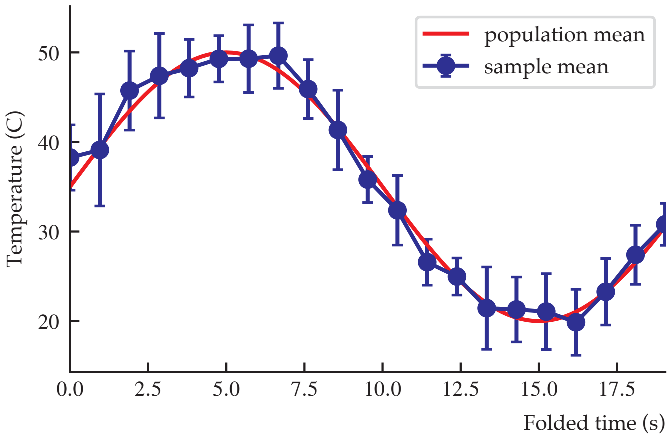

Figure 4.6. Sample means over a single period with error bars.

4.3 Confidence

One really ought to have it to give a lecture named it, but we'll give it a try anyway. **Confidence** is used in the common sense, although we do endow it with a mathematical definition to scare business majors, who aren't actually impressed, but indifferent. Approximately: if, under some reasonable assumptions (probabilistic model), we estimate the probability of some event to be $P\%$, we say we have $P\%$ confidence in it. I mean, business majors are all, "Supply and demand? Let's call that a 'law,' " so I think we're even.

So we're back to computing probability from distributions—probability density functions (PDFs) and probability mass functions (PMFs). Usually we care most about estimating the mean of our distribution. Recall from the previous lecture that when several samples are taken, each with its own mean, the mean is itself a random variable—with a mean, of course. Meanception.

But the mean has a probability distribution of its own. The **central limit theorem** has as one of its implications that, as the sample size N gets large, *regardless of the sample distributions, this distribution of means approaches the Gaussian distribution*.

But sometimes I always worry I'm being lied to, so let's check.

4.3.1 Checking the Central Limit Theorem

We proceed in Python. First, load packages:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
```

Generate Data Generate some data to test the central limit theorem {#generate-some-data-to-test-the-central-limit-theorem h="3y"}

Data can be generated by constructing an array using a (seeded for consistency) random number generator. Let's use a uniformly distributed PDF between 0 and 1.

```
N = 150  # Sample size (number of measurements per sample)
M = 120  # Number of samples
n = N * M  # Total number of measurements
mu_pop = 0.5  # Because it's a uniform PDF between 0 and 1
np.random.seed(11)  # Seed the random number generator
signal_a = np.random.rand(N, M)  # Uniform PDF
#
# Let's take a look at the data by plotting the first ten samples
# (columns) versus index, as shown in the figure below
```

```
samples_to_plot = 10
fig, ax = plt.subplots()
for j in range(samples_to_plot):
    ax.plot(signal_a[:, j], 'o-', markersize=3)
plt.xlabel('index')
plt.ylabel('measurement')
plt.draw()
```

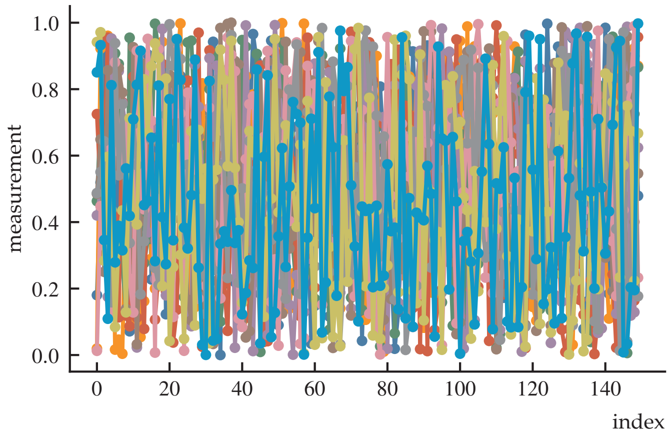

Figure 4.7. Raw data with colors corresponding to samples.

This is something like what we might see for continuous measurement data. Now make a histogram of each sample:

```
c = plt.cm.jet(np.linspace(0, 1, samples_to_plot))  # Color array
fig, ax = plt.subplots()
for j in range(samples_to_plot):
    plt.hist(signal_a[:, j],
             bins=30,  # Number of bins
             color=c[j],
             alpha=0.3,
             density=True)  # For PMF
plt.xlim([-0.05, 1.05])
plt.xlabel('Measurement')
plt.ylabel('Probability')
plt.draw()
```

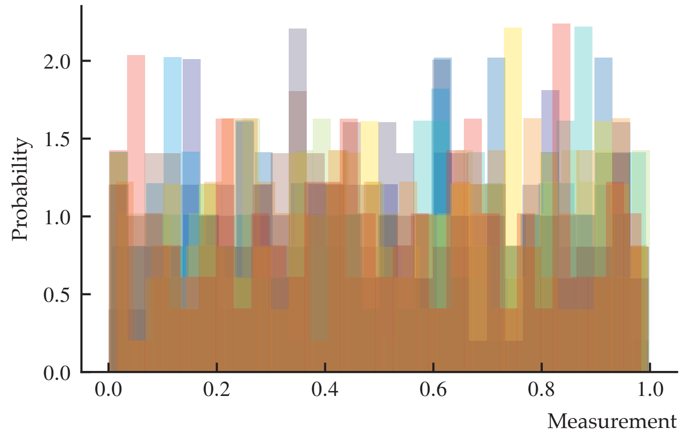

Figure 4.8. Histograms of the approximately uniform distribution of each sample (color).

This isn't a great plot, but it shows roughly that each sample is fairly uniformly distributed.

Sample Statistics Now let's check out the sample statistics. We want the sample mean and standard deviation of each column. Let's use the built-in functions `mean` and `std`.

```
mu_a = np.mean(signal_a, axis=0)  # Mean of each column
s_a = np.std(signal_a, axis=0)  # Standard deviation of each column
```

Now we can compute the mean statistics, both the mean of the mean $\overline{\overline{X}}$ and the standard deviation of the mean $s_{\overline{X}}$, which we don't strictly need for this part, but we're curious. We choose to use the direct estimate instead of the $s_X/\sqrt{N}$ formula, but they should be close.

```
mu_mu = np.mean(mu_a)
s_mu = np.std(mu_a)
```

The Truth about Sample Means It's the moment of truth. Let's plot the histogram of the sample means as follows:

```python
fig, ax = plt.subplots()
plt.hist(mu_a,
         bins=30,  # You can adjust the number of bins as needed
         density=True)  # For PMF
plt.xlabel('Measurement')
plt.ylabel('Probability')
plt.draw()
```

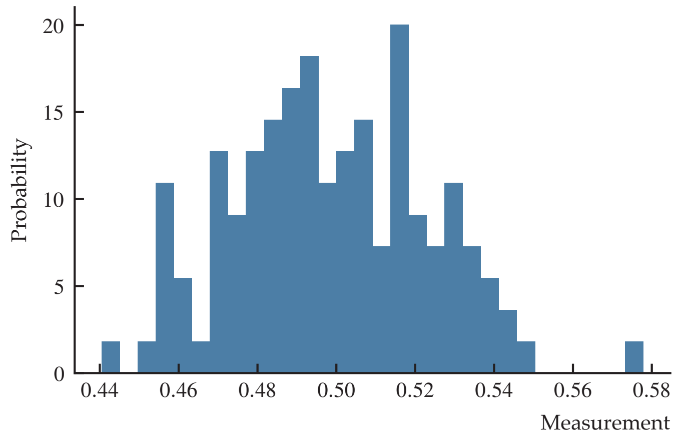

Figure 4.9. Histogram of the approximately normal distribution of the means.

This looks like a Gaussian distribution about the mean of means, so I guess the central limit theorem is legit.

Gaussian and Probability We already know how to compute the probability P a value of a random variable X lies in a certain interval from a PMF or PDF (the sum or the integral, respectively). This means that, for sufficiently large sample size N such that we can assume from the central limit theorem that the sample means $\overline{x_i}$ are normally distributed, *the probability a sample mean value $\overline{x_i}$ is in a certain interval* is given by integrating the Gaussian PDF. The Gaussian PDF for random variable Y representing the sample means is

$$f(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-(y-\mu)^2}{2\sigma^2}.$$

where μ is the population mean and σ is the population standard deviation.

The integral of f over some interval is the probability a value will be in that interval. Unfortunately, that integral is uncool. It gives rise to the definition of the *error function*, which, for the Gaussian random variable Y, is

$$\text{erf}(y_b) = \frac{1}{\sqrt{\pi}} \int_{-y_b}^{y_b} e^{-t^2} dt.$$

This expresses the probability a sample mean being in the interval $[-y_b, y_b]$ if Y has mean 0 and variance $1/2$.

Python has the error function in the `scipy.special` package. Let's plot the error function:

```python
from scipy.special import erf
y_a = np.linspace(0, 3, 100)
fig, ax = plt.subplots()
ax.plot(y_a, erf(y_a), linewidth=2)
plt.grid(True)
plt.xlabel(r'Interval bound $y_b$')
plt.ylabel(r'Error function $\text{erf}(y_b)$')
plt.draw()
```

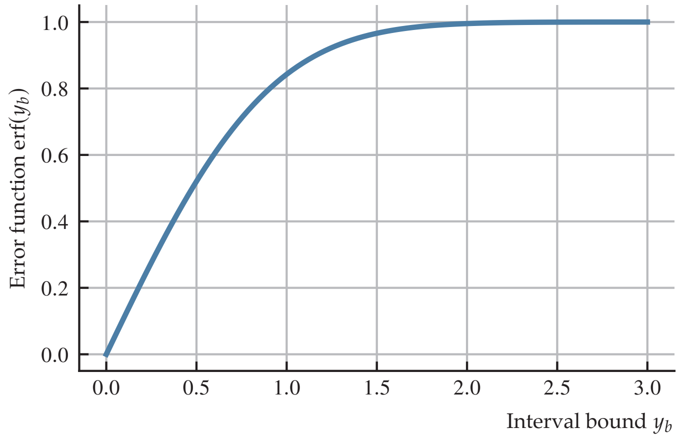

Figure 4.10. Error function for Gaussian random variable.

We could deal directly with the error function, but most people don't and we're weird enough, as it is. Instead, people use the **Gaussian cumulative distribution function** (CDF) $\Phi : \mathbb{R} \to \mathbb{R}$, which is defined as

$$\Phi(z) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{z}{\sqrt{2}}\right)\right)$$

and which expresses the probability of a Gaussian random variable Z with mean 0 and standard deviation 1 taking on a value in the interval $(-\infty, z]$. The Gaussian CDF and PDF are plotted below.

