Probability and Statistics

FRE6871 & FRE7241, Fall 2024

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

September 4, 2024

Pseudo-Random Numbers

Pseudo-random numbers are deterministic sequences of numbers which have some of the properties of random numbers, but they are not truly random numbers.

Pseudo-random number generators depend on a *seed* value, and produce the same sequence of numbers for a given *seed* value.

The function `set.seed()` initializes the random number generator by specifying the *seed* value.

The choice of *seed* value isn't important, and a given value is just good as any other one.

The function `runif()` produces random numbers from the *uniform* distribution.

The function `rnorm()` produces random numbers from the *normal* distribution.

The function `rt()` produces random numbers from the *t-distribution* with *df* degrees of freedom.

```
> # Initialize the random number generator
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> runif(3)  # three numbers from uniform distribution
> runif(3)  # Simulate another three numbers
> # Initialize the random number generator
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> runif(3)  # Simulate another three numbers
> # Simulate random number from standard normal distribution
> rnorm(1)
> # Simulate five standard normal random numbers
> rnorm(5)
> # Simulate five non-standard normal random numbers
> rnorm(n=5, mean=1, sd=2)  # Match arguments by name
> # Simulate t-distribution with 2 degrees of freedom
> rt(n=5, df=2)
```

The Logistic Map

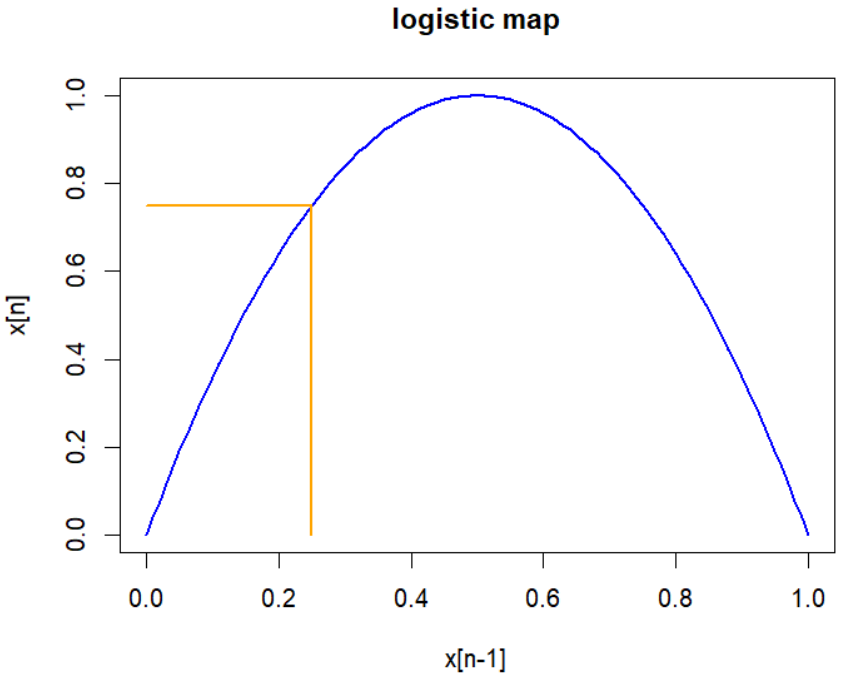
The *logistic map* is a recurrence relation which produces a deterministic sequence of numbers:

$$x_n = rx_{n-1}(1 - x_{n-1})$$

If the *seed* value x_0 is in the interval $(0, 1)$ and if $r = 4$, then the sequence x_n is also contained in the interval $(0, 1)$.

The function `curve()` plots a function defined by its name.

```
> # Define logistic map function
> log_map <- function(x, r=4) r*x*(1-x)
> log_map(0.25, 4)
> # Plot logistic map
> x11(width=6, height=5)
> curve(expr=log_map, type="l", xlim=c(0, 1),
+ xlab="x[n-1]", ylab="x[n]", lwd=2, col="blue",
+ main="logistic map")
> lines(x=c(0, 0.25), y=c(0.75, 0.75), lwd=2, col="orange")
> lines(x=c(0.25, 0.25), y=c(0, 0.75), lwd=2, col="orange")
```



Generating Pseudo-Random Numbers Using Logistic Map

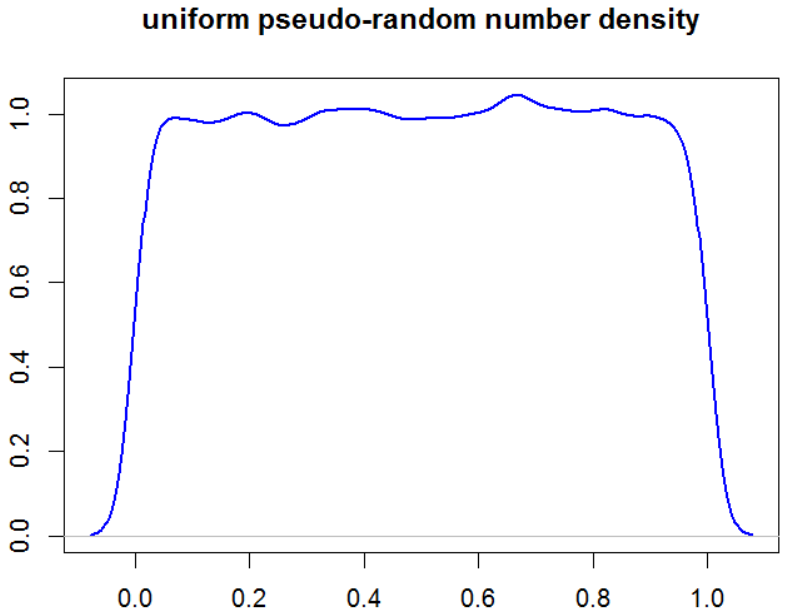
The *logistic map* can be used to calculate sequences of pseudo-random numbers.

For most *seed* values x_0 and $r = 4$, the *logistic map* produces a pseudo-random sequence, but it's not uniformly distributed.

The inverse cosine function `acos()` transforms a *logistic map* sequence into a uniformly distributed sequence,

$$u_n = \arccos(1 - 2x_n)/\pi$$

```
> # Calculate uniformly distributed pseudo-random sequence
> # using logistic map function.
> unifun <- function(seedv, n=10) {
+   # Pre-allocate vector instead of "growing" it
+   datav <- numeric(n)
+   # initialize
+   datav[1] <- seedv
+   # Perform loop
+   for (i in 2:n) {
+     datav[i] <- 4*datav[i-1]*(1-datav[i-1])
+   }  # end for
+   acos(1-2*datav)/pi
+ }  # end unifun
```




```
> unifun(seedv=0.1, n=15)
> plot(
+   density(unifun(seedv=runif(1), n=1e5)),
+   xlab="", ylab="", lwd=2, col="blue",
+   main="uniform pseudo-random number density")
```

Generating Binomial Random Numbers

A *binomial* trial is a coin flip, that results in either a success or failure.

The *binomial* distribution specifies the probability of obtaining a certain number of successes in a sequence of independent *binomial* trials.

Let p be the probability of obtaining a success in a *binomial* trial, and let $(1-p)$ be the probability of failure.

$p = 0.5$ corresponds to flipping an unbiased coin.

The probability of obtaining k successes in n independent *binomial* trials is equal to:

$$\binom{n}{k} p^k (1-p)^{(n-k)}$$

The function `rbinom()` produces random numbers from the *binomial* distribution.

```
> # Initialize the random number generator
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> # Flip unbiased coin once, 20 times
> rbinom(n=20, size=1, 0.5)
> # Number of heads after flipping twice, 20 times
> rbinom(n=20, size=2, 0.5)
> # Number of heads after flipping thrice, 20 times
> rbinom(n=20, size=3, 0.5)
> # Number of heads after flipping biased coin thrice, 20 times
> rbinom(n=20, size=3, 0.8)
> # Number of heads after flipping biased coin thrice, 20 times
> rbinom(n=20, size=3, 0.2)
> # Flip unbiased coin once, 20 times
> sample(x=0:1, size=20, replace=TRUE)  # Fast
> as.numeric(runif(20) < 0.5)  # Slower
```

Generating Random Samples and Permutations

A *sample* is a subset of elements taken from a set of data elements.

The function `sample()` selects a random sample from a vector of data elements.

By default the *size* of the sample (the `size` argument) is equal to the number of elements in the data vector.

So the call `sample(da_ta)` produces a random permutation of all the elements of `da_ta`.

The function `sample()` with `replace=TRUE` selects samples with replacement (the default is `replace=FALSE`).

Monte Carlo simulation consists of generating random samples from a given probability distribution.

The *Monte Carlo* data samples can then used to calculate different parameters of the probability distribution (moments, quantiles, etc.), and its functionals.

```
> # Permutation of five numbers
> sample(x=5)
> # Permutation of four strings
> sample(x=c("apple", "grape", "orange", "peach"))
> # Sample of size three
> sample(x=5, size=3)
> # Sample with replacement
> sample(x=5, replace=TRUE)
> sample(  # Sample of strings
+   x=c("apple", "grape", "orange", "peach"),
+   size=12,
+   replace=TRUE)
> # Binomial sample: flip coin once, 20 times
> sample(x=0:1, size=20, replace=TRUE)
> # Flip unbiased coin once, 20 times
> as.numeric(runif(20) > 0.5)  # Slower
```

Statistical Estimators

A data *sample* is a set of observations $\{x_1, \ldots, x_n\}$ of a *random variable* x.

Let x follow a probability distribution with population *mean* equal to μ and population *standard deviation* equal to σ.

A *statistic* is a function of the data *sample*: $f(x_1, \ldots, x_n)$, so it is itself a *random variable*.

A statistical *estimator* is a *statistic* that provides an estimate of a distribution *parameter*.

For example:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Is an *estimator* of the *population mean* of the *distribution*.

```
> # Initialize the random number generator
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> # Sample from Standard Normal Distribution
> datav <- rnorm(1000)
>
> mean(datav)  # Sample mean
[1] -0.0354
>
> median(datav)  # Sample median
[1] -0.0153
>
> sd(datav)  # Sample standard deviation
[1] 1.01
```

Estimators of Higher Moments

The estimators of the moments of a probability distribution, based on a *sample* of data x_i, are given by:

Sample mean: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

Sample variance: $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$

Their expected values are equal to the population mean and standard deviation:
$\mathbb{E}[\bar{x}] = \mu$ and $\mathbb{E}[\hat{\sigma}] = \sigma\mathbb{E}[\hat{\sigma}] = \sigma\mathbb{E}[\hat{\sigma}] = \sigma\mathbb{E}[\hat{\sigma}] = \sigma\mathbb{E}[\hat{\sigma}] = \sigma\mathbb{E}[\hat{\sigma}] = \sigma\mathbb{E}[\hat{\sigma}] = \sigma$

The third and fourth moments are equal to:

$$\mu 3 = \frac{n}{(n-1)(n-2)}\sum_{i=1}^{n}(x_i - \bar{x})^3$$

$$\mu 4 = \frac{n}{(n-1)^2}\sum_{i=1}^{n}(x_i - \bar{x})^4$$

The skewness and kurtosis are equal to the moments scaled by the standard deviation:

$$\varsigma = \frac{\mu 3}{\sigma^3}, \quad \kappa = \frac{\mu 4}{\sigma^4}$$

```
> # VTI returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> # Number of observations
> nrows <- NROW(retp)
> # Mean of VTI returns
> meanv <- mean(retp)
> # Standard deviation of VTI returns
> stdev <- sd(retp)
> # Standardize returns
> retp <- (retp - meanv)/stdev
> # Skewness and kurtosis of VTI returns
> nrows/((nrows-1)*(nrows-2))*sum(retp^3)
[1] -0.382
> nrows/(nrows-1)^2*sum(retp^4)
[1] 13.8
> # Random normal returns
> retp <- rnorm(nrows)
> # Mean and standard deviation of random normal returns
> mean(retp); sd(retp)
[1] -0.0194
[1] 0.975
> # Skewness and kurtosis of random normal returns
> nrows/((nrows-1)*(nrows-2))*sum(retp^3)
[1] -0.0446
> nrows/(nrows-1)^2*sum(retp^4)
[1] 2.74
```

The normal distribution has skewness equal to zero $\varsigma = 0$, and kurtosis equal to three $\kappa = 3$.

Estimators of Quantiles

The *quantile* corresponding to a given *probability* p, is the value of the *random variable* x, such that the probability of obtaining values less than x is equal to the *probability* p.

The *quantile* of a data sample can be calculated by first sorting the sample, and then finding the value corresponding closest to the given *probability* p.

The function `quantile()` calculates the sample quantiles. It uses interpolation to improve the accuracy. Information about the different interpolation methods can be found by typing `?quantile`.

The function `sort()` returns a vector sorted into ascending order.

The function `pnorm()` calculates the cumulative *normal* distribution, i.e. the cumulative probability for a given quantile value.

The function `qnorm()` calculates the inverse cumulative *normal* distribution, i.e. the quantile for a given probability value.

The function `dnorm()` calculates the normal probability *density*.

```
> # Calculate cumulative standard normal distribution
> c(pnorm(-2), pnorm(2))
> # Calculate inverse cumulative standard normal distribution
> c(qnorm(0.75), qnorm(0.25))
> # Initialize the random number generator
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> # Sample from Standard Normal Distribution
> nrows <- 1000
> datav <- rnorm(nrows)
> # Sample mean - MC estimate
> mean(datav)
> # Sample standard deviation - MC estimate
> sd(datav)
> # Monte Carlo estimate of cumulative probability
> c(pnorm(1), sum(datav < 1)/nrows)
> # Monte Carlo estimate of quantile
> confl <- 0.99
> qnorm(confl)
> cutoff <- confl*nrows
> datav <- sort(datav)
> c(datav[cutoff], quantile(datav, probs=confl))
> # Read the source code of quantile()
> stats:::quantile.default
> # microbenchmark quantile
> library(microbenchmark)
> summary(microbenchmark(
+   monte_carlo=datav[cutoff],
+   quantv=quantile(datav, probs=confl),
+   times=100))[, c(1, 4, 5)]  # end microbenchmark summary
```

Standard Errors of Estimators

Statistical estimators are functions of samples (which are random variables), and therefore are themselves *random variables*.

The *standard error* (SE) of an estimator is defined as its *standard deviation* (not to be confused with the *population standard deviation* of the underlying random variable).

