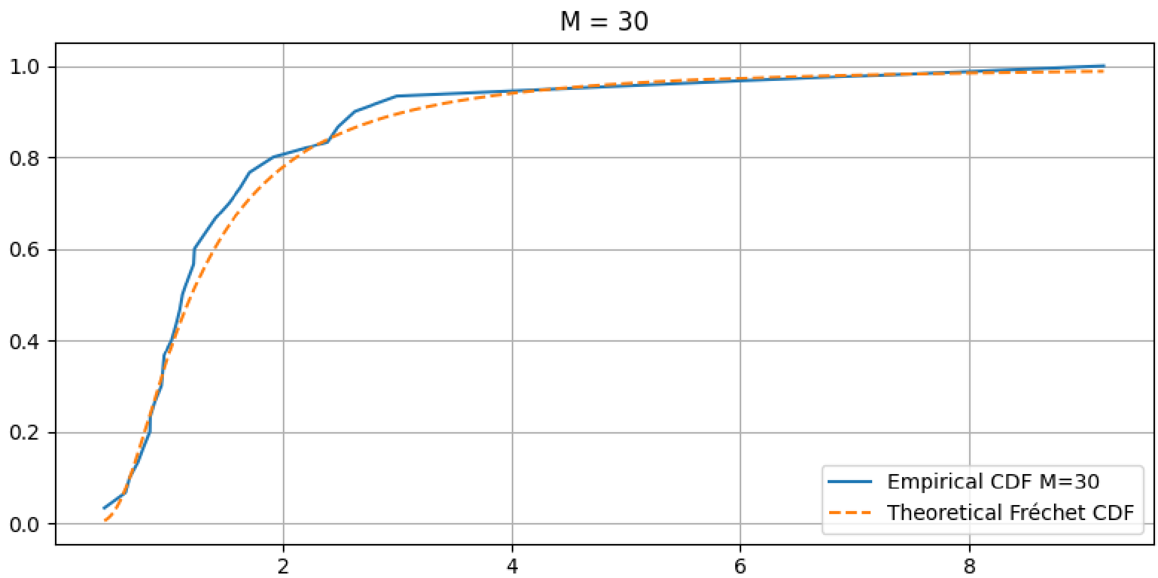
In [ ]:

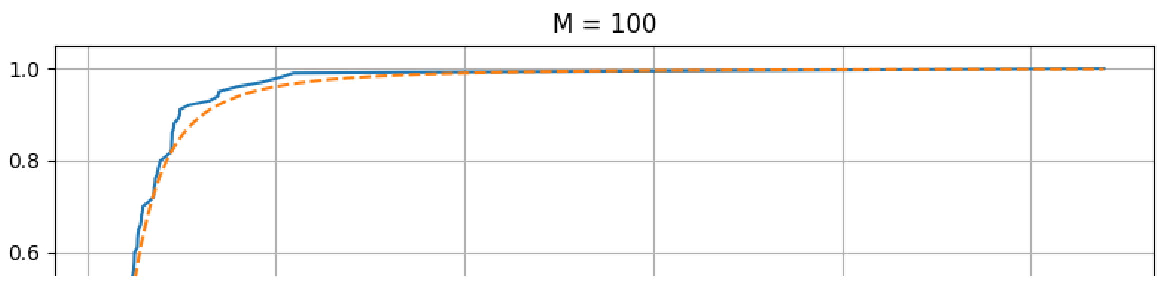
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
n = 30  # Size of each block

# List of parameter sets
```

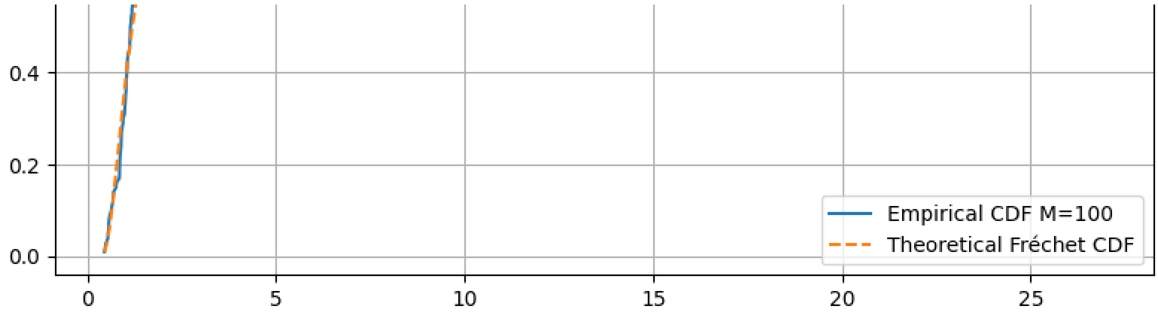
```
parameter_sets = [
    {"alpha": 2.0, "beta": 0.5, "u": 1.0},
    {"alpha": 2.5, "beta": 0.7, "u": 1.2},
    {"alpha": 3.0, "beta": 0.9, "u": 1.5}
]

def generate_blocks(M, n, alpha, beta, u):
    uniform_samples = np.random.rand(M, n)
    pareto_samples = beta + u * ((1 - uniform_samples)**(-1/alpha))
    return pareto_samples

def compute_maxima(blocks):
    return np.max(blocks, axis=1)

def normalize_maxima(maxima, n, alpha, u, beta):
    a_n = u * n**(1/alpha)
    b_n = beta
    return (maxima - b_n) / a_n

def empirical_cdf(data):
    sorted_data = np.sort(data)
    y = np.arange(1, len(data) + 1) / len(data)
    return sorted_data, y

