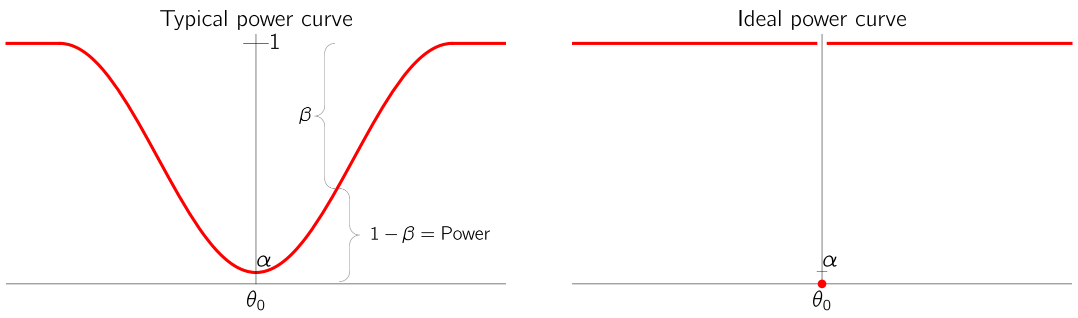
Tutorial - Week 4

Please read the related material and attempt these questions before attending your allocated tutorial. Solutions are released on Friday 4pm.

A large part of this course is concerned with trying to argue that various new and improved techniques (that use random matrix theory) are better than the well-known "classic" multivariate methods used in statistics **[A]** and machine learning **[B]**. At the core of most methods is the presence of hypothesis testing problem and we are trying to compare various methods/tests in regards to their *power*, the probability under repeated sampling of correctly rejecting an incorrect hypothesis, subject to some pre-specified *size*[1], the probability of incorrectly rejecting a true hypothesis.
More precisely, suppose we have a parameter space Θ and consider hypotheses of the form

$$H_0 : \theta \in \Theta_0 \qquad \text{vs.} \qquad H_1 : \theta \in \Theta_1,$$

where Θ_0 and Θ_1 are two disjoint subsets of Θ, possibly, but not necessarily, satisfying $\Theta_0 \cup \Theta_1 = \Theta$. For example, the parameter space could consist of the mean μ with $\Theta_0 = \{\mu : \mu = \mu_0\}$ and $\Theta_1 = \{\mu : \mu \neq \mu_0\}$. We call H_0 the *null hypothesis* and H_1 the *alternate hypothesis* (sometimes written H_a). A *Type I error* is made if H_0 is rejected when H_0 is true. The *probability of a Type I error* is denoted by α and is called the *size* of the test. A *Type II error* is made if H_0 is accepted when H_1 is true. The *probability of a type II error* is denoted by β. If θ_1 is the value of θ in the alternative hypothesis H_1, then the *power* of the test is $1-\beta$.
If $\Theta_0 = \{\theta : \theta = \theta_0\}$ and we plot power for varying $\theta \in \Theta$, we get the following typical power curve that we might observe compared to an ideal (i.e., best possible) power curve (1 everywhere except for $\theta = \theta_0$ where it is zero).



A test with a high statistical power means we have a small risk of committing Type II errors (false negatives) so, ideally, we want choose the method/test that has the highest power. In other words, the faster the power $1-\beta$ increases as you move away from θ_0 the better your method is, with the extreme example being the ideal power curve in the figure.

[1]size is aka. significance level, level, probability of a Type I error, or α.

A crucial ingredient to graphing such a plot for a given method is to be able to compute the value of $\beta(\theta)$ for every $\theta \in \Theta$. In some special cases[2] a closed-form solution can be found for simple distributions and simple parameters θ; see **[C]** for a refresher. Unfortunately, in this course, closed-form solutions are typically unavailable and we need to resort to a *simulation* approach; see **[D]** and **[E]**. For the case $H_0 : \theta = \theta_0$ vs. $H_1 : \theta \neq \theta_0$, the essence of the simulation approach is:

- To evaluate whether the *size* of a test achieves advertised α, sample data under $\theta = \theta_0$ and calculate the proportion of rejections of H_0. This approximates the true probability of rejecting H_0 when it is true. The proportion of rejections should be $\approx \alpha$.
- To evaluate *power*, sample data under some alternative $\theta \neq \theta_0$ and calculate the proportion of rejections of H_0. The approximates the true probability of rejecting H_0 when the alternative is true. The proportion of rejections ρ should be $\approx \beta$, then power is $\approx 1 - \rho$.

Careful, if the actual size of the test is $> \alpha$, then the evaluation of power is flawed. This is because a good test is one that makes $1 - \beta(\theta)$ as large as possible for θ on Θ_1 while satisfying the constraint $1 - \beta(\theta) \leq \alpha$ for all $\theta \in \Theta_0$; see ideal figure above for the case $\Theta_0 = \{\theta : \theta = \theta_0\}$ and $\Theta_1 = \{\theta : \theta \neq \theta_0\}$.