For example, the *standard error* of the estimator of the mean is equal to:

$$\sigma_\mu = \frac{\sigma}{\sqrt{n}}$$

Where σ is the *population standard deviation* (which is usually unkown).

The *estimator* of this *standard error* is equal to:

$$SE_\mu = \frac{\hat{\sigma}}{\sqrt{n}}$$

where: $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$ is the sample standard deviation (the estimator of the population standard deviation).

```
> # Initialize the random number generator
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> # Sample from Standard Normal Distribution
> nrows <- 1000
> datav <- rnorm(nrows)
> # Sample mean
> mean(datav)
> # Sample standard deviation
> sd(datav)
> # Standard error of sample mean
> sd(datav)/sqrt(nrows)
```

The Characteristic Function

The *characteristic function* $\hat{f}(t)$ is equal to the *Fourier transform* of the *probability density function* $f(x)$:

$$\hat{f}(t) = \mathbb{E}[e^{itx}] = \int_{-\infty}^{\infty} f(x)\, e^{itx}\, \mathrm{d}x$$

The *normal* probability density function:

$$\phi(x) = \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}$$

Has the *characteristic function* $\hat{\phi}(t)$ equal to:

$$\hat{\phi}(t) = e^{i\mu t} e^{-(\sigma t)^2/2}$$

The *probability function* $f(x)$ is equal to the *inverse Fourier transform* of the *characteristic function* $\hat{f}(t)$:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(t)\, e^{-itx}\, \mathrm{d}t$$

The *characteristic function* of the first derivative of $f(x)$ is equal to $(-it)\hat{f}(t)$, the *characteristic function* multiplied by $(-it)$:

$$\frac{df(x)}{dx} = \frac{1}{2\pi} \int_{-\infty}^{\infty} -it\, \hat{f}(t)\, e^{-itx}\, \mathrm{d}t$$

The *characteristic function* of the n-th derivative of $f(x)$ is equal to $(-it)^n\hat{f}(t)$, the *characteristic function* multiplied by $(-it)^n$:

$$\frac{d^n f(x)}{dx^n} = \frac{1}{2\pi} \int_{-\infty}^{\infty} (-it)^n\, \hat{f}(t)\, e^{-itx}\, \mathrm{d}t$$

The Moment Generating Function

The *moment generating function* $M_X(t)$ of a random variable x with the probability density function $f(x)$ is equal to:

$$M_X(t) = \mathbb{E}[e^{tx}] = \int_{-\infty}^{\infty} f(x)\, e^{tx}\, \mathrm{d}x$$

The n-th derivative of $M_X(t)$ with respect to t, at $t = 0$ is equal to the n-th *moment* μ_n:

$$\mu_n = \frac{d^n M_X(t)}{dt^n}|_{t=0} = \frac{d^n \mathbb{E}[e^{tx}]}{dt^n}|_{t=0}$$
$$= \mathbb{E}[x^n e^{tx}]|_{t=0} = \mathbb{E}[x^n]$$

The *moments* μ_n are related to the *central moments* $\mathbb{E}[(x-\mu)^n]$ but they are not equal to them.

The *moment generating function* can be expressed as a series of its *moments*:

$$M_X(t) = \sum_{n=0}^{\infty} \frac{\mu_n t^n}{n!}$$

The moment generating function for the *normal* distribution is equal to:

$$M_X(t) = \exp(\mu t + \frac{1}{2}\sigma^2 t^2)$$

The *characteristic function* $\hat{f}(t)$ is equal to the *moment generating function* with a purely *imaginary* argument:

$$\hat{f}(t) = \mathbb{E}[e^{itx}] = M_X(it)$$

Cumulants of Probability Distributions

The *cumulant generating function* $K_X(t)$ is equal to the logarithm of the *moment generating function*:

$$K_X(t) = \log M_X(t)$$

The n-th derivative of $K_X(t)$ with respect to t, at $t = 0$ is equal to the n-th *cumulant* κ_n:

$$\kappa_n = \frac{d^n K_X(t)}{dt^n}|_{t=0}$$

The *cumulants* are related to the *moments* of a distribution: with the first three cumulants being equal to the *central moments* (mean, variance, and skewness), while the higher order *cumulants* are polynomials of the *moments*.

The *cumulant generating function* $K_X(t)$ can be expanded into a power series of the *cumulants*:

$$K_X(t) = \sum_{n=1}^{\infty} \frac{\kappa_n t^n}{n!} = \mu t + \sigma^2 \frac{t^2}{2} + \sigma^3 s^3 \frac{t^3}{6} + \dots$$

The cumulant generating function for the *normal* distribution is equal to:

$$K_X(t) = \mu t + \frac{1}{2}\sigma^2 t^2$$

So that its first two *cumulants* are equal to the *mean* μ and the *variance* σ^2, and the *cumulants* of order 3 and higher are all equal to zero.

The advantage of *cumulants* over the *moments* is that the *cumulants* of the sum of independent random variables are equal to the sum of their *cumulants*:

$$K_{(X+Y)}(t) = \log \mathbb{E}[e^{t(X+Y)}] = \log\left(\mathbb{E}[e^{tX}]\mathbb{E}[e^{tY}]\right)$$
$$= \log \mathbb{E}[e^{tX}] \log \mathbb{E}[e^{tY}] = K_X(t) + K_Y(t)$$

The Hermite Polynomials

The n-th derivative of the *standard normal* distribution $\phi(x)$ is given by Rodrigues' formula:

$$\frac{d^n\phi(x)}{dx^n} = \frac{d^n}{dx^n}\frac{e^{-x^2/2}}{\sqrt{2\pi}} = (-1)^n H_n(x)\phi(x)$$

Where H_n are the *Hermite polynomials*.

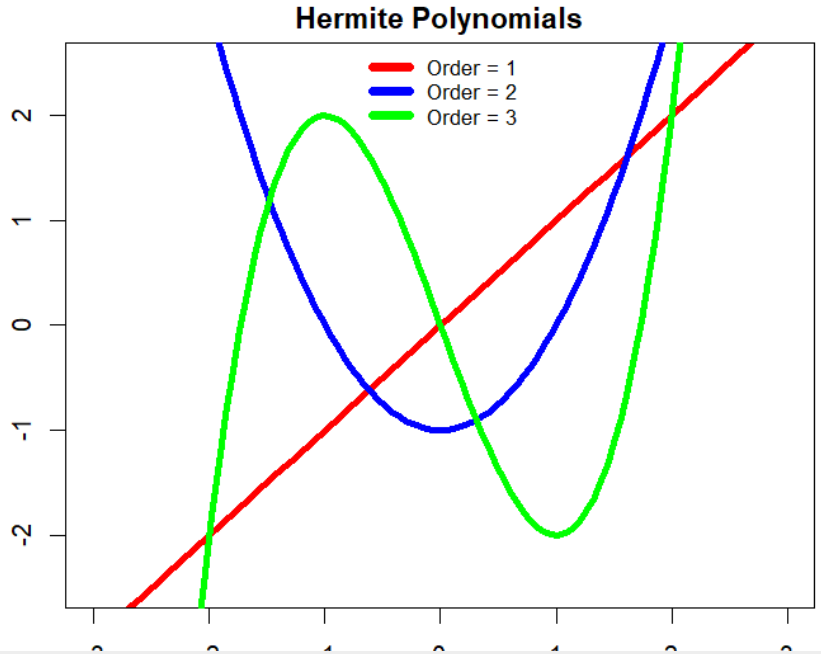
The first four *Hermite polynomials* are equal to:

$$H_0(x) = 1; H1(x) = x$$

$$H2(x) = x^2 - 1; H3(x) = x^3 - 3x$$

The even order polynomials are *symmetric* $H_{2n}(-x) = H_{2n}(x)$ while the odd order are *antisymmetric* $H_{2n-1}(-x) = -H_{2n-1}(x)$.

```
> # Define the Hermite polynomials
> hermitep <- function(x, n) {
+     switch(n+1, 1, x, (x^2 - 1), (x^3 - 3*x), 0)
+ }  # end hermitep
```




```
> colorv <- c("red", "blue", "green")
> for (indeks in 1:3) {  # Plot three curves
+   curve(expr=hermitep(x, indeks),
+   xlim=c(-3, 3), ylim=c(-2.5, 2.5),
+   xlab="", ylab="", lwd=4, col=colorv[indeks],
+   add=as.logical(indeks-1))
+ }  # end for

Error in hermitep(x, indeks):  could not find function "hermitep"

> # Add title and legend
> title(main="Hermite Polynomials", line=0.5)

Error in title(main = "Hermite Polynomials", line = 0.5):
plot.new has not been called yet

> labelv <- paste("Order", 1:3, sep=" = ")
```

The Hermite Functions

The *Hermite functions* $\psi_n(x)$ are equal to:

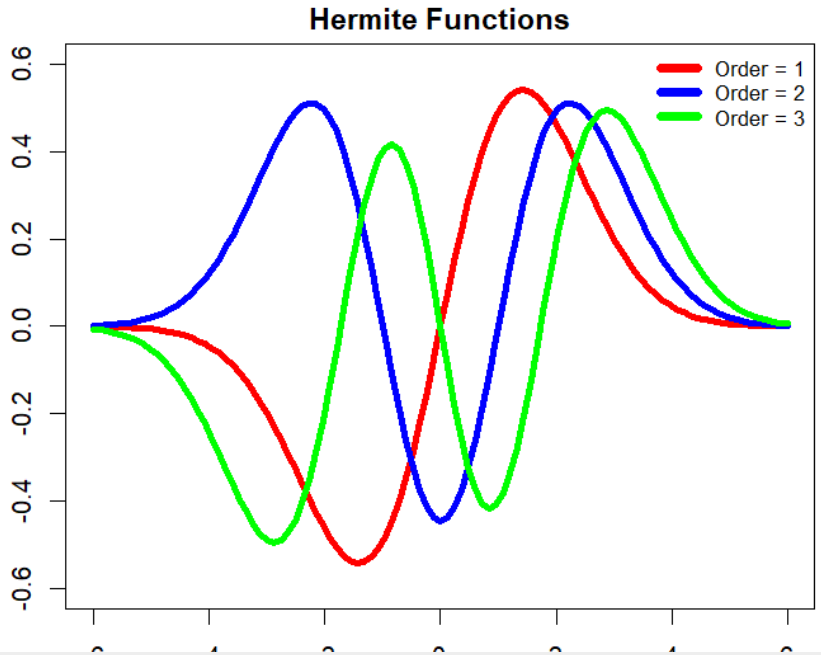
$$\psi_n(x) = \frac{1}{\sqrt{n!\sqrt{2\pi}}} e^{-x^2/4} H_n(x)$$

The *Hermite functions* form an orthonormal set:

$$\int_{-\infty}^{\infty} \psi_n(x)\, \psi_m(x)\, \mathrm{d}x = \begin{cases} 1 & \text{if } n = m \\ 0 & \text{if } n \neq m \end{cases}$$

The *Hermite functions* of increasing order oscillate more frequently, with the function of order n crossing zero n times.

```
> # Define the Hermite functions
> hermitef <- function(x, n)
+   exp(-x^2/4)*hermitep(x, n)/(2*pi)^(0.25)/sqrt(factorial(n))
> # Integrate the Hermite functions
> integrate(function(x, n, m)
+   hermitef(x, n)*hermitef(x, m),
+   lower=(-Inf), upper=Inf, n=2, m=3)
```




```
> colorv <- c("red", "blue", "green")
> for (indeks in 1:3) {  # Plot three curves
+   curve(expr=hermitef(x, indeks),
+   xlim=c(-6, 6), ylim=c(-0.6, 0.6),
+   xlab="", ylab="", lwd=4, col=colorv[indeks],
+   add=as.logical(indeks-1))
+ }  # end for
> # Add title and legend
> title(main="Hermite Functions", line=0.5)
> labelv <- paste("Order", 1:3, sep=" = ")
> legend("topright", inset=0.0, bty="n",
+  title=NULL, labelv, cex=0.8, lwd=6, lty=1,
+  col=colorv)
```

draft: The Hermite Series

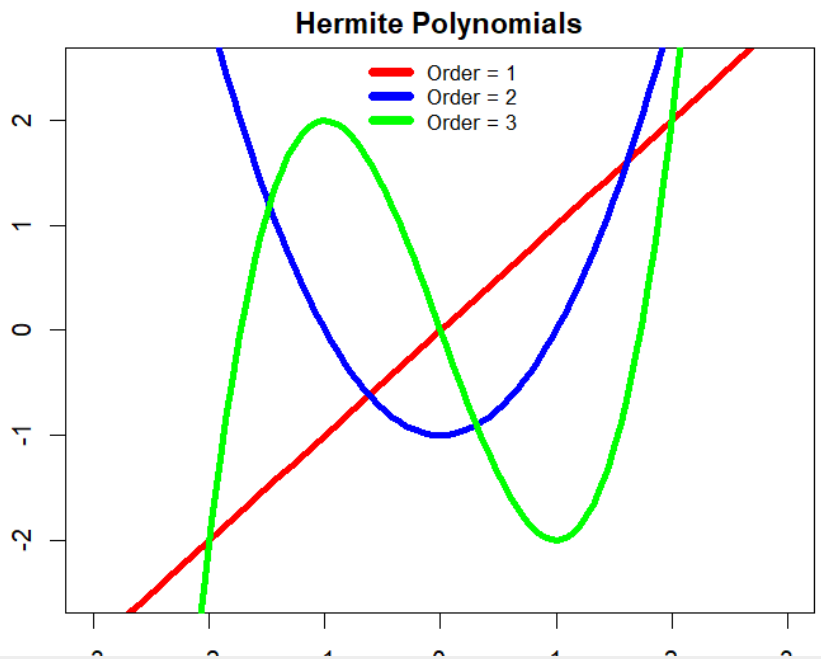
To-do: expand a probability distribution into a series of *Hermite functions*.