```python
from scipy.stats import norm
z_a = np.linspace(-3, 3, 300)
threshold = 1.5
a_pdf = lambda z: (z < threshold) * norm.pdf(z, 0, 1)
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 12))  # Two subplots
ax1.fill_between(z_a, a_pdf(z_a), color=[.8, .8, .8])
ax1.plot(z_a, norm.pdf(z_a, 0, 1), linewidth=2)
ax1.grid(True)
ax1.set_xlabel(r'$z$')
ax1.set_ylabel(r'Gaussian PDF $f(z)$')
ax1.text(1.5, norm.pdf(1.5, 0, 1) + .01, r'$z_b$')
ax1.legend([r'$\Phi(z_b)$', r'$f(z)$'])
ax2.plot(z_a, 1/2 * (1 + erf(z_a / np.sqrt(2))), linewidth=2)
ax2.grid(True)
ax2.set_xlabel(r'interval bound $z_b$')
ax2.set_ylabel(r'Gaussian CDF $\Phi(z_b)$')
plt.show()
```

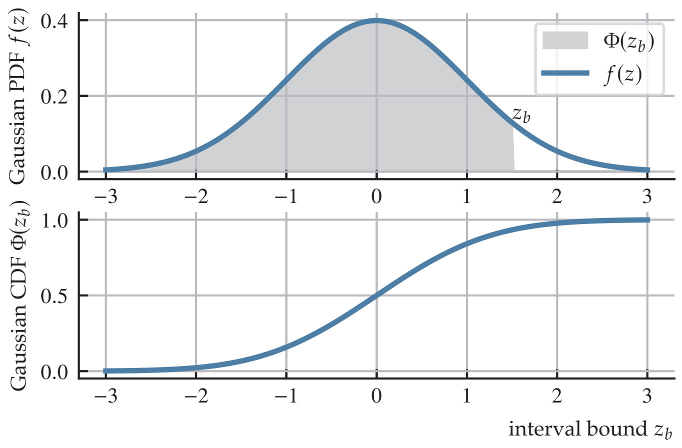

Figure 4.11. Gaussian PDF and CDF for z-scores.

Values can be taken directly from the graph, but it's more accurate to use the table of values in appendix A.1.

That's great and all, but occasionally (always) we have Gaussian random variables with nonzero means and nonunity standard deviations. It turns out we can shift any Gaussian random variable by its mean and scale it by its standard deviation to make it have zero mean and standard deviation. We can then use Φ and interpret the results as being relative to the mean and standard deviation, using phrases like "the probability it is within two standard deviations of its mean." The transformed random variable Z and its values z are sometimes called the **z-score**. For a particular value x of a random variable X, we can compute its z-score (or value z of random variable Z) with the formula

$$z = \frac{x - \mu_X}{\sigma_X}$$

and compute the probability of X taking on a value within the interval, say, $x \in [x_{b-}, x_{b+}]$ from the table. (Sample statistics $\overline{X}$ and S_X are appropriate when population statistics are unknown.)

For instance, compute the probability a Gaussian random variable X with $\mu_X = 5$ and $\sigma_X = 2.34$ takes on a value within the interval $x \in [3, 6]$.

1. Compute the z-score of each endpoint of the interval:

$$z_3 = \frac{3 - \mu_X}{\sigma_X} \approx -0.85$$
$$z_6 = \frac{6 - \mu_X}{\sigma_X} \approx 0.43.$$

2. Look up the CDF values for z_3 and z_6, which are $\Phi(z_3) = 0.1977$ and $\Phi(z_6) = 0.6664$. 3. The CDF values correspond to the probabilities $x < 3$ and $x < 6$. Therefore, to find the probability x lies in that interval, we subtract the lower bound probability:

$$\begin{aligned} P(x \in [3, 6]) &= P(x < 6) - P(x < 3) \\ &= \Phi(6) - \Phi(3) \\ &\approx 0.6664 - 0.1977 \\ &\approx 0.4689. \end{aligned}$$

So there is a 46.89 percent probability, and therefore we have 46.89 percent confidence, that $x \in [3, 6]$.

Often we want to go the other way, estimating the symmetric interval $[x_{b-}, x_{b+}]$ for which there is a given probability. In this case, we first look up the z-score corresponding to a certain probability. For concreteness, given the same population statistics above, let's find the symmetric interval $[x_{b-}, x_{b+}]$ over which we have 90 percent confidence. From the table, we want two, symmetric z-scores that have CDF-value difference 0.9. Or, in maths,

$$\Phi(z_{b+}) - \Phi(z_{b-}) = 0.9 \quad \text{and} \quad z_{b+} = -z_{b-}.$$

Due to the latter relation and the additional fact that the Gaussian CDF has antisymmetry,

$$\Phi(z_{b+}) + \Phi(z_{b-}) = 1.$$

Adding the two Φ equations, we get

$$\begin{aligned}\Phi(z_{b+}) &= 1.9/2 \\ &= 0.95\end{aligned}$$

and $\Phi(z_{b-}) = 0.05$. From the table, these correspond (with a linear interpolation) to $z_b = z_{b+} = -z_{b-} \approx 1.645$. All that remains is to solve the z-score formula for x:

$$x = \mu_X + z\sigma_X.$$

From this,

$$\begin{aligned}x_{b+} &= \mu_X + z_{b+}\sigma_X \approx 8.849 \\ x_{b-} &= \mu_X + z_{b-}\sigma_X \approx 1.151.\end{aligned}$$

and X has a 90 percent confidence interval $[1.151, 8.849]$.

Example 4.3

Consider the data set generated above. What is our 95% confidence interval in our estimate of the mean?

Assuming we have a sufficiently large data set, the distribution of means is approximately Gaussian. Following the same logic as above, we need z-score that gives an upper CDF value of $(1 + 0.95)/2 = 0.975$. From the table, we obtain the $z_b = z_{b+} = -z_{b-}$, below.

```
z_b = 1.96
```

Now we can estimate the mean using our sample and mean statistics,

$$\overline{X} = \overline{\overline{X}} \pm z_b S_{\overline{X}}. \tag{4.1}$$

```
mu_x_95 = mu_mu + np.array([-z_b,z_b])*s_mu
```

```
[0.4526     0.5449]
```

This is our 95 percent confidence interval in our estimate of the mean.

4.4 Student Confidence

IC

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:

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

Generate the data set.