Question 1

We consider a classic multivariate dataset of skull sizes. The data was collected in southeastern and eastern Tibet. Skulls 1-17 were found in one place (Sample 1) and skulls 18-32 were found in another place (Sample 2). On each skull, we have $p = 5$ measurements, all in millimeters. They are (1) `Length`: greatest length of the skull, (2) `Breadth`: greatest horizontal breadth of skull, (3) `Height`: height of the skull, (4) `Fheight`: upper face height, (5) `Fbreadth`: face breadth, between outermost points of cheek bones. An anthropologist would be interested to know if these skulls come from the same population or two different populations.

A standard way to answer this question is to use a multivariate generalisation of the Student's t-statistic called *Hotelling's T^2 statistic* which is given by

$$T^2 = n(\bar{\mathbb{x}} - \mu)\mathbb{S}^{-1}(\bar{\mathbb{x}} - \mu),$$

where $\mathbb{S}$ is the sample covariance, $\bar{\mathbb{x}}$ is the sample mean, and μ is the population mean; see **[F]**. Under the assumption that $\mathbb{x}_1, \dots, \mathbb{x}_n$ is a sample from $N_p(\mu_0, \Sigma)$, then the distribution of

$$\frac{(n-p)T^2}{p(n-1)}$$

is the non-central F-distribution with parameter p, $n - p$ degrees of freedom, and non-centrality parameter $\Delta = n(\mu - \mu_0)'\Sigma^{-1}(\mu - \mu_0)$. If $\mu = \mu_0$ then the F-distribution is central.

(a) Load the `tibetskull.dat` into R and plot the data in various ways so you get familiar with the data.

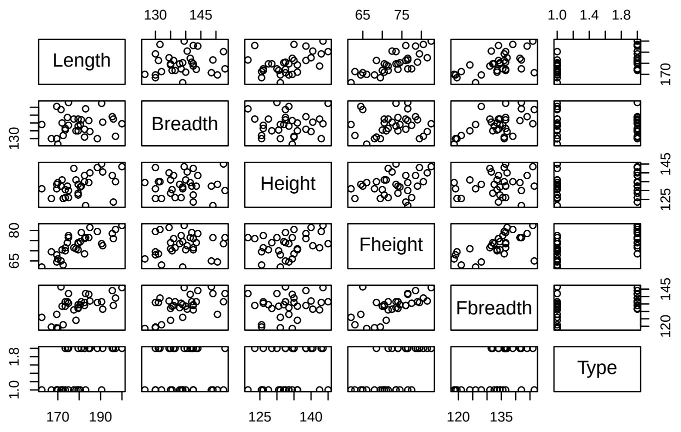
[2]Remember this from STAT2001?

Solution:

```
source('tibetskull.dat')
```

A "pairs" scatterplot matrix is really useful on multivariate datasets (since it's built into R) to spot correlations and other patterns. For example, you can see that there is "Type" field that is binary (1 or 2).

```
pairs(Tibet)
```



Let's remove the "Type" field because it's not a continuous value, it is the sample label. We store it in it's own variable as a vector.

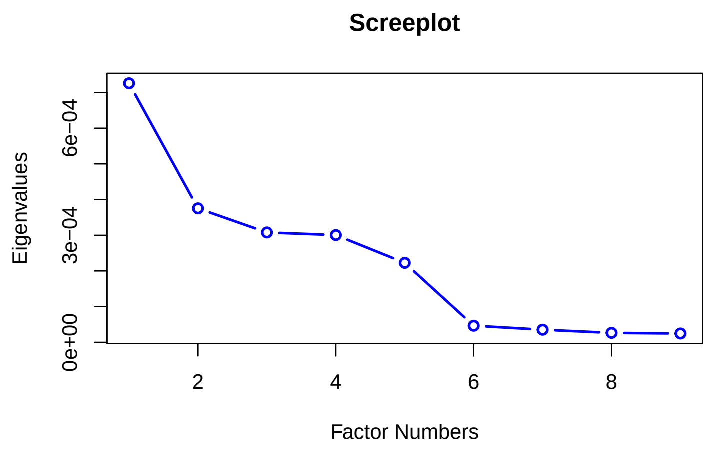
```
Sample = as.numeric(Tibet[,"Type"])
```

We then edit the `Tibet` variable by using the "!" operator to do the opposite of the logical condition, then we save the result back into the variable `Tibet`.

```
Tibet = Tibet[,!(colnames(Tibet) %in% c('Type'))]
```