def frechet_cdf(x, alpha):
    return np.exp(-x**(-alpha))

def plot_convergence(M_values, alpha, beta, u):
    fig, axs = plt.subplots(len(M_values), 1, figsize=(8, 4*len(M_values))

    for i, M in enumerate(M_values):
        blocks = generate_blocks(M, n, alpha, beta, u)
        maxima = compute_maxima(blocks)
        normalized_maxima = normalize_maxima(maxima, n, alpha, u, beta)

        x_empirical, y_empirical = empirical_cdf(normalized_maxima)

        x_vals = np.linspace(min(normalized_maxima), max(normalized_maxim
        y_vals = frechet_cdf(x_vals, alpha)

        axs[i].plot(x_empirical, y_empirical, label=f'Empirical CDF M={M}
        axs[i].plot(x_vals, y_vals, '--', label='Theoretical Fréchet CDF'
        axs[i].legend()
        axs[i].set_title(f"M = {M}")
        axs[i].grid(True)

    plt.tight_layout()
    plt.show()

# List of sample sizes to analyze
M_values = [3, 10, 30, 100, 300, 1000]

# Iterate over parameter sets and plot the convergence for each
for params in parameter_sets:
    alpha, beta, u = params["alpha"], params["beta"], params["u"]
    plot_convergence(M_values, alpha, beta, u)
```

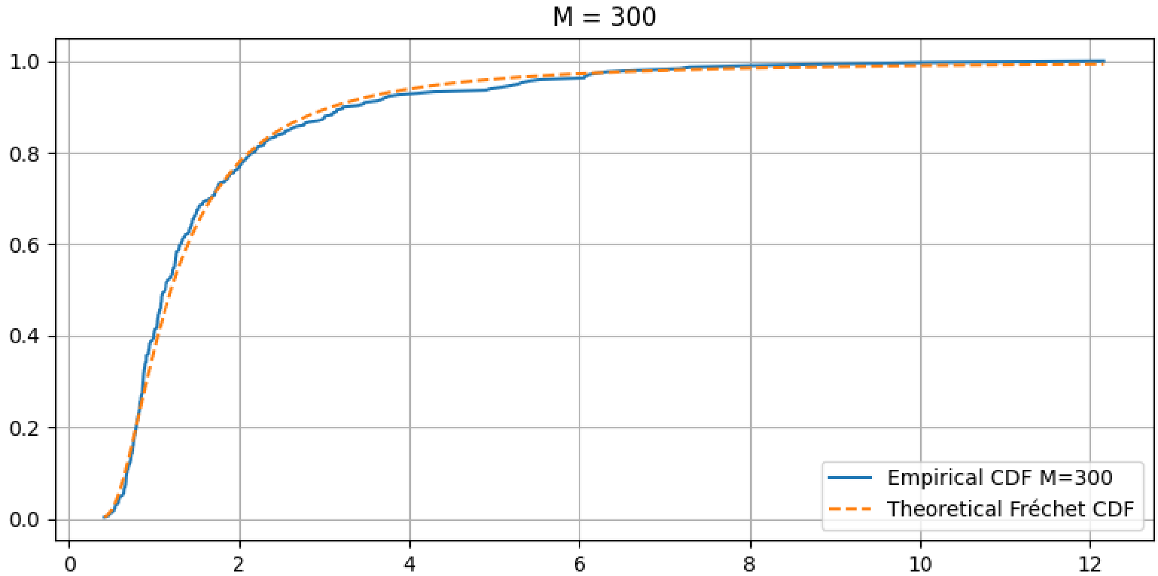

M = 3
Empirical CDF M=3
Theoretical Fréchet CDF

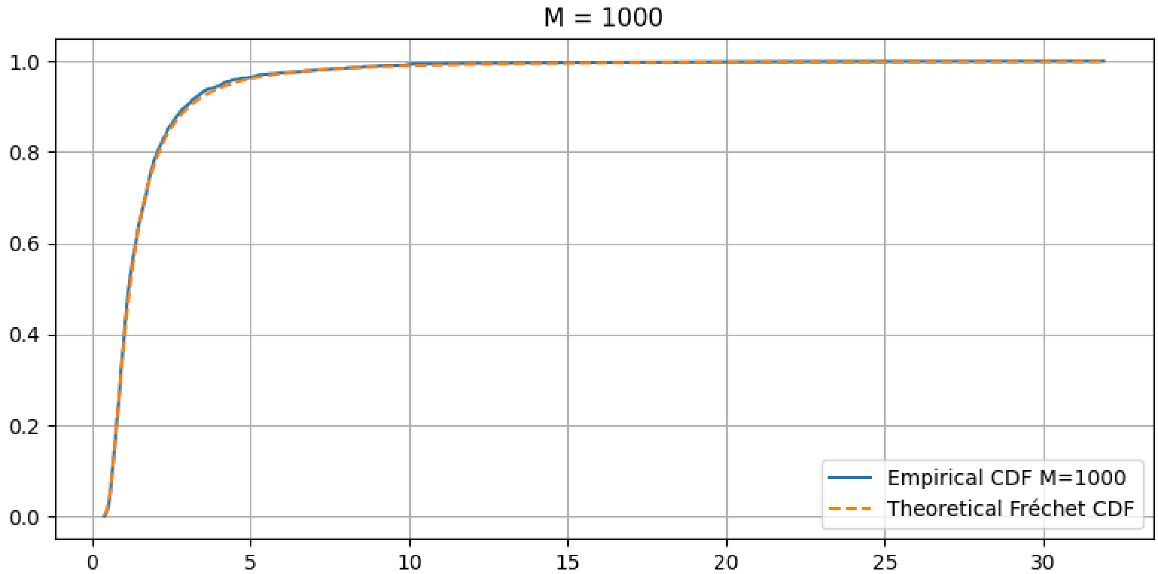

M = 10
Empirical CDF M=10
Theoretical Fréchet CDF

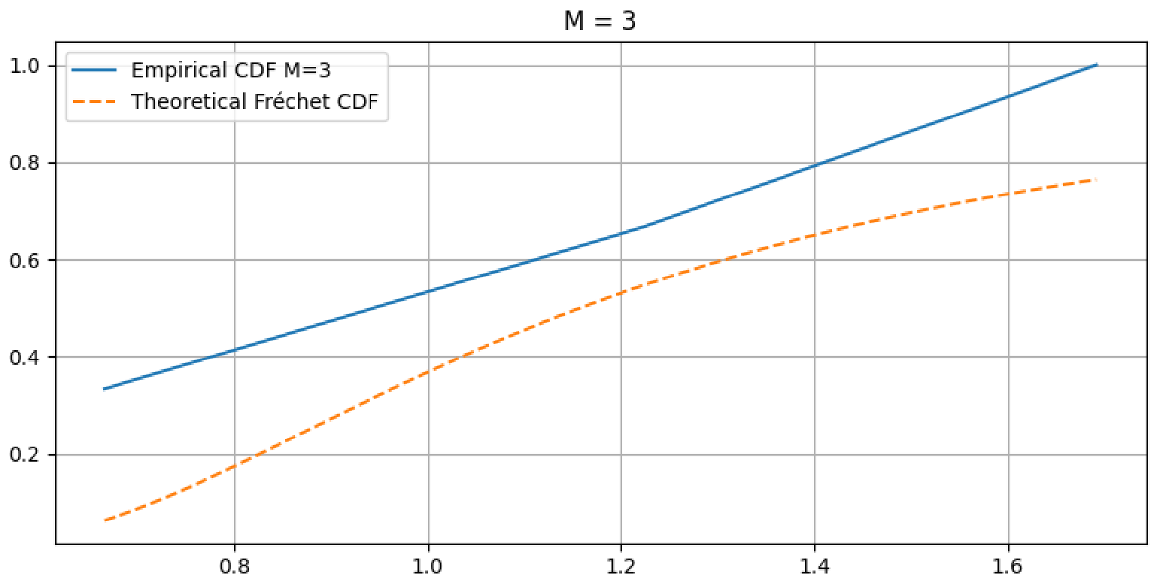

M = 30
Empirical CDF M=30
Theoretical Fréchet CDF

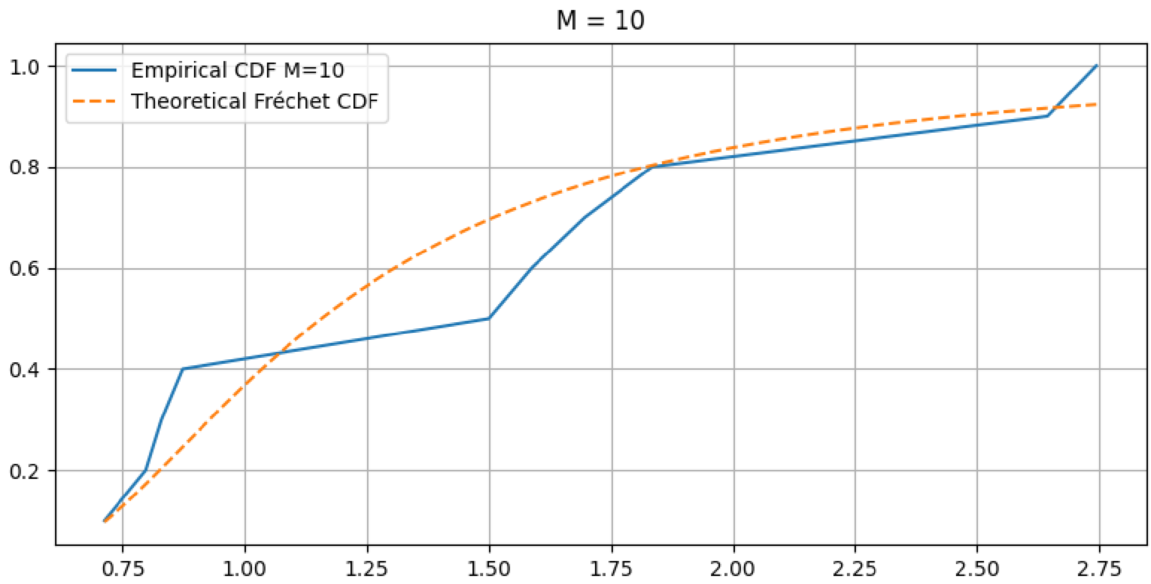

M = 100

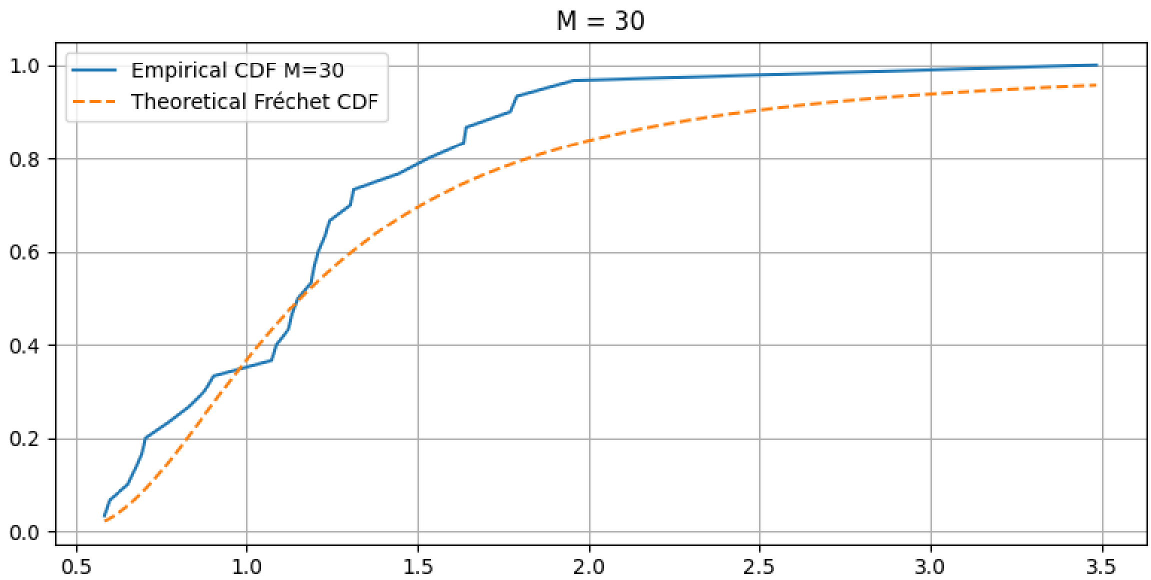

0.4
0.2
0.0
0
5
10
15
20
25
Empirical CDF M=100