The *Hermite functions* $\psi_i(x)$ form an orthonormal basis that can be used to expand a given probability distribution $f(x)$ into a series:

$$f(x) = \sum_{i=0}^{n} f_i \, \psi_i(x)$$

The coefficients f_i are equal to:

$$f_i = \int_{-\infty}^{\infty} f(x) \, \psi_i(x) \, \mathrm{d}x$$




```
> # Integrate Hermite functions
> integrate(hermitep, lower=(-Inf), upper=Inf, n=2)
> integrate(function(x, n, m) hermitep(x, n)*hermitep(x, m),
+     lower=(-Inf), upper=Inf, n=2, m=3)
> integrate(function(x, n, m) hermitep(x, n)*hermitep(x, m),
+     lower=(-Inf), upper=Inf, n=2, m=2)
```

```
> colorv <- c("red", "blue", "green")
> for (indeks in 1:3) {  # Plot three curves
+   curve(expr=hermitep(x, indeks),
+   xlim=c(-4, 4), ylim=c(-0.6, 0.6),
+   xlab="", ylab="", lwd=3, col=colorv,
+   add=as.logical(indeks-1))
+ }  # end for
> # Add title and legend
> title(main="Hermite Functions", line=0.5)
> labelv <- paste("Order", 1:3, sep=" = ")
```

The Bell polynomials

The *Bell polynomials* B_n are the coefficients in the expansion of the exponent of a series:

$$\exp(\sum_{n=1}^{\infty} \kappa_n \frac{t^n}{n!}) = \sum_{n=0}^{\infty} B_n(\kappa 1, \ldots, \kappa_n) \frac{t^n}{n!}$$

The first four *Bell polynomials* are equal to:

$$B_0 = 1$$
$$B1(\kappa 1) = \kappa 1$$
$$B2(\kappa 1, \kappa 2) = \kappa 1^2 + \kappa 2$$
$$B3(\kappa 1, \kappa 2, \kappa 3) = \kappa 1^3 + 3\kappa 1\kappa 2 + \kappa 3$$

The Gram-Charlier Series

The *Gram-Charlier series* expresses the *density function* $f(x)$ in terms of its *cumulants* and a *basis function* $\phi(x)$, with *characteristic* $\hat{\phi}(t)$.

The *characteristic functions* $\hat{f}(t)$ and $\hat{\phi}(t)$ can be expressed in terms of their *cumulants* as:

$$\hat{f}(t) = e^{K_X(it)} = \exp(\sum_{n=1}^{\infty} \kappa_n \frac{(it)^n}{n!})$$

$$\hat{\phi}(t) = \exp(\sum_{n=1}^{\infty} \phi_n \frac{(it)^n}{n!})$$

Then $\hat{f}(t)$ can be expressed in terms of $\hat{\phi}(t)$ as:

$$\hat{f}(t) = \exp[\sum_{n=1}^{\infty} (\kappa_n - \phi_n) \frac{(it)^n}{n!}]\, \hat{\phi}(t)$$

The *basis function* $\phi(x)$ can be chosen to be a *normal distribution*, with *mean* and *standard deviation* equal to that of $f(x)$, and with all its *normal cumulants* ϕ_n of order 3 and higher equal to zero.

Then we get a series starting at `n=3`:

$$\hat{f}(t) = \exp[\sum_{n=3}^{\infty} \kappa_n \frac{(it)^n}{n!}]\, \hat{\phi}(t)$$

We can expand the exponent and collect terms with the same power of t using the *Bell polynomials* B_n:

$$\hat{f}(t) = \sum_{n=0}^{\infty} B_n(0, 0, \kappa 3, \ldots, \kappa_n)\, \frac{(it)^n}{n!}\, \hat{\phi}(t)$$

The *inverse Fourier transform* of the above equation gives the *probability function* $f(x)$:

$$f(x) = \sum_{n=0}^{\infty} B_n(0, 0, \kappa 3, \ldots, \kappa_n) \frac{(-1)^n}{n!} \frac{d^n \phi(x)}{dx^n}$$

The derivatives of the *normal* distribution $\phi(x)$ can be expressed using the *Hermite polynomials* H_n so that the *Gram-Charlier series* becomes:

$$f(x) = \phi(x) \sum_{n=0}^{\infty} \frac{B_n(0, 0, \kappa 3, \ldots, \kappa_n)}{n! \sigma^n} H_n(\frac{x-\mu}{\sigma})$$

draft: The Edgeworth Expansion

The *Edgeworth expansion* expresses the *probability density function* $f(x)$ as a series of its *cumulants* and a *basis function* $\phi(x)$ (with *characteristic function* $\hat{\phi}(t)$).

The *characteristic functions* $\hat{f}(t)$ and $\hat{\phi}(t)$ can be expressed in terms of their corresponding *cumulants* as:

$$\hat{f}(t) = e^{K_X(it)} = \exp(\sum_{n=1}^{\infty} \kappa_n \frac{(it)^n}{n!})$$

$$\hat{\phi}(t) = \exp(\sum_{n=1}^{\infty} \phi_n \frac{(it)^n}{n!})$$

Then $\hat{f}(t)$ can be expressed in terms of $\hat{\phi}(t)$ as:

$$\hat{f}(t) = \exp[\sum_{n=1}^{\infty} (\kappa_n - \phi_n) \frac{(it)^n}{n!}]\, \hat{\phi}(t)$$

If the *basis function* $\phi(x)$ is chosen to be the *normal distribution*, with *mean* and *standard deviation* equal to that of $f(x)$, and since the *normal cumulants* of order 3 and higher are all equal to zero, then we get:

$$\hat{f}(t) = \exp[\sum_{n=3}^{\infty} \kappa_n (-1)^n \frac{(it)^n}{n!}]\, \hat{\phi}(t)$$

The *inverse Fourier transform* of the above equation gives the *probability function* $f(x)$:

$$f(x) = \exp[\sum_{n=3}^{\infty} \kappa_n (-1)^n \frac{d^n}{dx^n}]\, \phi(x)$$

Now expand the exponent in a series and collect terms with the same order of the derivative to obtain:

$$f(x) = \sum_{n=0}^{\infty} B_n(0, 0, \kappa 3, \ldots, \kappa_n)(-1)^n \frac{d^n \phi(x)}{dx^n}$$

draft: The Cornish-Fisher Expansion

The *Cornish-Fisher expansion* expresses the quantiles of a distribution as a series of its *moments*.

The Edgeworth Expansion The *cumulant generating function* $K_X(t)$ is equal to the logarithm of the *moment generating function*:

$$K_X(t) = \log M_X(t)$$

The n-th derivative of $K_X(t)$ with respect to t, at $t = 0$ is equal to the n-th *cumulant* κ_n:

$$\kappa_n = \frac{d^n K_X(t)}{dt^n}|_{t=0}$$

The *cumulants* are related to the *moments* of the distribution: the first three cumulants are equal to the *central moments* (mean, variance, and skewness), while the higher order *cumulants* can be expressed as polynomials of the *central moments*.

The *cumulant generating function* $K_X(t)$ can be expanded into a power series of the *cumulants*:

$$K_X(t) = \sum_{n=1}^{n} \frac{\kappa_n t^n}{n!} = \mu t + \sigma^2 t^2/2 +$$

The n-th *moment* μ_n is not equal to the *central moment* $\mathbb{E}[(x-\mu)^n]$.
The *moment generating function* $M_X(t)$ of a random variable x with the probability function $f(x)$ is equal to:

$$\int^{\infty}$$

```
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> # Sample from Standard Normal Distribution
> nrows <- 1000
> datav <- rnorm(nrows)
> # Sample mean
> mean(datav)
> # Sample standard deviation
> sd(datav)
```

Hypothesis Testing

Hypothesis Tests are designed to test the validity of *null hypotheses*, and they consist of:

- A *null hypothesis*,
- A test *statistic* derived from the data sample,
- A *p*-value: the conditional probability of observing the test statistic value, assuming the *null hypothesis* is TRUE,
- A *significance level* α corresponding to a *critical value*.

The *p*-value is compared to the *significance level* and if the *p*-value is less than the *significance level* α, then the *null hypothesis* is rejected.

It's possible for the *null hypothesis* to be TRUE, but to obtain a very small *p*-value purely by chance.

The *p*-value is the probability of erroneously rejecting a TRUE *null hypothesis*, due to the randomness of the data sample.

```
> ### Perform two-tailed test that sample is
> ### from Standard Normal Distribution (mean=0, SD=1)
> # generate vector of samples and store in data frame
> test_frame <- data.frame(samples=rnorm(1e4))
> # get p-values for all the samples
> test_frame$p_values <- sapply(test_frame$samples,
+         function(x) 2*pnorm(-abs(x)))
> # Significance level, two-tailed test, critical value=2*SD
> signif_confl <- 2*(1-pnorm(2))
> # Compare p_values to significance level
> test_frame$result <-
+   test_frame$p_values > signif_level
> # Number of null rejections
> sum(!test_frame$result) / NROW(test_frame)
> # Show null rejections
> head(test_frame[!test_frame$result, ])
```

The *p*-value is a conditional probability, and is not equal to the un-conditional probability of the hypothesis being TRUE.

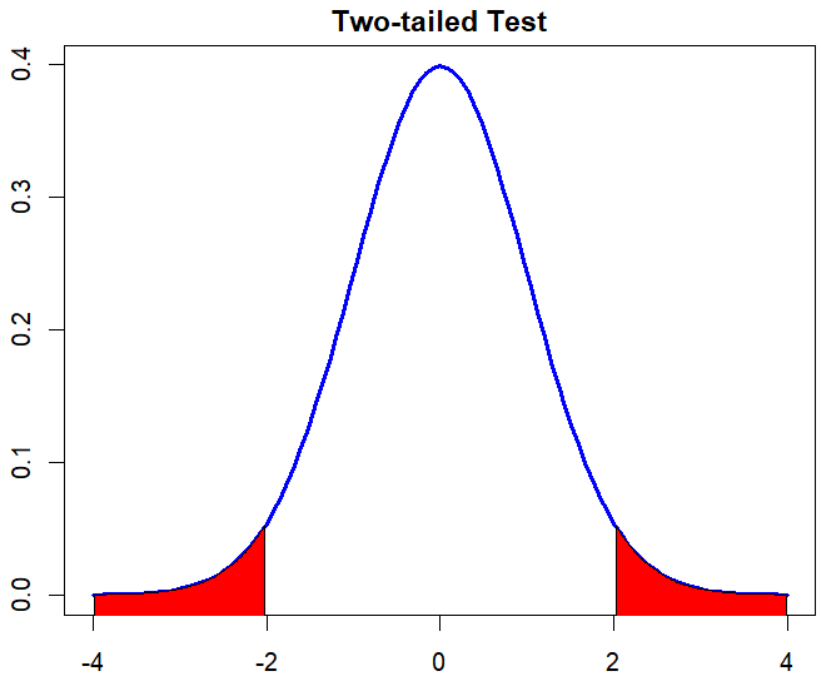
In statistics we cannot *prove* that a hypothesis is TRUE or not, but we can attempt to invalidate it, and conclude that it's unlikely to be TRUE, given the test statistic value and its *p*-value.

Two-tailed Hypothesis Tests

In two-tailed hypothesis tests, both tails of the probability distribution contribute to the *p*-value.

Two-tailed hypothesis tests are applied for testing if the absolute value of a sample exceeds the critical value.

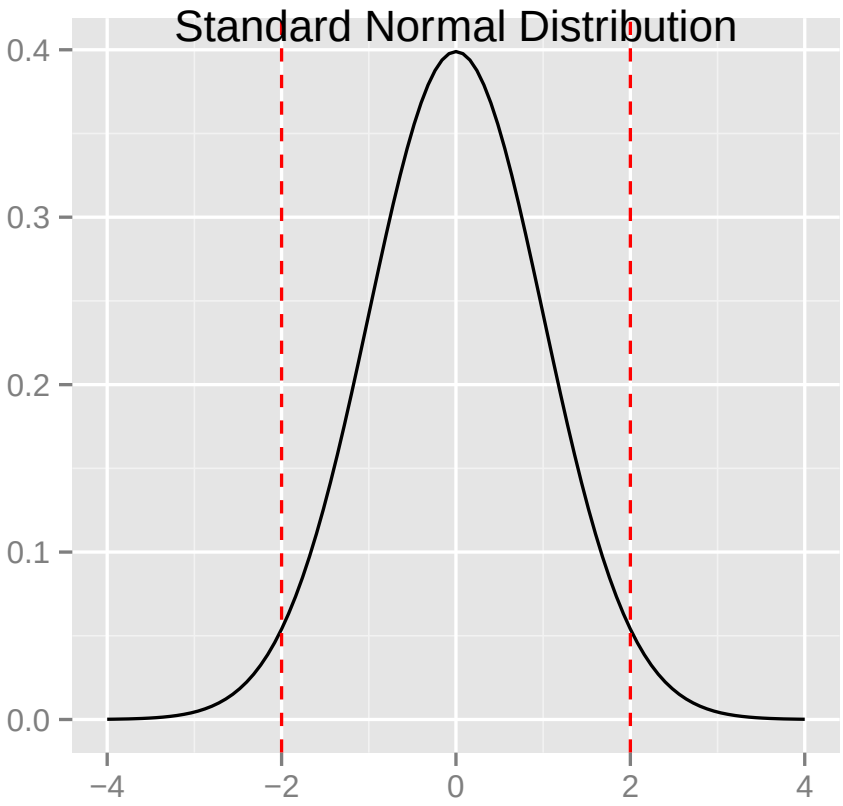
```
> # Plot the Normal probability distribution
> curve(expr=dnorm(x, sd=1), type="l", xlim=c(-4, 4),
+ xlab="", ylab="", lwd=3, col="blue")
> title(main="Two-tailed Test", line=0.5)
> # Plot tails of the distribution using polygons
> x1 <- 2; x2 <- 4
> # Plot right tail using polygon
> xvar <- seq(x1, x2, length=100)
> yxvar <- dnorm(xvar, sd=1)
> yxvar[1] <- (-1)
> yxvar[NROW(yxvar)] <- (-1)
> polygon(x=xvar, y=yxvar, col="red")
> # Plot left tail using polygon
> yxvar <- dnorm(-xvar, sd=1)
> yxvar[1] <- (-1)
> yxvar[NROW(yxvar)] <- (-1)
> polygon(x=(-xvar), y=yxvar, col="red")
```



Visualizing Hypothesis Testing Using Package *ggplot2*

In two-tailed hypothesis tests, both tails of the probability distribution contribute to the *p*-value.

