4.4 Student Confidence

The central limit theorem tells us that, for large sample size N, the distribution of the means is Gaussian. However, for small sample size, the Gaussian isn't as good of an estimate. **Student's t-distribution** is superior for lower sample size and equivalent at higher sample size. Technically, if the population standard deviation σ_X is known, even for low sample size we should use the Gaussian distribution. However, this rarely arises in practice, so we can usually get away with an "always t" approach.

A way that the t-distribution accounts for low-N is by having an entirely different distribution for each N (seems a bit of a cheat, to me). Actually, instead of N, it uses the **degrees of freedom** ν, which is N minus the number of parameters required to compute the statistic. Since the standard deviation requires only the mean, for most of our cases, $\nu = N - 1$.

As with the Gaussian distribution, the t-distribution's integral is difficult to calculate. Typically, we will use a t-table, such as the one given in appendix A.2. There are three points of note.

1. Since we are primarily concerned with going from probability / confidence values (e.g. $P\%$ probability / confidence) to intervals, typically there is a column for each probability.
2. The extra parameter ν takes over one of the dimensions of the table because three-dimensional tables are illegal.
3. Many of these tables are "two-sided," meaning their t-scores and probabilities assume you want the symmetric probability about the mean over the interval $[-t_b, t_b]$, where t_b is your t-score bound.

Consider the following example.

Example 4.4

Write a Python script to generate a data set with 200 samples and sample sizes $N \in \{10, 20, 100\}$ using any old distribution. Compare the distribution of the means for the different N. Use the sample distributions and a t-table to compute 99% confidence intervals.

We proceed in Python. First, load packages:

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

Generate the data set.

```
confidence = 0.99  # Requirement
M = 200  # Number of samples
N_a = [10, 20, 100]  # Sample sizes
mu = 27  # Population mean
sigma = 9  # Population standard deviation
np.random.seed(1)  # Seed random number generator
data_a = mu + sigma * np.random.randn(N_a[-1], M)  # Generate normal data
print(data_a[:10, :5])  # Check 10 rows and five columns
```

```
[[41.61910827 21.49419228 22.24645423 17.3432824  34.78866866]
 [23.39209627 34.41605057 21.93925112 44.59390268 15.012435  ]
 [15.24119334 27.68742432 30.30508632 38.09609273 23.19428735]
 [17.34332149 31.4564275  18.43144109 22.33669003 13.84736756]
 [34.32908816 34.02422937 13.82351784 25.60957926 26.16810913]
 [25.62087454  5.10742339 31.57185903 24.08370904 13.40031053]
 [22.14870678 32.79689989 28.65270217 26.22215814 25.07410997]
 [28.70278877 38.01542408 24.29162355 29.26178669 35.35461193]
 [29.61904088 36.67409774 20.7197109  21.79506855 19.37292716]
 [15.22825791 40.25156676 27.67388488 10.91758137 28.48689528]]
```

Compute the means for different sample sizes.

```
mu_a = np.full((len(N_a), M), np.nan)
for i in range(len(N_a)):
    mu_a[i, :] = np.mean(data_a[:N_a[i], :M], axis=0)
```

Plot a histogram of the distribution of the means.

```
fig, ax = plt.subplots()
for i in range(len(N_a)):
    plt.hist(mu_a[i, :], bins=30, alpha=0.5, label=f'Sample size {N_a[i]}')
plt.xlabel('Mean Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

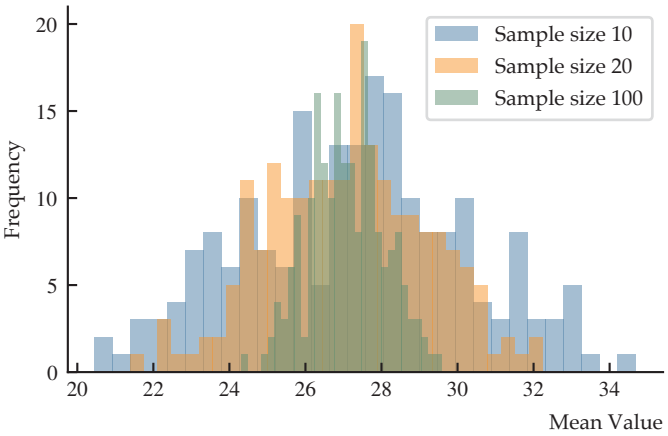

Figure 4.12. Histogram of the distribution of the means for different sample sizes.

It makes sense that the larger the sample size, the smaller the spread. A quantitative metric for the spread is, of course, the standard deviation of the means for each sample size.

```
S_mu = np.std(mu_a, axis=1, ddof=0)
print(S_mu)
```

```
[2.92548459 2.08250569 0.97864856]
```

Look up t-table values or use scipy to compute the t-value for different sample sizes and 99 percent confidence. Use these, the mean of means, and the standard deviation of means to compute the 99 percent confidence interval for each N.

```
from scipy.stats import t
t_a = t.ppf(confidence, np.array(N_a) - 1)  # t-value for confidence
for i in range(len(N_a)):
    interval = np.mean(mu_a[i, :]) + np.array([-1, 1]) * t_a[i] * S_mu[i]
    print(f'interval for N = {N_a[i]}: {interval}')
```

```
interval for N = 10: [19.04354748 35.55169379]
interval for N = 20: [21.81853996 32.39551634]
interval for N = 100: [24.77231819 29.40055444]
```

As expected, the larger the sample size, the smaller the interval over which we have 99 percent confidence in the estimate.

4.5 Regression

2L

Suppose we have a sample with two measurands: (1) the force F through a spring and (2) its displacement X (not from equilibrium). We would like to determine an analytic function that relates the variables, perhaps for prediction of the force given another displacement.

There is some variation in the measurement. Let's say the following is the sample.

```
X_a = 1e-3 * np.array(
    [10, 21, 30, 41, 49, 50, 61, 71, 80, 92, 100]
)  # m
F_a = np.array(
    [50.1, 50.4, 53.2, 55.9, 57.2, 59.9, 61.0, 63.9, 67.0, 67.9, 70.3]
)  # N
```

Let's take a look at the data.

```
fig, ax = plt.subplots()
p = ax.plot(X_a * 1e3, F_a, '.b', markersize=15)
ax.set_xlabel(r'$X$ (mm)')
ax.set_ylabel(r'$F$ (N)')
ax.set_xlim([0, np.max(X_a * 1e3)])
ax.grid(True)
plt.draw()
```

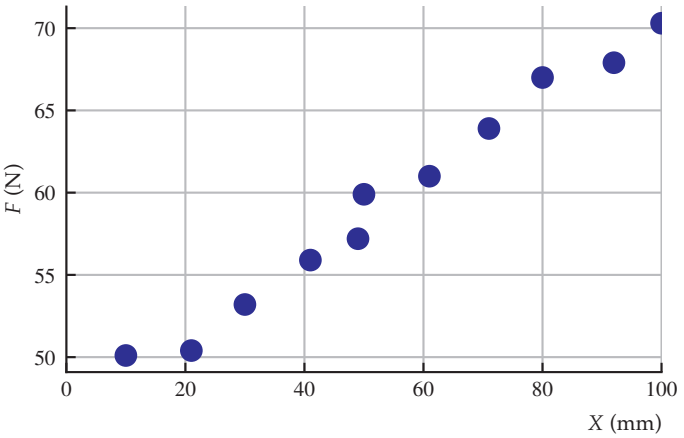

Figure 4.13. Force F as a function of displacement X.

How might we find an analytic function that agrees with the data? Broadly, our strategy will be to assume a general form of a function and use the data to set