Theoretical Fréchet CDF

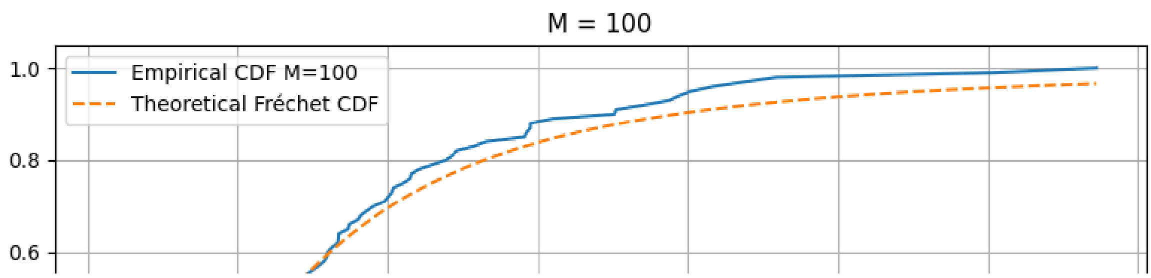

M = 300
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
Empirical CDF M=300
Theoretical Fréchet CDF

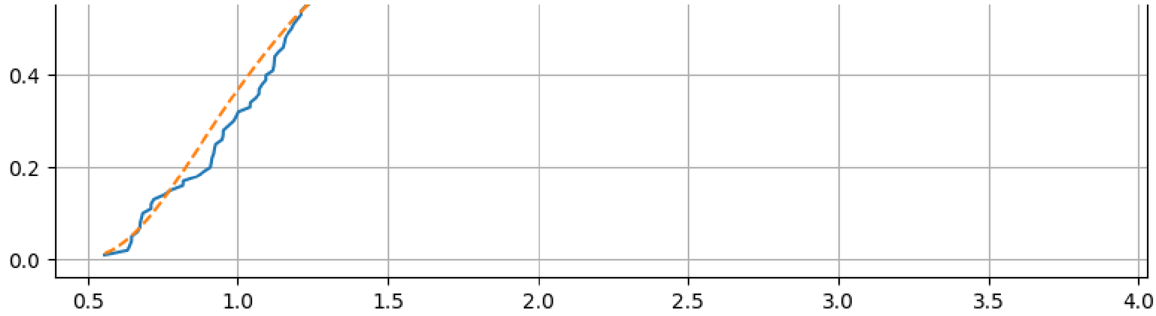

M = 1000
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
Empirical CDF M=1000
Theoretical Fréchet CDF

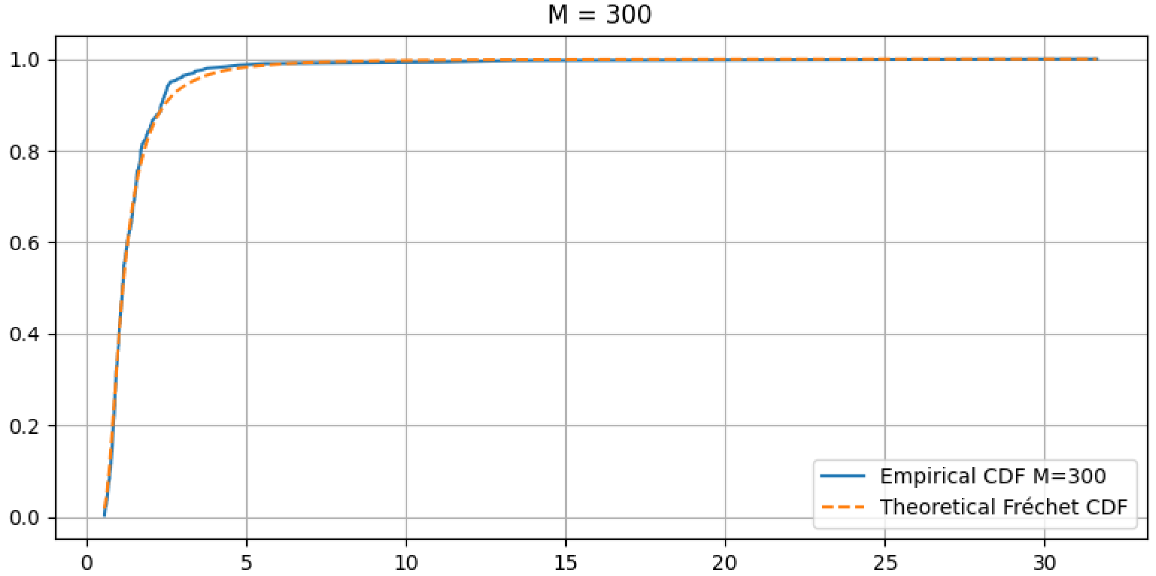

M = 3
Empirical CDF M=3
Theoretical Fréchet CDF
1.0
0.8
0.6
0.4
0.2
0.8
1.0
1.2
1.4
1.6

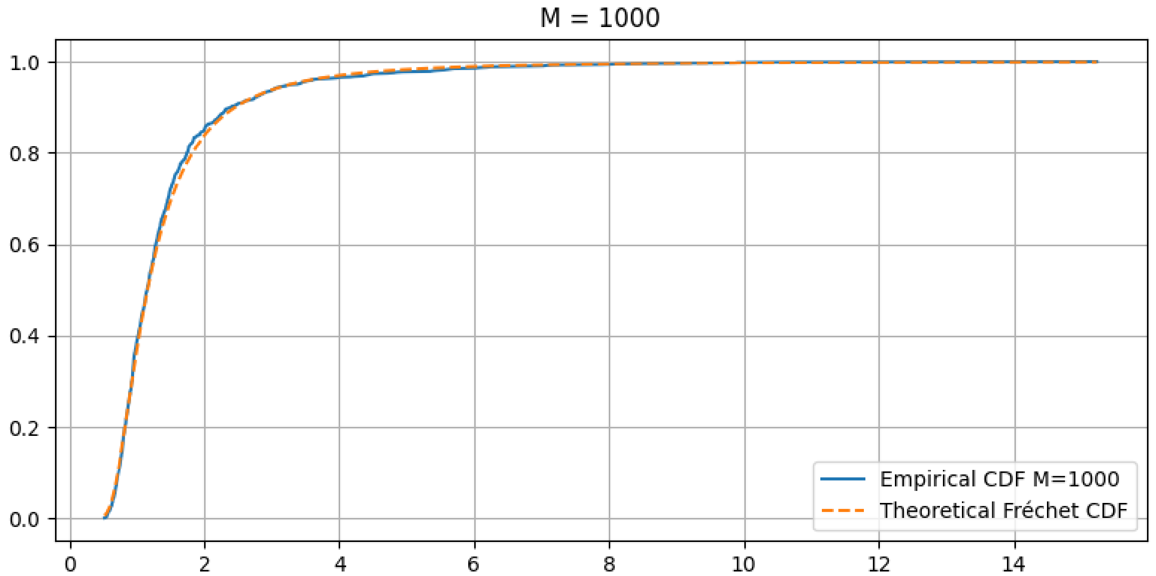

M = 10
Empirical CDF M=10