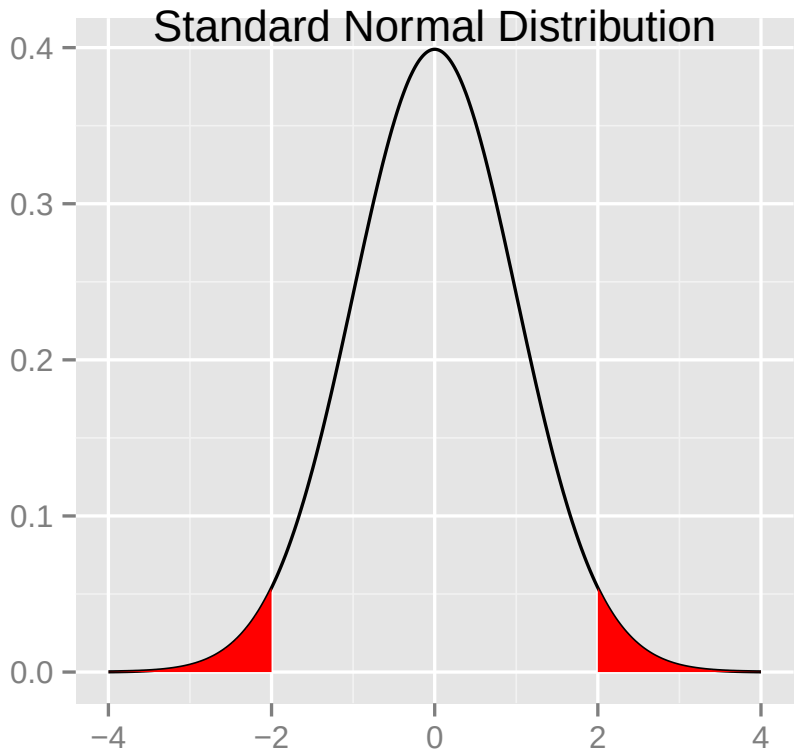
```
> library(ggplot2)  # Load ggplot2
>
> qplot(  # Simple ggplot2
+     main="Standard Normal Distribution",
+     c(-4, 4),
+     stat="function",
+     fun=dnorm,
+     geom="line",
+     xlab=NULL, ylab=NULL
+     ) +  # end qplot
+
+ theme(  # Modify plot theme
+     plot.title=element_text(vjust=-1.0),
+     plot.background=element_blank()
+     ) +  # end theme
+
+ geom_vline(  # Add vertical line
+   aes(xintercept=c(-2.0, 2.0)),
+   colour="red",
+   linetype="dashed"
+   )  # end geom_vline
```



Visualizing Hypothesis Testing Using *ggplot2* (cont.)

In two-tailed hypothesis tests, both tails of the probability distribution contribute to the *p*-value.

```
> ### Create ggplot2 with shaded area
> xvar <- -400:400/100
> norm_frame <- data.frame(xvar=xvar,
+                  d.norm=dnorm(xvar))
> norm_frame$shade <- ifelse(
+             abs(norm_frame$xvar) >= 2,
+             norm_frame$d.norm, NA)
> ggplot(  # Main function
+   data=norm_frame,
+   mapping=aes(x=xvar, y=d.norm)
+   ) +  # end ggplot
+ # Plot line
+   geom_line() +
+ # Plot shaded area
+   geom_ribbon(aes(ymin=0, ymax=shade), fill="red") +
+ # No axis labels
+   xlab("") + ylab("") +
+ # Add title
+   ggtitle("Standard Normal Distribution") +
+ # Modify plot theme
+   theme(
+   plot.title=element_text(vjust=-1.0),
+   plot.background=element_blank()
+   )  # end theme
```



Student's t-test for the Distribution Mean

Student's t-test is designed to test the *null hypothesis* that a sample: $\{x_1, \ldots, x_n\}$ was obtained from a normal distribution with a *mean* equal to μ.

The test statistic is equal to the *t-ratio*:

$$t = \frac{\bar{x} - \mu}{\hat{\sigma}/\sqrt{n}}$$

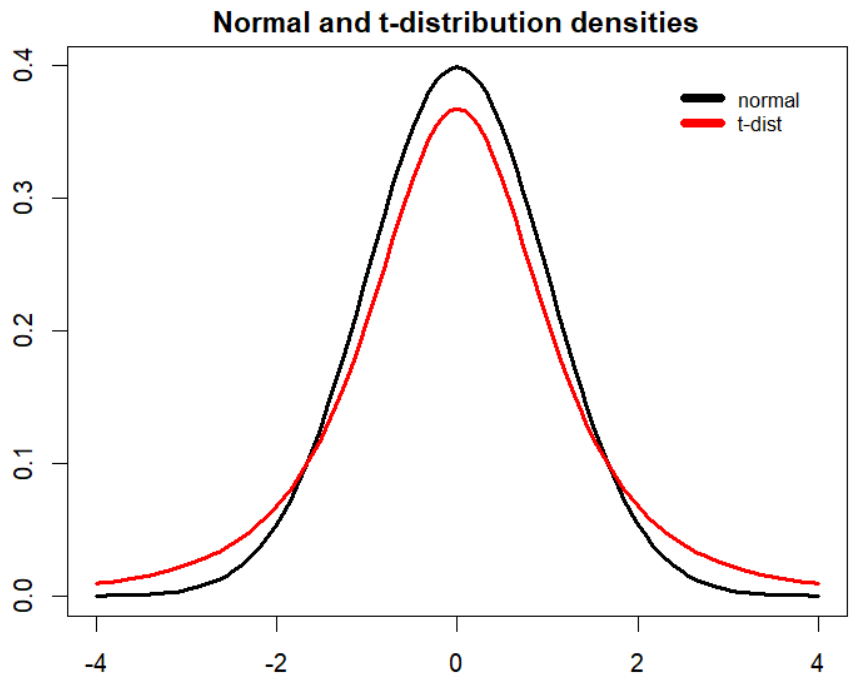
Where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the sample mean and $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$ is the sample variance.

Under the *null hypothesis* the *t-ratio* follows the *t-distribution* with n degrees of freedom, with the probability density function:

$$f(t) = \frac{\Gamma((n+1)/2)}{\sqrt{\pi n}\,\Gamma(n/2)}\,(1 + t^2/n)^{-(n+1)/2}$$

Student's t-test can also be used to test if two different normally distributed samples have equal *population means*.

Student's t-test is not valid for random variables that do not follow the normal distribution.




```
> # t-test for single sample
> t.test(rnorm(100))
> # t-test for two samples
> t.test(rnorm(100),
+        rnorm(100, mean=1))
> # Plot the normal and t-distribution densities
> x11(width=6, height=5)
> par(mar=c(3, 3, 3, 1), oma=c(0, 0, 0, 0))
> curve(expr=dnorm, xlim=c(-4, 4),
+       xlab="", ylab="", lwd=3)
> curve(expr=dt(x, df=3),
+       xlab="", ylab="", lwd=3,
+       col="red", add=TRUE)
> # Add title
> title(main="Normal and t-distribution densities", line=0.5)
> # Add legend
```

draft: The Analysis of Variance (*ANOVA*)

The Analysis of Variance (*ANOVA*) to test if the sub-samples of the data have the same mean.

ANOVA provides a statistical test of whether two or more population means are equal, and therefore generalizes the t-test beyond two means. *Student's t-test* is designed to test the *null hypothesis* that a sample: $\{x_1, \ldots, x_n\}$ was obtained from a normal distribution with a *mean* equal to μ.

For example, *ANOVA* is widely used to study the effect of medical treatments, with the *null hypothesis* being that the treatments have no effect and that differences are due to random chance.

The test statistic is equal to the *t-ratio*:

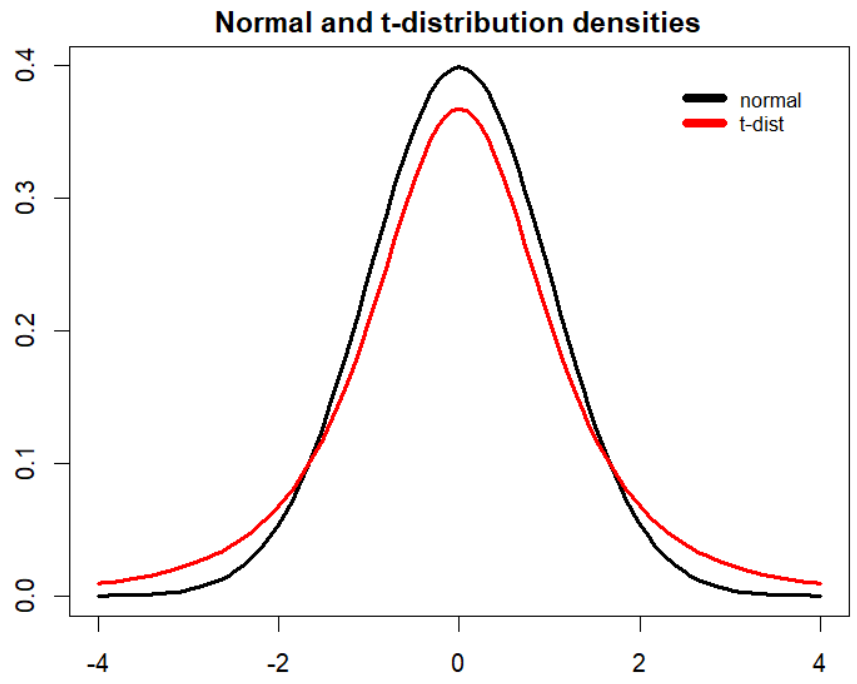
$$t = \frac{\bar{x} - \mu}{\hat{\sigma}/\sqrt{n}}$$

Where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the sample mean and $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$ is the sample variance.

Under the *null hypothesis* the *t-ratio* follows the *t-distribution* with n degrees of freedom, with the probability density function:

$$f(x) = \frac{\Gamma((n+1)/2)}{\sqrt{\pi n}\,\Gamma(n/2)}\,(1 + x^2/n)^{-(n+1)/2}$$

Student's t-test can also be used to test if two different normally distributed samples have equal *population*




```
> # t-test for single sample
> t.test(rnorm(100))
> # t-test for two samples
> t.test(rnorm(100),
+        rnorm(100, mean=1))
> # Plot the normal and t-distribution densities
> x11(width=6, height=5)
> par(mar=c(3, 3, 3, 1), oma=c(0, 0, 0, 0))
> curve(expr=dnorm, xlim=c(-4, 4),
+       xlab="", ylab="", lwd=3)
> curve(expr=dt(x, df=3),
+       xlab="", ylab="", lwd=3,
+       col="red", add=TRUE)
> # Add title
> title(main="Normal and t-distribution densities", line=0.5)
> # Add legend
```

The *Shapiro-Wilk* Test of Normality

The *Shapiro-Wilk* test is designed to test the *null hypothesis* that a sample: $\{x_1, \ldots, x_n\}$ is from a normally distributed population.

The test statistic is equal to:

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Where the: $\{a_1, \ldots, a_n\}$ are proportional to the *order statistics* of random variables from the normal distribution.

$x_{(k)}$ is the k-th *order statistic*, and is equal to the k-th smallest value in the sample: $\{x_1, \ldots, x_n\}$.

The *Shapiro-Wilk* statistic follows its own distribution, and is less than or equal to 1.

The *Shapiro-Wilk* statistic is close to 1 for samples from normal distributions.

The p-value for *VTI* returns is extremely small, and we conclude that the *null hypothesis* is `FALSE`, and the *VTI* returns are not from a normally distributed population.

The *Shapiro-Wilk* test is not reliable for large sample sizes, so it's limited to less than 5000 sample size.

```
> # Calculate VTI percentage returns
> library(rutils)
> retp <- as.numeric(na.omit(rutils::etfenv$returns$VTI))[1:499]
> # Reduce number of output digits
> ndigits <- options(digits=5)
> # Shapiro-Wilk test for normal distribution
> nrows <- NROW(retp)
> shapiro.test(rnorm(nrows))

 Shapiro-Wilk normality test

data:  rnorm(nrows)
W = 0.995, p-value = 0.098
> # Shapiro-Wilk test for VTI returns
> shapiro.test(retp)

 Shapiro-Wilk normality test

data:  retp
W = 0.991, p-value = 0.0029
> # Shapiro-Wilk test for uniform distribution
> shapiro.test(runif(nrows))

 Shapiro-Wilk normality test

data:  runif(nrows)
W = 0.953, p-value = 1.7e-11
> # Restore output digits
> options(digits=ndigits$digits)
```

The *Jarque-Bera* Test of Normality

The *Jarque-Bera* test is designed to test the *null hypothesis* that a sample: $\{x_1, \ldots, x_n\}$ is from a normally distributed population.

The test statistic is equal to:

$$JB = \frac{n}{6}(\varsigma^2 + \frac{1}{4}(\kappa - 3)^2)$$

Where the *skewness* and *kurtosis* are defined as:

$$\varsigma = \frac{1}{n}\sum_{i=1}^{n}(\frac{x_i - \bar{x}}{\hat{\sigma}})^3 \qquad \kappa = \frac{1}{n}\sum_{i=1}^{n}(\frac{x_i - \bar{x}}{\hat{\sigma}})^4$$

The *Jarque-Bera* statistic asymptotically follows the *chi-squared* distribution with two degrees of freedom.

The *Jarque-Bera* statistic is small for samples from normal distributions.

The *p*-value for *VTI* returns is extremely small, and we conclude that the *null hypothesis* is `FALSE`, and the *VTI* returns are not from a normally distributed population.

```
> library(tseries)  # Load package tseries
> # Jarque-Bera test for normal distribution
> jarque.bera.test(rnorm(nrows))

 Jarque Bera Test

data:  rnorm(nrows)
X-squared = 2, df = 2, p-value = 0.3
> # Jarque-Bera test for VTI returns
> jarque.bera.test(retp)

 Jarque Bera Test

data:  retp
X-squared = 22, df = 2, p-value = 2e-05
> # Jarque-Bera test for uniform distribution
> jarque.bera.test(runif(nrows))

 Jarque Bera Test

data:  runif(nrows)
X-squared = 28, df = 2, p-value = 1e-06
```

The *Kolmogorov-Smirnov* Test for Probability Distributions

The *Kolmogorov-Smirnov* test *null hypothesis* is that two samples: $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$ were obtained from the same probability distribution.

The *Kolmogorov-Smirnov* statistic depends on the maximum difference between two empirical cumulative distribution functions (cumulative frequencies):

$$D = \sup_i |P(x_i) - P(y_i)|$$

The function `ks.test()` performs the *Kolmogorov-Smirnov* test and returns the statistic and its *p*-value *invisibly*.

The second argument to `ks.test()` can be either a `numeric` vector of cumulative frequencies, or a name of a cumulative distribution function.

The *Kolmogorov-Smirnov* test can be used as a *goodness of fit* test, to test if a set of observations fits a probability distribution.