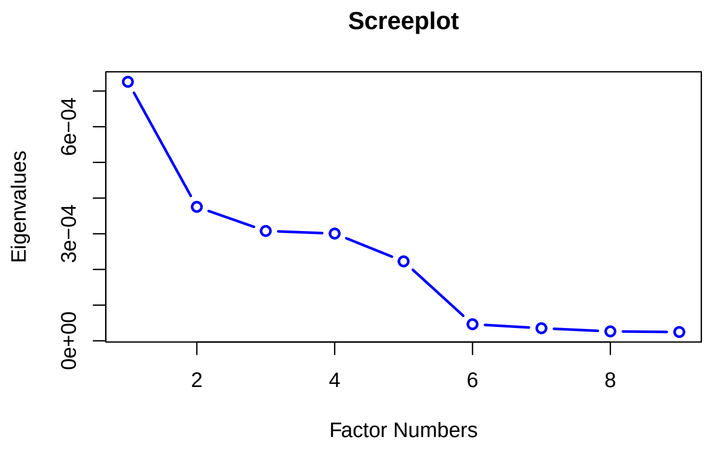
Much better:

```
pairs(Tibet)
```



A boxplot is also useful.

```
boxplot(Tibet)
```

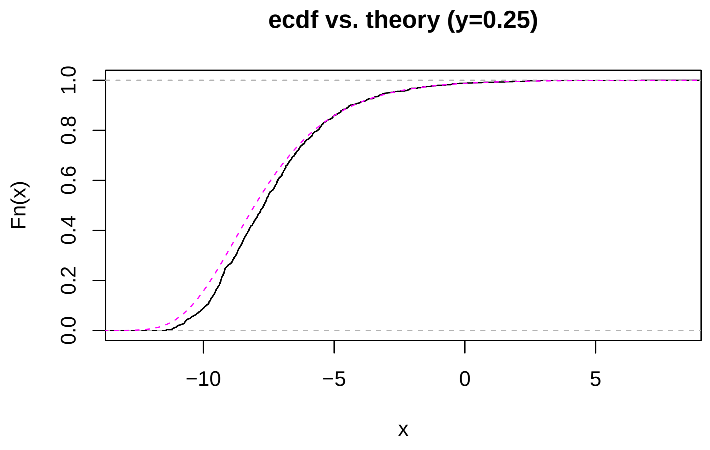


When the dimension gets large, I am a big fan of the parallel coordinate plot. We can use `Sample` to color the two samples differently. The `MASS` library has a parallel coordinate plot function called `parcoord()`.

```
library(MASS)
```

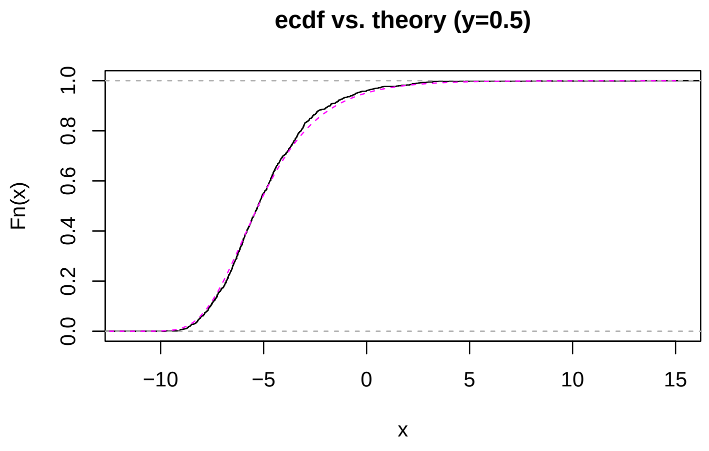
No colors applied.

```
parcoord(Tibet)
```



Now with coloring based on the sample. Often, this plot allows us to spot features/covariates that can help separate two classes/samples.

```
parcoord(Tibet, col=Sample)
```



(b) To illustrate the Hotelling method, consider the (artificial) example whereby we perform a one-sample test $H_0 : \mu = \mu_0$ versus $H_1 : \mu \neq \mu_0$ where $\mu_0 = (180, 140, 135, 74, 135)'$. Do you reject or accept H_0?

Solution: We now perform a one-sample test for mean. First, we set the hypothesized value for the mean.

```
mu.0 = c(180, 140, 135, 74, 135)
```

We extract the dimensions of the data.

```
n = nrow(Tibet)
p = ncol(Tibet)
```

We calculate the sample mean.

```
x.bar = colMeans(Tibet)
x.bar
```

```
##   Length  Breadth   Height  Fheight Fbreadth
## 179.9375 139.0625 133.2969  72.9375 133.7031
```

We compute the sample covariance matrix.

```
S = var(Tibet)
S
```

```
##             Length    Breadth     Height   Fheight Fbreadth
## Length   87.705645  6.9798387 24.4223790 38.100806 39.50504
## Breadth   6.979839 46.8024194  0.6018145  3.141129 27.93851
## Height   24.422379  0.6018145 36.9977319  9.406250  9.05872
## Fheight  38.100806  3.1411290  9.4062500 29.060484 24.77117
## Fbreadth 39.505040 27.9385081  9.0587198 24.771169 55.41709
```