Theoretical Fréchet CDF
1.0
0.8
0.6
0.4
0.2
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75


M = 30
Empirical CDF M=30
Theoretical Fréchet CDF
1.0
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

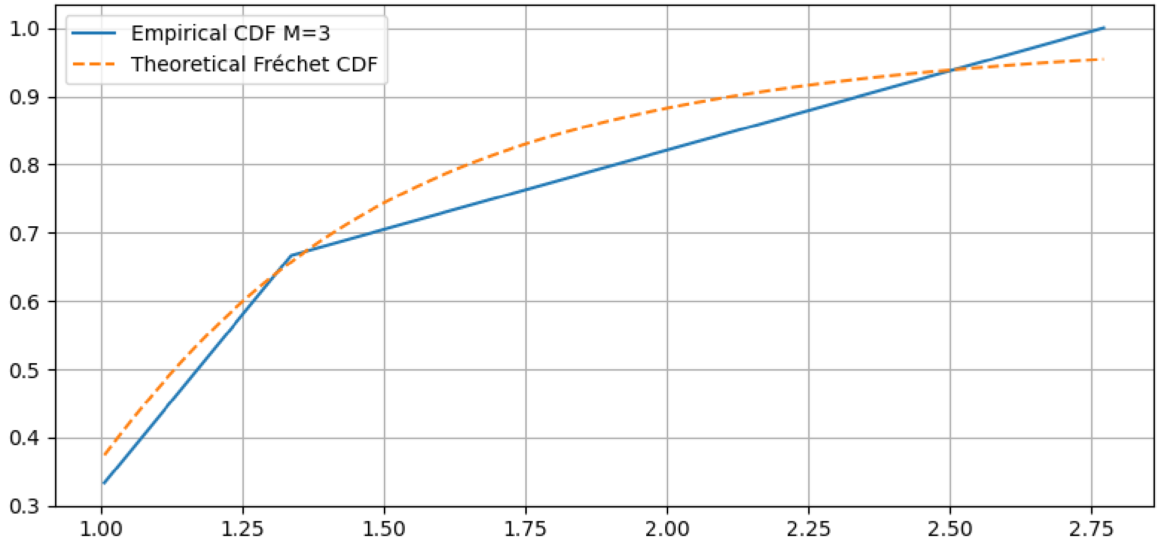

M = 100
Empirical CDF M=100
Theoretical Fréchet CDF
1.0
0.8
0.6


0.4
0.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

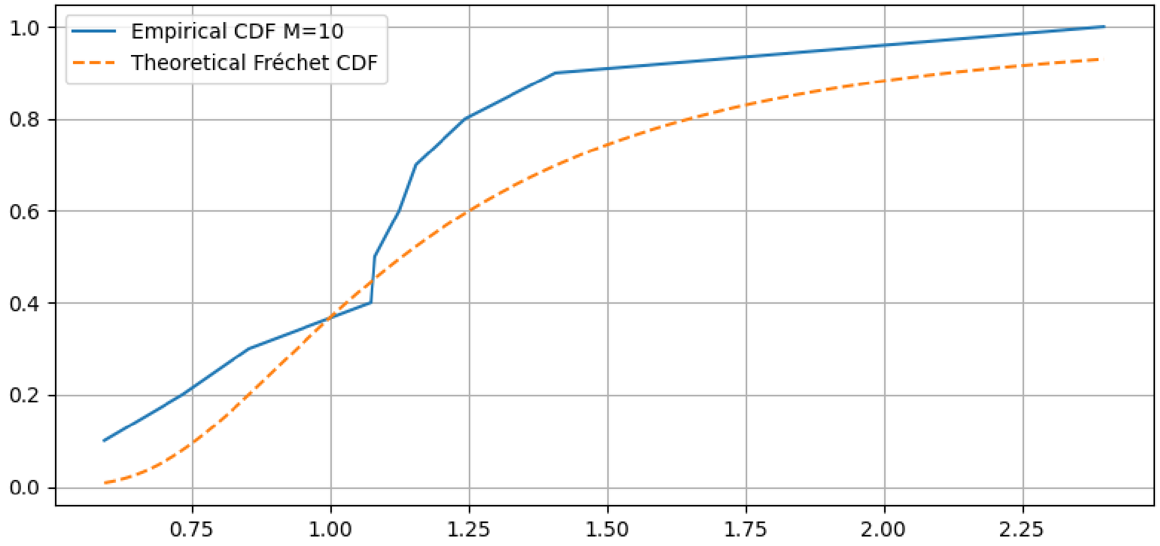

M = 300
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
Empirical CDF M=300
Theoretical Fréchet CDF