The *K-S* test is more sensitive to the tails of the distribution because it depends on the maximum difference between the two empirical cumulative frequencies.

```
> # KS test for normal distribution
> kstest <- ks.test(rnorm(100), pnorm)
> kstest$p.value
> # KS test for uniform distribution
> ks.test(runif(100), pnorm)
> # KS test for two shifted normal distributions
> ks.test(rnorm(100), rnorm(100, mean=0.1))
> ks.test(rnorm(100), rnorm(100, mean=1.0))
> # KS test for two different normal distributions
> ks.test(rnorm(100), rnorm(100, sd=2.0))
> # KS test for VTI returns vs normal distribution
> retp <- as.numeric(na.omit(rutils::etfenv$returns$VTI))
> retp <- (retp - mean(retp))/sd(retp)
> ks.test(retp, pnorm)
```

Chi-squared Distribution

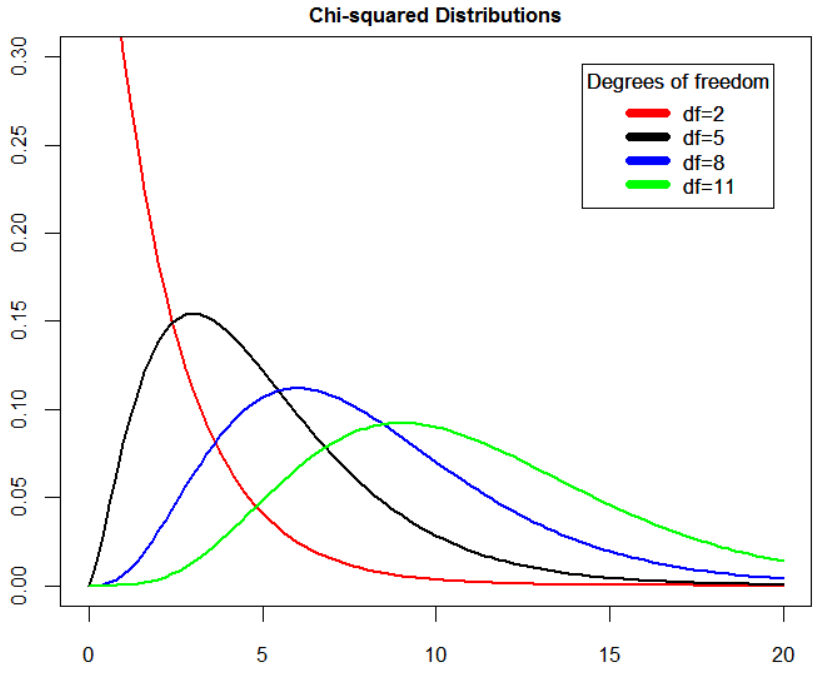
Let $z_1, \ldots, z_k$ be independent standard *Normal* random variables.

Then the random variable $X = \sum_{i=1}^{k} z_i^2$ is distributed according to the *Chi-squared* distribution with k degrees of freedom: $X \sim \chi_k^2$, and its probability density function is given by:

$$f(x) = \frac{x^{k/2-1}\, e^{-x/2}}{2^{k/2}\, \Gamma(k/2)}$$

The *Chi-squared* distribution with k degrees of freedom has mean equal to k and variance equal to $2k$.

```
> # Degrees of freedom
> degf <- c(2, 5, 8, 11)
> # Plot four curves in loop
> colorv <- c("red", "black", "blue", "green")
> for (indeks in 1:4) {
+   curve(expr=dchisq(x, df=degf[indeks]),
+   xlim=c(0, 20), ylim=c(0, 0.3),
+   xlab="", ylab="", col=colorv[indeks],
+   lwd=2, add=as.logical(indeks-1))
+ }  # end for
```




```
> # Add title
> title(main="Chi-squared Distributions", line=0.5)
> # Add legend
> labelv <- paste("df", degf, sep="=")
> legend("topright", inset=0.05, bty="n",
+        title="Degrees of freedom", labelv,
+        cex=0.8, lwd=6, lty=1, col=colorv)
```

The *Chi-squared* Test for the Goodness of Fit

Goodness of Fit tests are designed to test if a set of observations fits an assumed theoretical probability distribution.

The *Chi-squared* test tests if a frequency of counts fits the specified distribution.

The *Chi-squared* statistic is the sum of squared differences between the observed frequencies o_i and the theoretical frequencies p_i:

$$\chi^2 = N \sum_{i=1}^{n} \frac{(o_i - p_i)^2}{p_i}$$

Where N is the total number of observations.

The *null hypothesis* is that the observed frequencies are consistent with the theoretical distribution.

The function `chisq.test()` performs the *Chi-squared* test and returns the statistic and its p-value *invisibly*.

The parameter `breaks` in the function `hist()` should be chosen large enough to capture the shape of the frequency distribution.

```
> # Observed frequencies from random normal data
> histp <- hist(rnorm(1e3, mean=0), breaks=100, plot=FALSE)
> countsn <- histp$counts
> # Theoretical frequencies
> countst <- rutils::diffit(pnorm(histp$breaks))
> # Perform Chi-squared test for normal data
> chisq.test(x=countsn, p=countst, rescale.p=TRUE, simulate.p.value=
> # Return p-value
> chisqtest <- chisq.test(x=countsn, p=countst, rescale.p=TRUE, simu
> chisqtest$p.value
> # Observed frequencies from shifted normal data
> histp <- hist(rnorm(1e3, mean=2), breaks=100, plot=FALSE)
> countsn <- histp$counts/sum(histp$counts)
> # Theoretical frequencies
> countst <- rutils::diffit(pnorm(histp$breaks))
> # Perform Chi-squared test for shifted normal data
> chisq.test(x=countsn, p=countst, rescale.p=TRUE, simulate.p.value=
> # Calculate histogram of VTI returns
> histp <- hist(retp, breaks=100, plot=FALSE)
> countsn <- histp$counts
> # Calculate cumulative probabilities and then difference them
> countst <- pt((histp$breaks-loc)/scalev, df=2)
> countst <- rutils::diffit(countst)
> # Perform Chi-squared test for VTI returns
> chisq.test(x=countsn, p=countst, rescale.p=TRUE, simulate.p.value=
```

Sorting and Ranking Data

The function `sort()` returns a vector sorted into ascending order, from smallest to largest.

A permutation is a re-ordering of the elements of a vector.

The permutation index specifies how the elements are re-ordered in a permutation.

The function `order()` calculates the permutation index to sort a given vector into ascending order.

Applying the function `order()` twice: `order(order())`, calculates the permutation index to sort the vector from ascending order into its original unsorted order.

The permutation index produced by: `order(order())` is the reverse of the permutation index produced by: `order()`.

The function `rank()` calculates the ranks of the elements, according to their magnitude, from smallest to largest.

The ranks of the elements are equal to the reverse permutation index.

```
> # Sort a vector into ascending order
> datav <- round(runif(7), 3)
> sortv <- sort(datav)
> datav  # original data
[1] 0.683 0.424 0.082 0.536 0.119 0.388 0.299
> sortv  # sorted data
[1] 0.082 0.119 0.299 0.388 0.424 0.536 0.683
> # Calculate index to sort into ascending order
> indeks <- order(datav)
> indeks  # permutation index to sort
[1] 3 5 7 6 2 4 1
> all.equal(sortv, datav[indeks])
[1] TRUE
> # Sort the ordered vector back to its original unsorted order
> indeks <- order(order(datav))
> indeks  # permutation index to unsort
[1] 7 5 1 6 2 4 3
> all.equal(datav, sortv[indeks])
[1] TRUE
> # Calculate ranks of the vector elements
> rank(datav)
[1] 7 5 1 6 2 4 3
> all.equal(rank(datav), indeks)
[1] TRUE
```

The Mean and Median Estimators of Location

The *mean* and the *median* are both estimators of the *location* (centrality) of a distribution.

For *normally* distributed data, the *mean* has a smaller standard error than the *median* (it is more *efficient*).

But for distributions with very large *kurtosis* (fat tails), the *median* may be more *efficient* than the *mean*, because it's less sensitive to data outliers.

In addition, the *median* is often defined even for distributions for which the *mean* and *variance* are infinite.

For *symmetric* distributions, the expected values of the sample *mean* and the *median* are equal to each other, and equal to the population mean.

But for *skewed* distributions (for example asset returns), the *median* estimator is *biased* (its expected value is not equal to the population mean).

```
> # VTI returns
> retp <- as.numeric(na.omit(rutils::etfenv$returns[, "VTI"]))
> nrows <- NROW(retp)
> retp <- 100*(retp-mean(retp))/sd(retp)
> # Simulate normal random data
> ndata <- rnorm(nrows, sd=100)
> # Bootstrap the mean and median estimators
> bootd <- sapply(1:1e3, function(x) {
+   # Simulate data
+   ndata <- rnorm(nrows, sd=100)
+   retp <- retp[sample.int(nrows, replace=TRUE)]
+   c(n_mean=mean(ndata),
+     n_median=median(ndata),
+     vti_mean=mean(retp),
+     vti_median=median(retp))
+ })  # end sapply
> bootd <- t(bootd)
> # Analyze bootstrapped data
> head(bootd)
> sum(is.na(bootd))
> # Means and medians from bootstrap
> apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
> # Parallel bootstrap under Windows
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1  # Number of cores
> compclust <- makeCluster(ncores)  # initialize compute cluster
> bootd <- parLapply(compclust, 1:1e4,
+   function(x, nrows, retp) {
+     # Simulate data
+     ndata <- rnorm(nrows, sd=100)
+     retp <- retp[sample.int(nrows, replace=TRUE)]
+     c(n_mean=mean(ndata),
+ n_median=median(ndata),
+ vti_mean=mean(retp),
+ vti_median=median(retp))
+   }, nrows, retp)  # end parLapply
> stopCluster(compclust)  # Stop R processes over cluster
```

The Bias-Variance Tradeoff

The tradeoff between *unbiased* estimators but with *higher variance*, and efficient estimators with *lower variance* but with some *bias* is called the *bias-variance tradeoff*.

Let $\hat{\theta}$ be an estimator of the parameter θ, with expected value $\mathbb{E}[\hat{\theta}] = \bar{\theta}$ (which may not necessarily be equal to θ).

The *accuracy* of the estimator $\hat{\theta}$ can be measured by its *mean squared error* (MSE), equal to the expected value of the squared difference $(\hat{\theta} - \theta)^2$:

$$\text{MSE} = \mathbb{E}[(\hat{\theta} - \theta)^2] = \mathbb{E}[(\hat{\theta} - \bar{\theta} + \bar{\theta} - \theta)^2] =$$
$$\mathbb{E}[(\hat{\theta} - \bar{\theta})^2 + 2(\hat{\theta} - \bar{\theta})(\bar{\theta} - \theta) + (\bar{\theta} - \theta)^2] =$$
$$\mathbb{E}[(\hat{\theta} - \bar{\theta})^2] + (\bar{\theta} - \theta)^2 = \text{var}(\bar{\theta}) + \text{bias}(\bar{\theta})^2$$

Since $\mathbb{E}[(\hat{\theta} - \bar{\theta})(\bar{\theta} - \theta)] = (\bar{\theta} - \theta)\mathbb{E}[(\hat{\theta} - \bar{\theta})] = 0$

The above formula shows that the *MSE* is equal to the sum of the estimator *variance* plus the square of the estimator *bias*.

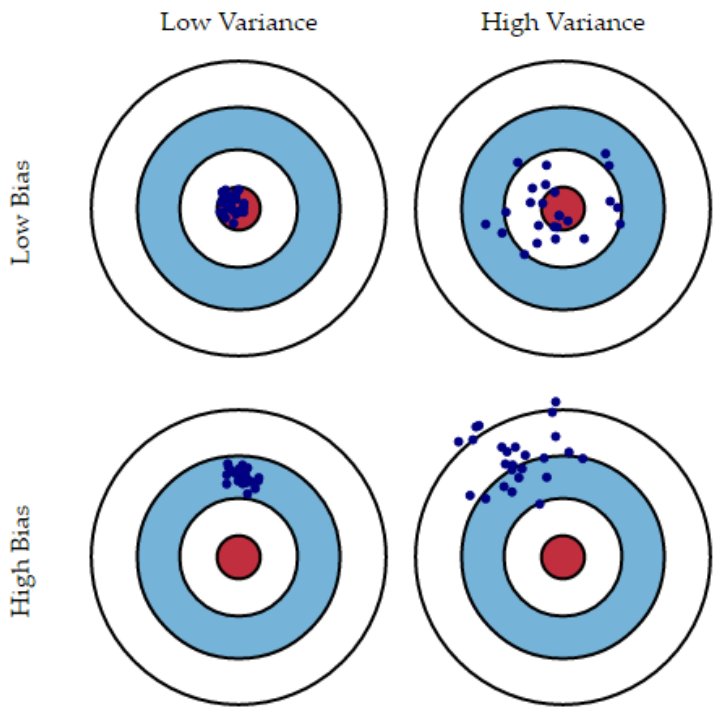


Fig. 1 Graphical illustration of bias and variance.

```
> # Bias and variance from bootstrap
> bias_var <- apply(bootd, MARGIN=2,
+   function(x) c(bias=mean(x), variance=var(x)))
> # MSE of mean
> bias_var[1, 3]^2 + bias_var[2, 3]
> # MSE of median
> bias_var[1, 4]^2 + bias_var[2, 4]
```

draft: The *Hodges-Lehmann* Estimator of Location

For distributions which are both *leptokurtic* (fat tailed) and *skewed*, the *median* estimator is more *efficient* but it's *biased*, while the *mean* is *unbiased* but it's less *efficient*.

The *Hodges-Lehmann* estimator of location is a compromise between the *mean* and the *median* estimators.

The *Hodges-Lehmann* estimator is the median of the means of all the possible one and two-element subsets.

For a dataset with n measurements, the set of all possible one or two-element subsets of it has $n(n+1)/2$ elements. For each such subset, the mean is computed. Finally, the median of these $n(n+1)/2$ averages is defined to be the *Hodges-Lehmann* estimator of location.

A subset is obtained by removing a few elements, and the subset mean is the mean of this subset.

Nonparametric Estimators distribution free methods, which do not rely on assumptions that the data are drawn from a given parametric family of probability distributions. As such it is the opposite of parametric statistics. accuracy

The *Median Absolute Deviation* (*MAD*) is a robust measure of dispersion (variability), defined using the *median* instead of the mean:

$$\text{MAD} = \text{median}(\text{abs}(x - \text{median}(x)))$$

```
> retp <- as.numeric(na.omit(rutils::etfenv$returns[, "VTI"]))
> nrows <- NROW(retp)
> retp <- 100*(retp-mean(retp))/sd(retp)
> # Simulate normal random data
> ndata <- rnorm(nrows, sd=100)
>
> # Hodges-Lehmann estimator
>
>
> # Bootstrap the mean and median estimators
> bootd <- sapply(1:1e3, function(x) {
+   # Simulate data
+   ndata <- rnorm(nrows, sd=100)
+   retp <- retp[sample.int(nrows, replace=TRUE)]
+   c(n_mean=mean(ndata),
+     n_median=median(ndata),
+     vti_mean=mean(retp),
+     vti_median=median(retp))
+ })  # end sapply
> bootd <- t(bootd)
> # Analyze bootstrapped data
> head(bootd)
> sum(is.na(bootd))
> # Means and medians from bootstrap
> apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
```

draft: Nonparametric Estimators of Location

Robust Order statistics, which are based on the ranks of observations, is one example of such statistics.

The *Median Absolute Deviation* (*MAD*) is a robust measure of dispersion (variability), defined using the *median* instead of the mean:

$$\text{MAD} = \text{median}(\text{abs}(x_i - \text{median}(\mathbf{x})))$$

The advantage of *MAD* is that it's always well defined, even for data that has infinite variance.

Explain breakdown point of estimators breakdown point of estimators

For normally distributed data, the *MAD* has a larger standard error than the standard deviation.

But for distributions with fat tails (like asset returns), the standard deviation has a larger standard error than the *MAD*.

The *MAD* for normally distributed data is equal to $\Phi^{-1}(0.75) \cdot \hat{\sigma} = 0.6745 \cdot \hat{\sigma}$.

The function `mad()` calculates the *MAD* and divides it by $\Phi^{-1}(0.75)$ to make it comparable to the standard deviation.

```
> nrows <- 1e3
> datav <- rnorm(nrows)
> sd(datav)
> mad(datav)
> median(abs(datav - median(datav)))
> median(abs(datav - median(datav)))/qnorm(0.75)
> # Bootstrap of sd and mad estimators
> bootd <- sapply(1:1e4, function(x) {
+   samplev <- datav[sample.int(nrows, replace=TRUE)]
+   c(sd=sd(samplev), mad=mad(samplev))
+ })  # end sapply
> bootd <- t(bootd)
> # Analyze bootstrapped variance
> head(bootd)
> sum(is.na(bootd))
> # Means and standard errors from bootstrap
> apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
> # Parallel bootstrap under Windows
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1  # Number of cores
> compclust <- makeCluster(ncores)  # initialize compute cluster
> bootd <- parLapply(compclust, 1:1e4,
+   function(x, datav) {
+     samplev <- datav[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   }, datav=datav)  # end parLapply
> stopCluster(compclust)  # Stop R processes over cluster
> # Parallel bootstrap under Mac-OSX or Linux
> bootd <- mclapply(1:1e4, function(x) {
+     samplev <- datav[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   }, mc.cores=ncores)  # end mclapply
> # Means and standard errors from bootstrap
> bootd <- rutils::do_call(rbind, bootd)
> apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
```

draft: Robust Estimators and Influence Functions

The influence function measures the sensitivity of an estimator to changes in the values of individual data points.

But for distributions with very large kurtosis (fat tails), the *median* may have a smaller standard error than the mean, because it's less sensitive to outliers.

Statistical estimators are functions of samples (which are random variables), and therefore are themselves *random variables*.

The *standard error* (SE) of an estimator is defined as its *standard deviation* (not to be confused with the *population standard deviation* of the underlying random variable).

For example, the *standard error* of the estimator of the mean is equal to:

$$\sigma_\mu = \frac{\sigma}{\sqrt{n}}$$

Where σ is the *population standard deviation* (which is usually unkown).

The *estimator* of this *standard error* is equal to:

$$SE_\mu = \frac{\hat{\sigma}}{\sqrt{n}}$$

where: $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$ is the sample standard deviation (the estimator of the population standard deviation)

```
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> # Sample from Standard Normal Distribution
> nrows <- 1e3
> datav <- rnorm(nrows)
> # Sample mean
> mean(datav)
> # Sample standard deviation
> sd(datav)
```

The *Wilcoxon Signed Rank* Test for Distribution Symmetry

The *null hypothesis* of the *Wilcoxon Signed Rank* test is that the data sample is symmetric about a given location.

The *null hypothesis* is rejected if the data sample is not centered on the given location or if it's skewed.

For a single sample of data, the *Wilcoxon Signed Rank* test tests if the distribution is symmetric about its location.

For *symmetric* distributions, the *Wilcoxon* test only requires that the distributions have equal *medians*, but they can have different *standard deviations*. and shapes (skewness)

But for *skewed* distributions, the *Wilcoxon* test requires that the distributions be the same.

The *Wilcoxon Signed Rank* test is a nonparametric generalization of the Student's t-test.

The *null hypothesis* of the *Wilcoxon Signed Rank* test is that the two data samples, x_i and y_i, were obtained from *similar* probability distributions.

The function `wilcox.test()` with parameter `paired=TRUE` calculates the *Wilcoxon Signed Rank* test statistic and its *p*-value.

If a single argument is passed into `wilcox.test()` then it tests if the data has zero *median*.

```
> nrows <- 1e3
> # Wilcoxon test for normal distribution
> normv <- rnorm(nrows)
> wilcox.test(normv)
> wilcox.test(normv+1)
> # Skewed distribution with median=0
> lognormv <- exp(normv)
> # lognormv <- rlnorm(nrows, sdlog=1)
> mean(lognormv); median(lognormv)
> # Skewed distribution with median!=0
> wilcox.test(lognormv)
> # Skewed distribution with median=0
> wilcox.test(lognormv-median(lognormv))
>
> # Skewed distributions with median!=0
> wilcox.test(lognormv, normv, paired=TRUE)
>
> # Two distributions with median=0
> wilcox.test(lognormv-median(lognormv), normv-median(normv),
+       paired=TRUE)
>
> # Skewed distribution with median=0
> wilcox.test(lognormv-median(lognormv))
>
>
> # Skewed distribution with mean=0
> mean(lognormv); median(lognormv)
> wilcox.test(lognormv-median(lognormv))
> # Same as
> wilcox.test(lognormv-median(lognormv), rep(0, nrows), paired=TRUE)
>
> # Skewed distributions with median!=0
> wilcox.test(lognormv, normv, paired=TRUE)
> # Two distributions with median=0
> wilcox.test(lognormv-median(lognormv), normv-median(normv),
+       paired=TRUE)
> # Normal samples with different standard deviations
> sample1 <- rnorm(nrows, sd=1)
```

draft: The W Statistic of the *Wilcoxon Signed Rank* Test

Let $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$ be two samples of data, which form pairs of observations: $\{x_1, y_1\}, \ldots, \{x_n, y_n\}$.

The W statistic of the *Wilcoxon Signed Rank* test is equal to the sum of the ranks of the absolute differences $r_i = \text{rank}(|x_i - y_i|)$, weighted by their signs:

$$W = \sum_{i=1}^{n} \text{sgn}(x_i - y_i) r_i$$

The function `wilcox.test()` returns the V statistic, not the the W statistic:

$$V = \sum_{i=1}^{n} \text{H}(x_i - y_i) r_i$$

Where $\text{H}(x) = 1$ if $x > 0$, and 0 otherwise.

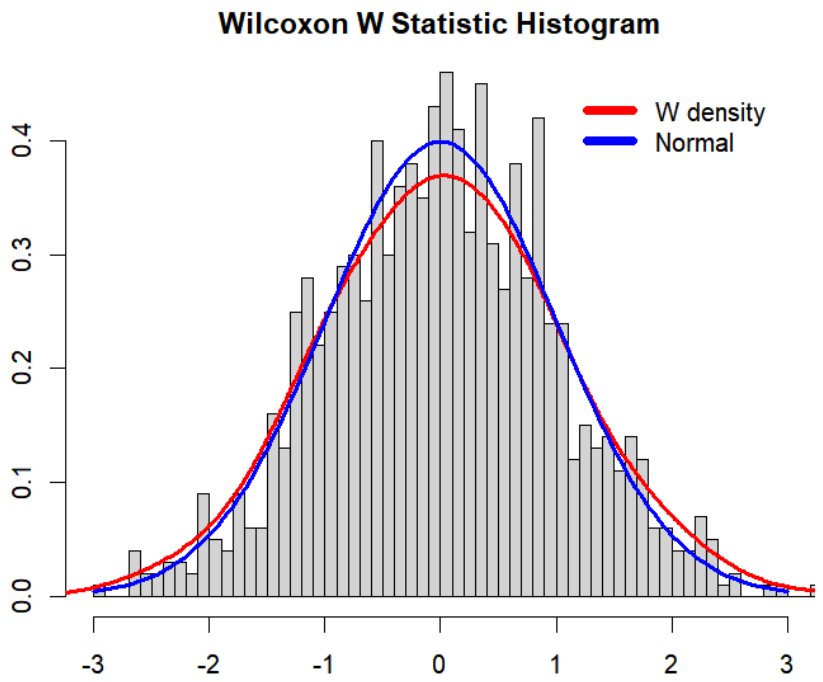
```
> # Wilcoxon test for random data around 0
> unifv <- (runif(nrows) - 0.5)
> wilcoxt <- wilcox.test(unifv)
> # Calculate V statistic of Wilcoxon test
> wilcoxt$statistic
> sum(rank(abs(unifv))[unifv>0])
> # Calculate W statistic of Wilcoxon test
> sum(sign(unifv)*rank(abs(unifv)))
> # Two sets of normal data
> sample1 <- rnorm(nrows)
> sample2 <- rnorm(nrows, mean=0.1)
> # Wilcoxon test
> wilcoxt <- wilcox.test(sample1, sample2, paired=TRUE)
> wilcoxt$statistic
> # Calculate V statistic of Wilcoxon test
> datav <- (sample1 - sample2)
> sum(rank(abs(datav))[datav>0])
> # Calculate W statistic of Wilcoxon test
> sum(sign(datav)*rank(abs(datav)))
```

draft: The Distribution of the W Statistic

The W statistic follows a distribution without a simple formula, which converges to the normal distribution for large sample size n, with an expected value equal to 0 and a variance equal to $\frac{n(n+1)(2n+1)}{6}$.

```
> # Calculate distributon of Wilcoxon W statistic
> wilcoxw <- sapply(1:1e3, function(x) {
+   datav <- (runif(nrows) - 0.5)
+   sum(sign(datav)*rank(abs(datav)))
+ })  # end sapply
> wilcoxw <- wilcoxw/sqrt(nrows*(nrows+1)*(2*nrows+1)/6)
> var(wilcoxw)
```

```
> hist(wilcoxw, col="lightgrey",
+      xlab="returns", breaks=50, xlim=c(-3, 3),
+      ylab="frequency", freq=FALSE,
+      main="Wilcoxon W Statistic Histogram")
> lines(density(wilcoxw, bw=0.4), lwd=3, col="red")
> curve(expr=dnorm, add=TRUE, lwd=3, col="blue")
> # Add legend
> legend("topright", inset=0.05, bty="n",
+  leg=c("W density", "Normal"),
+  lwd=6, lty=1, col=c("red", "blue"))
```



draft: The *Wilcoxon Signed Rank* Two-Sample Test

The *null hypothesis* of the *Wilcoxon Signed Rank* test is that the two data samples, x_i and y_i, were obtained from *similar* probability distributions.

Fix?
For *symmetric* distributions, the *Wilcoxon* test only requires that the distributions have equal *medians*, but they can have different *standard deviations*. and shapes (skewness)

Fix?
But for *skewed* distributions, the *Wilcoxon* test requires that the distributions be the same.

The *Wilcoxon Signed Rank* test is a nonparametric generalization of the Student's t-test.

The function `wilcox.test()` with parameter `paired=TRUE` calculates the *Wilcoxon Signed Rank* test statistic and its *p*-value.

If a single argument is passed into `wilcox.test()` then it tests if the data has zero *median*.

```
> nrows <- 1e3
> # Wilcoxon test for normal distribution
> normv <- rnorm(nrows)
> wilcox.test(normv)
> wilcox.test(normv+1)
> # Skewed distribution with median=0
> lognormv <- rlnorm(nrows, sdlog=1)
> mean(lognormv); median(lognormv)
> # Skewed distribution with median!=0
> wilcox.test(lognormv)
> # Skewed distribution with median=0
> wilcox.test(lognormv-median(lognormv))
>
> # Skewed distributions with median!=0
> wilcox.test(lognormv, normv, paired=TRUE)
>
> # Two distributions with median=0
> wilcox.test(lognormv-median(lognormv), normv-median(normv),
+       paired=TRUE)
>
> # Skewed distribution with median=0
> wilcox.test(lognormv-median(lognormv))
>
>
> # Skewed distribution with mean=0
> mean(lognormv); median(lognormv)
> wilcox.test(lognormv-median(lognormv))
> # Same as
> wilcox.test(lognormv-median(lognormv), rep(0, nrows), paired=TRUE)
>
> # Skewed distributions with median!=0
> wilcox.test(lognormv, normv, paired=TRUE)
> # Two distributions with median=0
> wilcox.test(lognormv-median(lognormv), normv-median(normv),
+       paired=TRUE)
> # Normal samples with different standard deviations
> sample1 <- rnorm(nrows, sd=1)
> sample2 <- rnorm(nrows, sd=10)
```

draft: The W Statistic of the *Wilcoxon Signed Rank* Two-Sample Test

Let $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$ be two samples of data, which form pairs of observations: $\{x_1, y_1\}, \ldots, \{x_n, y_n\}$.

The W statistic of the *Wilcoxon Signed Rank* test is equal to the sum of the ranks of the absolute differences $r_i = \text{rank}(|x_i - y_i|)$, weighted by their signs:

$$W = \sum_{i=1}^{n} \text{sgn}(x_i - y_i) r_i$$

The function `wilcox.test()` returns the V statistic, not the the W statistic:

$$V = \sum_{i=1}^{n} \text{H}(x_i - y_i) r_i$$

Where $\text{H}(x) = 1$ if $x > 0$, and 0 otherwise.