M = 1000
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14
Empirical CDF M=1000
Theoretical Fréchet CDF

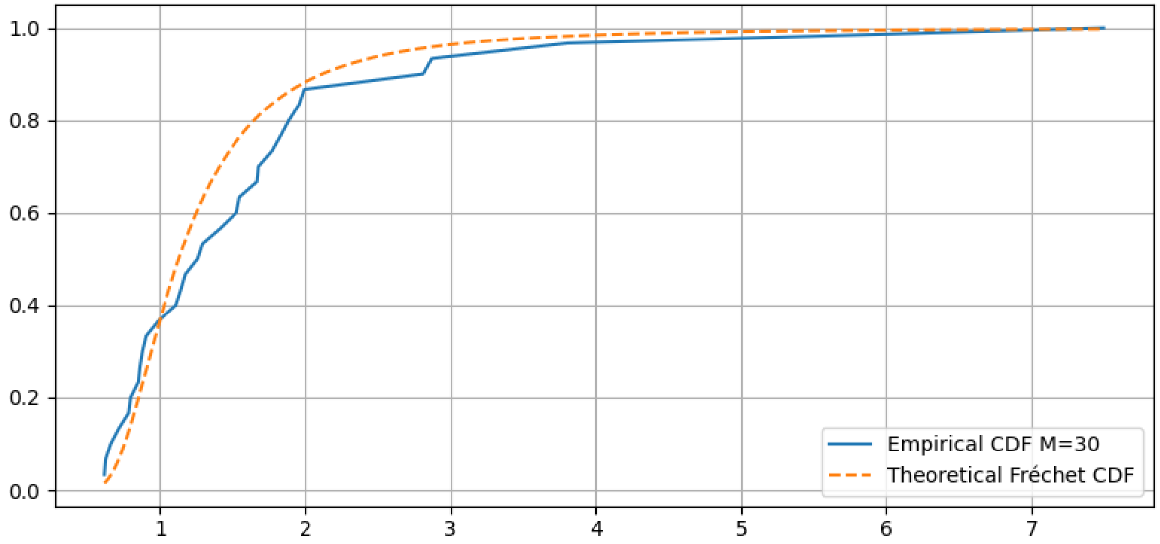
M = 3



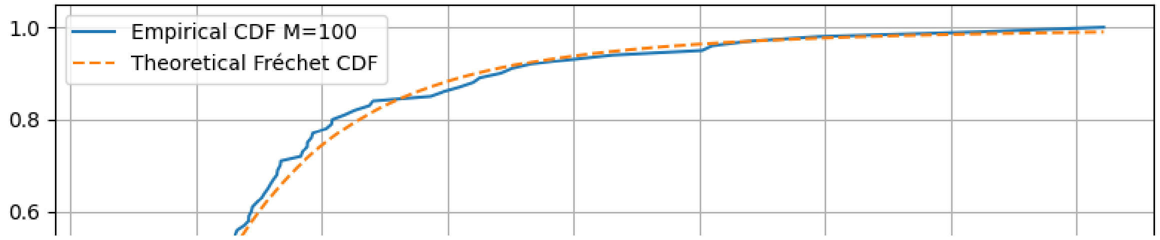
M = 10

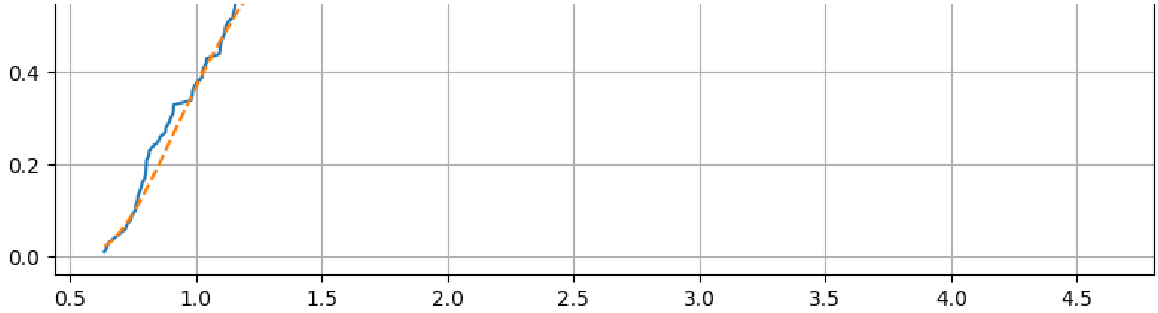


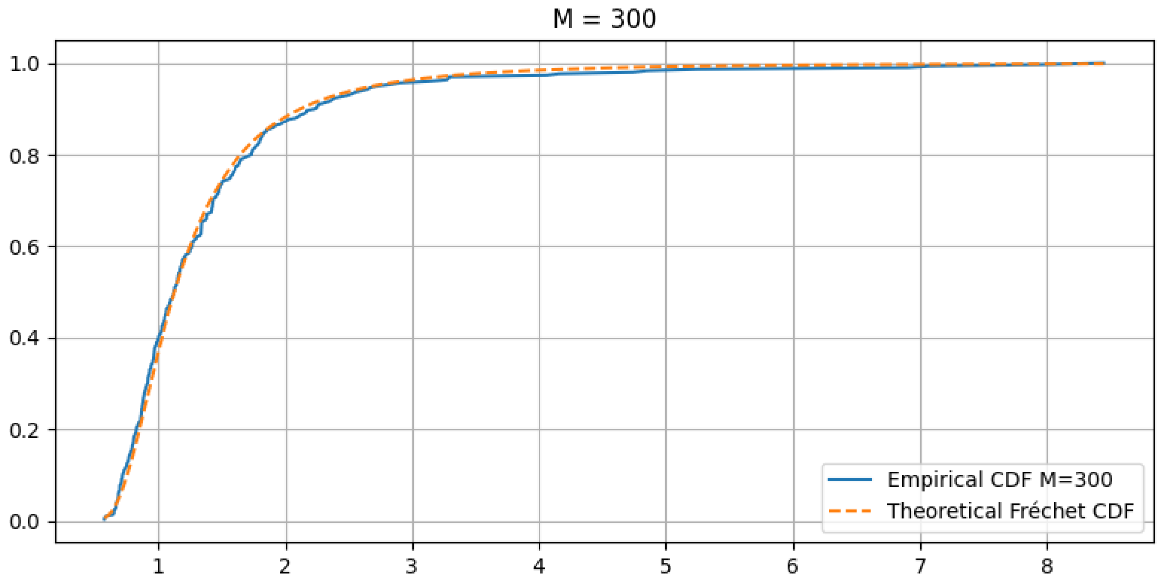
M = 30

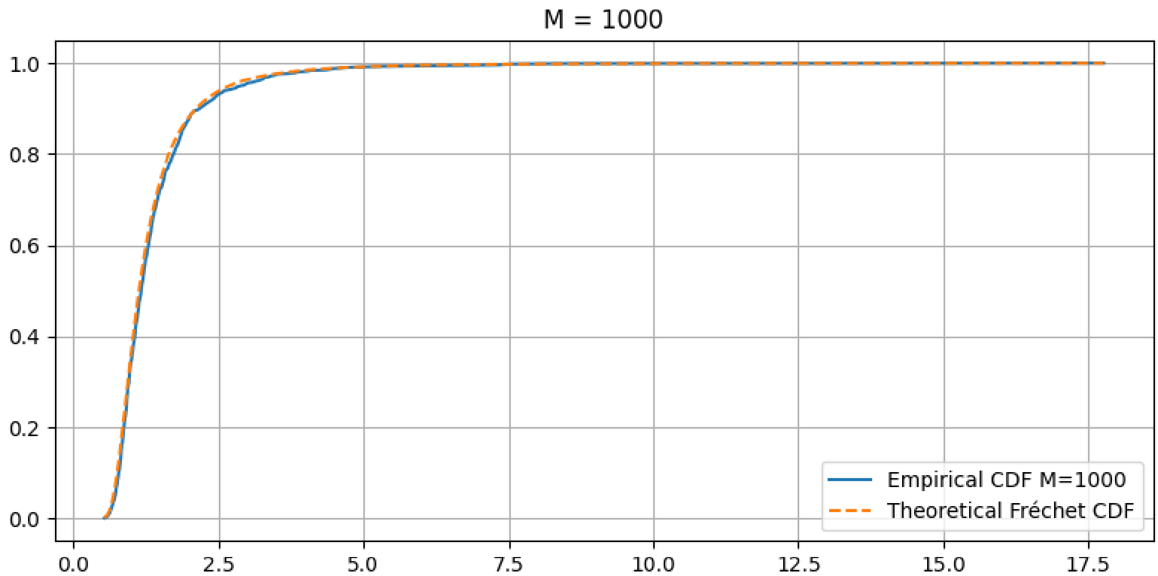


M = 100





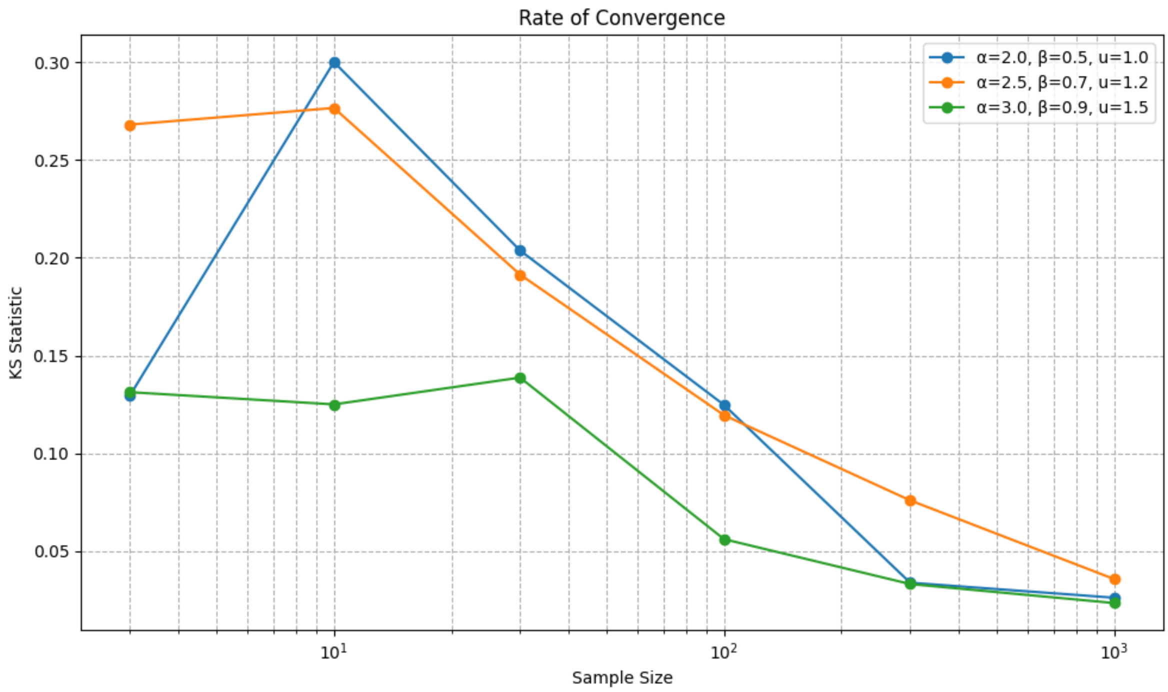




Now let's see the differnece between the rates of convergence

In [ ]:
```
import matplotlib.pyplot as plt