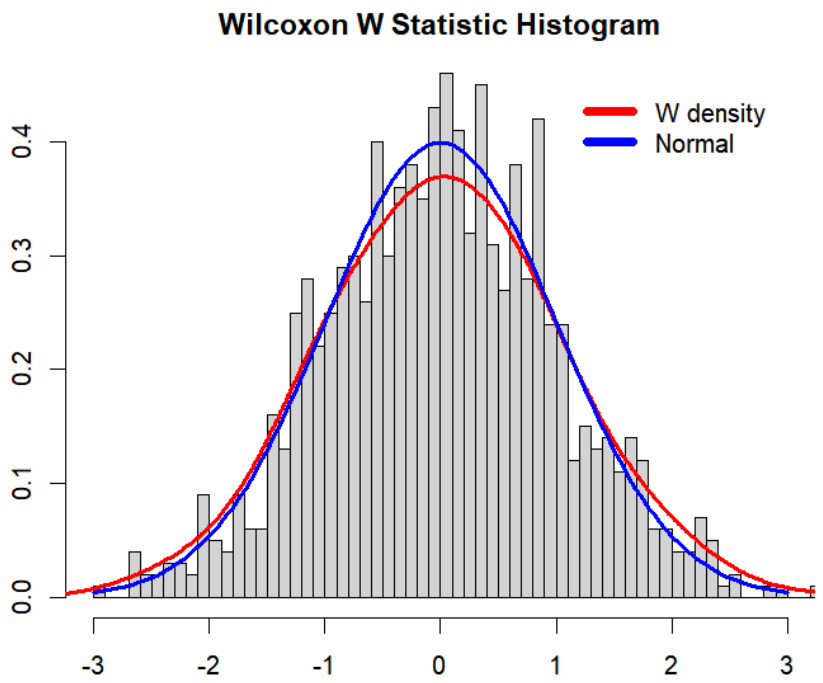
```
> # Wilcoxon test for random data around 0
> unifv <- (runif(nrows) - 0.5)
> wilcoxt <- wilcox.test(unifv)
> # Calculate V statistic of Wilcoxon test
> wilcoxt$statistic
> sum(rank(abs(unifv))[unifv>0])
> # Calculate W statistic of Wilcoxon test
> sum(sign(unifv)*rank(abs(unifv)))
> # Two sets of normal data
> sample1 <- rnorm(nrows)
> sample2 <- rnorm(nrows, mean=0.1)
> # Wilcoxon test
> wilcoxt <- wilcox.test(sample1, sample2, paired=TRUE)
> wilcoxt$statistic
> # Calculate V statistic of Wilcoxon test
> datav <- (sample1 - sample2)
> sum(rank(abs(datav))[datav>0])
> # Calculate W statistic of Wilcoxon test
> sum(sign(datav)*rank(abs(datav)))
```

draft: The Distribution of the W Statistic

The W statistic follows a distribution without a simple formula, which converges to the normal distribution for large sample size n, with an expected value equal to 0 and a variance equal to $\frac{n(n+1)(2n+1)}{6}$.

```
> # Calculate distributon of Wilcoxon W statistic
> wilcoxw <- sapply(1:1e3, function(x) {
+   datav <- (runif(nrows) - 0.5)
+   sum(sign(datav)*rank(abs(datav)))
+ })  # end sapply
> wilcoxw <- wilcoxw/sqrt(nrows*(nrows+1)*(2*nrows+1)/6)
> var(wilcoxw)
```

```
> hist(wilcoxw, col="lightgrey",
+      xlab="returns", breaks=50, xlim=c(-3, 3),
+      ylab="frequency", freq=FALSE,
+      main="Wilcoxon W Statistic Histogram")
> lines(density(wilcoxw, bw=0.4), lwd=3, col="red")
> curve(expr=dnorm, add=TRUE, lwd=3, col="blue")
> # Add legend
> legend("topright", inset=0.05, bty="n",
+  leg=c("W density", "Normal"),
+  lwd=6, lty=1, col=c("red", "blue"))
```



draft: The *Wilcoxon Signed Rank* Test Versus *Student's t-test*

The *Wilcoxon* test can be considered to be a *nonparametric* analogue of the *Student's t-test*, but it tests for the *medians*, rather than the *means*.

The *Wilcoxon* test is *nonparametric* because it doesn't assume any type of sample distribution, unlike the *Student's t-test* which assumes that the sample is taken from the *normal* distribution.

The *Wilcoxon* test is more *robust* with respect to data outliers because it only depends on the ranks of the sample differences $(x_i - y_i)$, not the differences themselves.

Therefore the *Wilcoxon* test reports fewer *false positive* cases when there are outliers.

For many distributions, the *Wilcoxon* test has greater *sensitivity* than the *Student's t-test*.

The *sensitivity* of a statistical test is the ability to correctly identify *true positive* cases (when the null hypothesis is `FALSE`).

```
> # Wilcoxon test for two normal distributions
> sample1 <- rnorm(1e2)
> sample2 <- rnorm(1e2, mean=0.1)
> wilcox.test(sample1, sample2, paired=TRUE)$p.value
> t.test(sample1, sample2)$p.value
> # Wilcoxon test with data outliers
> sample2 <- rnorm(1e2)
> sample2[1:3] <- sample2[1:3] + 1e3
> wilcox.test(sample1, sample2, paired=TRUE)$p.value
> t.test(sample1, sample2)$p.value
```

The *sensitivity* of a statistical test is equal to the *true positive* rate, i.e the fraction of `FALSE` null hypothesis cases that are correctly classified as `FALSE`.

The *specificity* of a statistical test is the ability to correctly identify *true negative* cases (when the null hypothesis is `TRUE`).

The *specificity* of a statistical test is equal to the *true negative* rate, i.e the fraction of `TRUE` null hypothesis cases that are correctly classified as `TRUE`.

draft: The *Mann-Whitney* Test for Distribution Similarity

The *Mann-Whitney* test *null hypothesis* is that the two samples, x_i and y_i, were obtained from probability distributions with the same median (location).

The function `wilcox.test()` with parameter `paired=FALSE` (the default) calculates the *Mann-Whitney* test statistic and its p-value.

Let $\{x_1, \ldots, x_{n1}\}$ and $\{y_1, \ldots, y_{n2}\}$ be two samples of data, and let $\{z_1, \ldots, z_n\}$ be the combined data sample, with $n = n1 + n2$.

In contrast to the *Wilcoxon Signed Rank* test, the two samples don't have to be of equal length ($n1 \neq n2$).

The *Mann-Whitney* test is also known as the *Wilcoxon Rank Sum* test, or the *Mann-Whitney-Wilcoxon* test.

```
> # Data samples
> datav <- sort(rnorm(38))
> indeks <- c(1:9, 20:29)
>
> # Or
> datav <- sort(rnorm(398))
> indeks <- c(1:99, 200:299)
> sample1 <- datav[indeks]
> sample2 <- datav[-indeks]
>
> # Or
> indeks <- sample(1:nrows, size=nrows/2)
> sample1 <- retp[indeks]
> sample2 <- (-retp[-indeks])
>
>
> sample1 <- rpois(1e2, lambda=4)
> sample1 <- rnorm(1e2)
> sample1 <- (sample1- median(sample1))
> sample2 <- runif(1e2)
> sample2 <- (sample2- median(sample2))
> moments::moment(sample1, order=3)
> moments::moment(sample2, order=3)
>
> # Mann-Whitney test for normal distribution
> wilcox.test(sample1, sample2, paired=FALSE)
> wilcox.test(sample1, sample2, paired=TRUE)
> blue <- rgb(0, 0, 1, alpha=0.5)
> red <- rgb(1, 0, 0, alpha=0.5)
> barplot(sample2, col=red)
> barplot(sample1, col=blue, add=TRUE)
> hist(sample1)
>
>
> # Mann-Whitney test for normal distribution
> datav <- rnorm(nrows, sd=100)
> wilcox.test(datav, paired=FALSE)
> # Skewed distribution with mean=0
```

draft: The U Statistic of the *Mann-Whitney* Test

Let z_i be the combined data of two sub-samples: x_i and y_i, with $n = n1 + n2$.

Let $r_i = \text{rank}(z_i)$ be the ranks of the sample, and r_i^1 and r_i^2 be the ranks of the sub-samples.

The sum of all the ranks is equal to:

$$\sum_{i=1}^{n} r_i = \sum_{i=1}^{n1} r_i^1 + \sum_{i=1}^{n2} r_i^2 = \frac{n(n+1)}{2}$$

The *Mann-Whitney* test statistics $U1$ and $U2$ are equal to the sum of the ranks minus a term accounting for the size of the samples:

$$U1 = \sum_{i=1}^{n1} r_i^1 - \frac{n1(n1+1)}{2}$$

$$U2 = \sum_{i=1}^{n2} r_i^2 - \frac{n2(n2+1)}{2}$$

So that $U1 + U2 = n1n2$.

The U statistic follows a distribution without a simple formula, which converges to the normal distribution for large sample size n.

In the extreme case when the ranks of the first sample are all greater than the second sample ($r_i^1 > r_j^2$), then $U2 = 0$ and $U1 = n1n2$.

```
> # Data samples
> sample1 <- rnorm(200)
> sample2 <- rnorm(100, mean=0.1)
> # Mann-Whitney-Wilcoxon rank sum test
> wilcoxt <- wilcox.test(sample1, sample2,
+                  paired=FALSE)
> wilcoxt$statistic
> # Calculate U statistics of Mann-Whitney-Wilcoxon test
> datav <- c(sample1, sample2)
> ranks <- rank(datav)
> sum(ranks[1:200]) - 100*201
> sum(ranks[201:300]) - 50*101
```

draft: The *U* Statistic of the *Wilcoxon Rank Sum* Test

The *Wilcoxon Rank Sum* test statistic U is equal to the sum of the ranks $r_i = \text{rank}(|x_i - y_i|)$ of the absolute differences weighted by their signs:

$$U = \sum_{i=1}^{n} \text{sgn}(x_i - y_i) r_i$$

The statistic U follows a distribution without a simple formula, which converges to the normal distribution for large sample size n, with an expected value equal to 0 and a variance equal to $\frac{n(n+1)(2n+1)}{6}$.

The *Wilcoxon Rank Sum* test is *nonparametric* because it doesn't assume any type of sample distribution, unlike the *Student's t-test* which assumes that the sample is taken from the *normal* distribution.

The *Wilcoxon Rank Sum* test is more *robust* with respect to data outliers because it only depends on the ranks of the sample differences $(x_i - y_i)$, not the differences themselves.

```
> # Wilcoxon test for random data around 0
> nrows <- 1e3
> datav <- (runif(nrows) - 0.5)
> wilcoxt <- wilcox.test(datav)
> # Calculate V statistic of Wilcoxon test
> wilcoxt$statistic
> sum(rank(abs(datav))[datav>0])
> # Calculate W statistic of Wilcoxon test
> sum(sign(datav)*rank(abs(datav)))
> # Calculate distributon of Wilcoxon W statistic
> wilcoxw <- sapply(1:1e3, function(x) {
+   datav <- (runif(nrows) - 0.5)
+   sum(sign(datav)*rank(abs(datav)))
+ })  # end sapply
> wilcoxw <- wilcoxw/sqrt(nrows*(nrows+1)*(2*nrows+1)/6)
> var(wilcoxw)
> hist(wilcoxw)
```

The function `wilcox.test()` returns the V statistic, not the the W statistic:

$$V = \sum_{i=1}^{n} \text{H}(x_i - y_i) r_i$$

Where $\text{H}(x) = 1$ if $x > 0$, and 0 otherwise.

The *Kruskal-Wallis* Test for the Distribution Similarity

The *Kruskal-Wallis* test is designed to test the *null hypothesis* that sub-samples of the data corresponding to different categories follow similar distributions.

The *Kruskal-Wallis* test can be used as a type of *nonparametric ANOVA* test, to test if the sub-samples of the data have the same mean.

For example, given the heights of several different species of trees, the *Kruskal-Wallis* test can test if all the species have the same height.

The *Kruskal-Wallis* test can also test if samples have different *skewness*, even if they have the same *means*.

The function `kruskal.test()` accepts a vector of sample data and a factor specifying the categories, and calculates the *Kruskal-Wallis* statistic and its *p*-value.

The function `kruskal.test()` can also accept the data as a formula combined with a matrix or data frame.

```
> # iris data frame
> aggregate(Sepal.Length ~ Species, data=iris,
+   FUN=function(x) c(mean=mean(x), sd=sd(x)))
> # Kruskal-Wallis test for iris data
> ktest <- kruskal.test(Sepal.Length ~ Species, data=iris)
> str(ktest)
> ktest$statistic
> # Kruskal-Wallis test for independent normal distributions
> sample1 <- rnorm(1e3)
> sample2 <- rnorm(1e3)
> groupv <- c(rep(TRUE, 1e3), rep(FALSE, 1e3))
> kruskal.test(x=c(sample1, sample2), g=groupv)
> # Kruskal-Wallis test for shifted normal distributions
> kruskal.test(x=c(sample1+1, sample2), g=groupv)
> # Kruskal-Wallis test for beta distributions
> sample1 <- rbeta(1e3, 2, 8) + 0.3
> sample2 <- rbeta(1e3, 8, 2) - 0.3
> mean(sample1); mean(sample2)
> kruskal.test(x=c(sample1, sample2), g=groupv)
> # Plot the beta distributions
> x11()
> plot(density(sample1), col="blue", lwd=3,
+      xlim=range(c(sample1, sample2)), xlab="samples",
+      main="Two samples from beta distributions with equal means")
> lines(density(sample2), col="red", lwd=3)
```

The *Kruskal-Wallis* Test Statistic

Given a data sample x_i with n elements, and a factor of k categories, the sample can be divided into k sub-samples x_i^j.

Let $r_i = \text{rank}(x_i)$ be the ranks of the sample, and r_i^j be the ranks of the sub-samples.

The *Kruskal-Wallis* test statistic H is proportional to the sum of squared differences between the average rank of the sample $\bar{r} = \frac{n+1}{2}$, minus the average ranks of the sub-samples $\bar{r}_j$:

$$H = \frac{12}{n(n+1)} \sum_{j=1}^{k} \left(\frac{n+1}{2} - \bar{r}_j\right)^2 n_j$$

Where the sum is over all the k categories, and n_j is the number of elements in sub-sample j.