def ks_statistic(empirical_x, empirical_y, theoretical_cdf, alpha):
    """Compute the Kolmogorov-Smirnov statistic."""
    theoretical_y = [theoretical_cdf(x, alpha) for x in empirical_x]
    return np.max(np.abs(empirical_y - theoretical_y))

def plot_ks_statistics(M_values, parameter_sets):
    """Plot the KS statistics for different sample sizes and parameter se
    plt.figure(figsize=(10, 6))

    for params in parameter_sets:
```

```
        alpha, beta, u = params["alpha"], params["beta"], params["u"]
        ks_stats = []

        for M in M_values:
            blocks = generate_blocks(M, n, alpha, beta, u)
            maxima = compute_maxima(blocks)
            normalized_maxima = normalize_maxima(maxima, n, alpha, u, bet
            x_empirical, y_empirical = empirical_cdf(normalized_maxima)

            ks = ks_statistic(x_empirical, y_empirical, frechet_cdf, alph
            ks_stats.append(ks)

        plt.plot(M_values, ks_stats, '-o', label=f'α={alpha}, β={beta}, u

    plt.xscale("log")
    plt.title('Rate of Convergence')
    plt.xlabel('Sample Size')
    plt.ylabel('KS Statistic')
    plt.legend()
    plt.grid(True, which="both", ls="--")
    plt.tight_layout()
    plt.show()

plot_ks_statistics(M_values, parameter_sets)
```



4. Estimate the GEV Parameters

With the distribution from question 1 or 3, generate a dataset of an appropriate size and estimate the GEV parameters with a maximum-likelihood method. Discuss the convergence towards the true parameters.

Let's pick up the pareto distribution from question 3 and generate a dataset of size 10000.

In [ ]:
```
M = 10000
alpha_true = 2.0
```

```
beta_true = 0.5
u_true = 1.0

blocks = generate_blocks(M, n, alpha_true, beta_true, u_true)
maxima = compute_maxima(blocks)

from scipy.stats import genextreme as gev
from scipy.optimize import minimize

# Negative log-likelihood for the GEV distribution
def negative_gev_log_likelihood(params, data):
    c, loc, scale = params
    return -np.sum(gev.logpdf(data, c, loc=loc, scale=scale))

# Maximum-likelihood estimation
initial_guess = [0.5, np.mean(maxima), np.std(maxima)]
res = minimize(negative_gev_log_likelihood, initial_guess, args=(maxima,)
c_hat, loc_hat, scale_hat = res.x

print(f"Estimated parameters: c = {c_hat:.4f}, loc = {loc_hat:.4f}, scale
```

```
Estimated parameters: c = 0.5000, loc = 10.4209, scale = 15.9310
```

```
/Users/wadimoughanim/Library/Python/3.9/lib/python/site-packages/scipy/opt
imize/_numdiff.py:576: RuntimeWarning:

invalid value encountered in subtract
```

We obtain a shape value of 0.5000, a location (which is a bit like an average) of 10.3723 and a scale (which indicates how scattered our data are) of 15.1434.