The H statistic follows a distribution without a simple formula, which is approximately equal to the *chi-squared* distribution with $k-1$ degrees of freedom.

```
> # Kruskal-Wallis test for iris data
> ktest <- kruskal.test(Sepal.Length ~ Species, data=iris)
> # Calculate Kruskal-Wallis test Statistic
> nrows <- NROW(iris)
> irisdf <- data.frame(ranks=rank(iris$Sepal.Length),
+                      species=iris$Species)
> kruskal_stat <- (12/nrows/(nrows+1))*sum(
+   aggregate(ranks ~ species, data=irisdf,
+       FUN=function(x) {NROW(x)*((nrows+1)/2 - mean(x))^2})[, 2])
> c(ktest=unname(ktest$statistic), k_stat=kruskal_stat)
```

The *Kruskal-Wallis* Test with Data Outliers

The *Kruskal-Wallis* test is *nonparametric* because it doesn't assume any type of sample distribution.

The *Kruskal-Wallis* test is also *robust* with respect to data outliers, since it only depends on the ranks of the sample.

When a few data outliers are added to the data, *Student's t-test* rejects the *null hypothesis* that the means are equal, but the *Kruskal-Wallis* test still accepts the *null hypothesis* that the distributions are similar.

```
> # Kruskal-Wallis test with data outliers
> sample1 <- rnorm(1e3)
> sample2 <- rnorm(1e3)
> sample2[1:11] <- sample2[1:11] + 50
> groupv <- c(rep(TRUE, 1e3), rep(FALSE, 1e3))
> kruskal.test(x=c(sample1, sample2), g=groupv)$p.value
> t.test(sample1, sample2)$p.value
```

draft: Robust Estimators of Dispersion

The *Median Absolute Deviation* (*MAD*) is a robust measure of dispersion (variability), defined using the *median* instead of the mean:

$$\text{MAD} = \text{median}(\text{abs}(x_i - \text{median}(\mathbf{x})))$$

The advantage of *MAD* is that it's always well defined, even for data that has infinite variance.

For normally distributed data, the *MAD* has a larger standard error than the standard deviation.

But for distributions with fat tails (like asset returns), the standard deviation has a larger standard error than the *MAD*.

The *MAD* for normally distributed data is equal to $\Phi^{-1}(0.75) \cdot \hat{\sigma} = 0.6745 \cdot \hat{\sigma}$.

The function `mad()` calculates the *MAD* and divides it by $\Phi^{-1}(0.75)$ to make it comparable to the standard deviation.

```
> nrows <- 1e3
> datav <- rnorm(nrows)
> sd(datav)
> mad(datav)
> median(abs(datav - median(datav)))
> median(abs(datav - median(datav)))/qnorm(0.75)
> # Bootstrap of sd and mad estimators
> bootd <- sapply(1:1e4, function(x) {
+   samplev <- datav[sample.int(nrows, replace=TRUE)]
+   c(sd=sd(samplev), mad=mad(samplev))
+ })  # end sapply
> bootd <- t(bootd)
> # Analyze bootstrapped variance
> head(bootd)
> sum(is.na(bootd))
> # Means and standard errors from bootstrap
> apply(bootd, MARGIN=2,
+ function(x) c(mean=mean(x), stderror=sd(x)))
> # Parallel bootstrap under Windows
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1  # Number of cores
> compclust <- makeCluster(ncores)  # initialize compute cluster
> bootd <- parLapply(compclust, 1:1e4,
+   function(x, datav) {
+     samplev <- datav[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   }, datav=datav)  # end parLapply
> stopCluster(compclust)  # Stop R processes over cluster
> # Parallel bootstrap under Mac-OSX or Linux
> bootd <- mclapply(1:1e4,
+   function(x) {
+     samplev <- datav[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   }, mc.cores=ncores)  # end mclapply
> # Means and standard errors from bootstrap
> bootd <- rutils::do_call(rbind, bootd)
> apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
```

draft: Robust Estimators of Skewness

The medcouple robust skewness estimator

The *Median Absolute Deviation* (*MAD*) is a robust measure of dispersion (variability), defined using the *median* instead of the mean:

$$\text{MAD} = \text{median}(\text{abs}(x_i - \text{median}(\mathbf{x})))$$

The advantage of *MAD* is that it's always well defined, even for data that has infinite variance.

For normally distributed data, the *MAD* has a larger standard error than the standard deviation.

But for distributions with fat tails (like asset returns), the standard deviation has a larger standard error than the *MAD*.

The *MAD* for normally distributed data is equal to $\Phi^{-1}(0.75) \cdot \hat{\sigma} = 0.6745 \cdot \hat{\sigma}$.

The function `mad()` calculates the *MAD* and divides it by $\Phi^{-1}(0.75)$ to make it comparable to the standard deviation.

```
> nrows <- 1e3
> datav <- rnorm(nrows)
> sd(datav)
> mad(datav)
> median(abs(datav - median(datav)))
> median(abs(datav - median(datav)))/qnorm(0.75)
> # Bootstrap of sd and mad estimators
> bootd <- sapply(1:1e4, function(x) {
+   samplev <- datav[sample.int(nrows, replace=TRUE)]
+   c(sd=sd(samplev), mad=mad(samplev))
+ })  # end sapply
> bootd <- t(bootd)
> # Analyze bootstrapped variance
> head(bootd)
> sum(is.na(bootd))
> # Means and standard errors from bootstrap
> apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
> # Parallel bootstrap under Windows
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1  # Number of cores
> compclust <- makeCluster(ncores)  # initialize compute cluster
> bootd <- parLapply(compclust, 1:1e4,
+   function(x, datav) {
+     samplev <- datav[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   }, datav=datav)  # end parLapply
> stopCluster(compclust)  # Stop R processes over cluster
> # Parallel bootstrap under Mac-OSX or Linux
> bootd <- mclapply(1:1e4, function(x) {
+     samplev <- datav[sample.int(nrows, replace=TRUE)]
+     c(sd=sd(samplev), mad=mad(samplev))
+   }, mc.cores=ncores)  # end mclapply
> # Means and standard errors from bootstrap
> bootd <- rutils::do_call(rbind, bootd)
> apply(bootd, MARGIN=2, function(x)
+   c(mean=mean(x), stderror=sd(x)))
```

Pearson Correlation

The *covariance* σ_{xy} between two sets of data, x_i and y_i, is defined as:

$$\sigma_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

Where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ are the *mean* values.

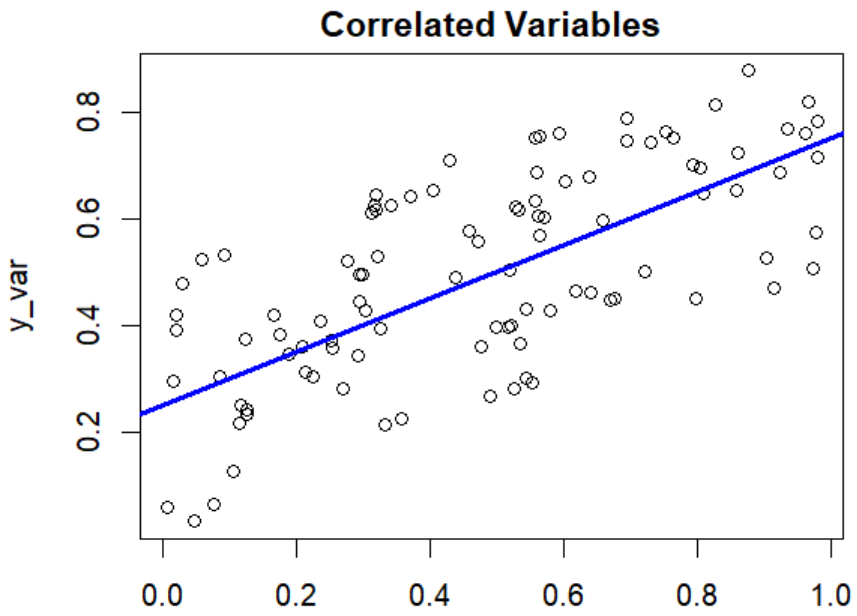
The function `cov()` calculates the covariance between two numeric vectors.

The *Pearson* correlation ρ_P is equal to the *covariance* divided by the standard deviations σ_x and σ_y:

$$\rho_P = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

The function `cor()` calculates the correlation between two numeric vectors.

Depending on the argument `"method"`, it calculates either the *Pearson* (default), *Spearman*, or *Kendall* correlations.




```
> # Initialize the random number generator
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> # Define variables and calculate correlation
> nrows <- 100
> xvar <- runif(nrows); yxvar <- runif(nrows)
> cor(xvar, yxvar)
> # Correlate the variables and calculate correlation
> rho <- 0.5
> yxvar <- rho*xvar + (1-rho)*yxvar
> # Plot in x11 window
> x11(width=5, height=4)
> # Set plot parameters to reduce whitespace around plot
> par(mar=c(4, 4, 2, 1), oma=c(0.5, 0.5, 0, 0))
> # Plot scatterplot and exact regression line
> plot(xvar, yxvar, xlab="xvar", ylab="yxvar")
> title(main="Correlated Variables", line=0.5)
> abline(a=0.25, b=rho, lwd=3, col="blue")
> # Calculate regression
```

draft: Standard Error of *Pearson* Correlation

See my comment to Jean Paul comment:
https://www.jstor.org/stable/2277400
https://www.tandfonline.com/doi/abs/10.1080/00220973.1956.11010555
https://stats.stackexchange.com/questions/73621/standard-error-from-correlation-coefficient/262893
https://stats.stackexchange.com/questions/226380/derivation-of-the-standard-error-for-pearsons-correlation-coefficient
https://stats.stackexchange.com/questions/154362/confidence-interval-on-point-biserial-correlation-coefficient

Let x be standard normally distributed with mean zero $E[x] = 0$ and standard deviation equal to one: $E[x^2] = 1$.

Let $y = \rho x + \varepsilon\sqrt{1-\rho^2}$, where ε is standard normally distributed, with zero correlation to x: $E[x\varepsilon] = 0$.

Then y is also standard normally distributed, with correlation to x equal to ρ:
$E[xy] = E[x(\rho x + \varepsilon\sqrt{1-\rho^2})] = E[\rho x^2] = \rho$.

The variance of the correlation is equal to:
$E[(xy-\rho)^2] = E[\rho^2 x^4 + 2x^3\varepsilon\rho\sqrt{1-\rho^2} + x^2\varepsilon^2(1-\rho^2) - 2\rho xy + \rho^2] = 1-\rho^2+\rho^2 E[x^4] = 3\rho^2+1-\rho^2-2\rho^2+\rho^2$.

$E[\rho^2 x^4 + 2x^3\varepsilon\rho\sqrt{1-\rho^2} + x^2\varepsilon^2(1-\rho^2)] = 1-\rho^2+3\rho^2$

The standard error of the correlation estimator is equal to: $\frac{1}{\sqrt{n-2}}$, and slowly decreases as the square root of n - the length of the data.

The function `cor.test()` performs a test of the

```
> # Simulation of sample correlation
> nrows <- 1e4
> rho <- 0.99
> rho2 <- sqrt(1-rho^2)
> datav <- sapply(1:1000, function(x) {
+   xvar <- rnorm(nrows)
+   yxvar <- (rho*xvar + rho2*rnorm(nrows))
+   cor(xvar, yxvar)
+ })  # end sapply
> sd(datav)
> # Correct formula
> (1-rho^2)/sqrt(nrows-2)
> # Incorrect formula
> sqrt((1-rho^2)/(nrows-2))
>
>
> # Correlation
> corv <- cor(xvar, yxvar)
> # Standard error of correlation
> stderror <- sqrt((1-corv^2)/(nrows-2))
> # t-value of correlation
> corv/stderror
> # 95% confidence intervals
> corv*c(1-qnorm(0.975)*stderror, 1+qnorm(0.975)*stderror)
>
>
> # Test statistical significance of correlation
> cor.test(xvar, yxvar)
>
> rho <- 0.9
> rho2 <- sqrt(1-rho^2)
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> # Bootstrap of sample mean and median
> bootd <- sapply(1:1000, function(x) {
+   xvar <- rnorm(nrows)
+   yxvar <- (rho*xvar + rho2*rnorm(nrows))
+   c(rho=mean(yxvar*xvar), y_sd=sd(yxvar), cor=cor(xvar, yxvar))
+ })  # end sapply
```

Spearman Rank Correlation

The *Spearman* correlation ρ_S is equal to the *Pearson* correlation between the ranks rx_i and ry_i of the variables x_i and y_i:

$$\rho_S = \frac{\sum_{i=1}^{n}(rx_i - \bar{rx})(ry_i - \bar{ry})}{(n-1)\sigma_{rx}\sigma_{ry}}$$

If the ranks are all distinct integers, then the *Spearman* correlation ρ_S can be expressed as:

$$\rho_S = 1 - \frac{6\sum_{i=1}^{n} dr_i^2}{n(n^2-1)}$$

Where $dr_i = rx_i - ry_i$ are the differences between the ranks.

The *Spearman* correlation is a *robust* measure of association because it depends on the ranks, so it's not sensitive to the extreme values of the variables x_i and y_i.

The *Spearman* correlation is considered a *nonparametric* estimator because it does not depend on the joint probability distribution of the variables x_i and y_i.

```
> # Calculate correlations
> cor(xvar, yxvar, method="pearson")
> cor(xvar, yxvar, method="spearman")
> # Test statistical significance of correlations
> cor.test(xvar, yxvar, method="pearson")
> cor.test(xvar, yxvar, method="spearman")
```

Kendall's τ Correlation

The pair of observations $\{x_i, y_i\}$ is concordant with the pair $\{x_j, y_j\}$ if the signs of the differences $(rx_i - rx_j)$ and $(ry_i - ry_j)$ are the same, i.e. if their ranks follow the same order.

The *Kendall* correlation τ_K (*Kendall's* τ) is equal to the difference between the number of concordant pairs of observations, minus the number of discordant pairs:

$$\tau_K = \frac{2}{n(n-1)} \sum_{i<j}^{n} \operatorname{sgn}(rx_i - rx_j) \operatorname{sgn}(ry_i - ry_j)$$

The *Kendall* correlation τ_K is also a *robust* and *nonparametric* estimator of association, because it only depends on the ranks, so it's not sensitive to the extreme values of the variables x_i and y_i.

```
> # Calculate correlations
> cor(xvar, yxvar, method="pearson")
> cor(xvar, yxvar, method="kendall")
> # Test statistical significance of correlations
> cor.test(xvar, yxvar, method="pearson")
> cor.test(xvar, yxvar, method="kendall")
```