We compute the inverse of sample covariance matrix.

```
S.inverse = solve(S)
S.inverse
```

```
##                Length       Breadth        Height      Fheight      Fbreadth
## Length    0.031226415  0.0021331248 -0.0108588043 -0.031144732 -0.0076391385
## Breadth   0.002133125  0.0360925419  0.0005101477  0.016179063 -0.0270320336
## Height   -0.010858804  0.0005101477  0.0332994720  0.002688476  0.0008386848
## Fheight  -0.031144732  0.0161790629  0.0026884755  0.098593863 -0.0304650517
## Fbreadth -0.007639139 -0.0270320336  0.0008386848 -0.030465052  0.0505994969
```

We compute the T^2 statistic.

```
T2 = n * t(x.bar - mu.0) %*% S.inverse %*% (x.bar - mu.0)
T2
```

```
##          [,1]
## [1,] 6.880282
```

We perform the appropriate transformation to get the F-distribution and compute the p-value. Under the null hypothesis, this statistic has a $F(p, n-p)$ distribution. We use `lower=FALSE` to get the upper tail of the distribution.

```
p.value = pf((n - p) / ((n - 1) * p) * T2, p, n - p, lower = FALSE)
p.value
```

```
##           [,1]
## [1,] 0.3363598
```

At this probability level, we do *not* reject the null hypothesis.

(c) Perform the two-sample test

$$H_0 : \mu_1 = \mu_2 \qquad \text{vs.} \qquad H_1 : \mu_1 \neq \mu_2,$$

where μ_k is the population mean of Sample k. You can consult **[F]** for the derivation of the two-sample test. Do you reject or accept H_0? What could an anthropologist conclude?

Solution: We start by breaking up the `Tibet` data into two samples (based on `Sample`).

```
Tibet1 = Tibet[Sample==1,]
Tibet2 = Tibet[Sample==2,]
```

We extract the dimensions.

```
n1 = nrow(Tibet1)
n2 = nrow(Tibet2)
n  = n1+n2
p  = ncol(Tibet1)
```

We compute the sample mean vectors.

```
x.bar = colMeans(Tibet1)
y.bar = colMeans(Tibet2)
```

We compute pooled estimate for the covariance matrix and its inverse.

```
S.pooled = ((n1 - 1) * var(Tibet1) + (n2 - 1) * var(Tibet2)) / (n - 2)
S.pooled.inverse <- solve(S.pooled)
```

We compute the sample version of the Mahalonobis distance (squared).

```
D2 <- t(x.bar - y.bar) %*% S.pooled.inverse %*% (x.bar - y.bar)
```

We compute the T^2 statistic.

```
T2 = n1 * n2 / n * D2
```

We appropriately scale the T^2 statistic so that under the null hypothesis, this (scaled) statistic has a $F(p, n-1-p)$ distribution. We then compute the p-value.

```
p.value = pf((n - 1 - p) / ((n - 2) * p) * T2, p, n - 1 - p, lower = FALSE)
p.value
```

```
##             [,1]
## [1,] 0.002936228
```

At this p-value, we reject the null hypothesis.

Question 2

Let's consider the performance of the Hotelling T^2 test.

(a) Write a function to compute the power of the Hotelling T^2 test using the "closed-form" functions `qf` and `pf` in R. Plot the power function in the case where $\alpha = 0.05$, $p = 10$, $\Sigma = I_p$, $n_1 = 50$, $n_2 = 50$, over the alternative $\mu_1 - \mu_2 = (\delta, \delta, \dots, \delta)'$ with $-0.5 \leq \delta \leq 0.5$. Add a horizontal line at height α on the same plot.

Solution: We create our power function by first, under the assumption that the null hypothesis holds, calculating the critical x value such that the area under the F-distribution probability density up to x is equal to $1-\alpha$, i.e., $\mathbf{P}(F \leq x) = 1-\alpha$. We then use this critical x to calculate the p-value but this time under the *alternate hypothesis* which has a non-centrality parameter Δ.

```r
power.hotellingT2 = Vectorize(function(delta, n1, n2, Sigma, alpha=0.05) {
  d = rep(delta, p)
  Delta = (n1 * n2) / (n1 + n2) * t(d) %*% solve(Sigma) %*% d
  x = qf(1 - alpha, p, n1 + n2 - p - 1)
  1 - pf(x, p, n1 + n2 - p - 1, Delta)
}, "delta")
```

We test it.

```r
delta = 0.01
p = 10
n1 = 50
n2 = 50
Sigma = diag(rep(1, p))

power.hotellingT2(delta, n1, n2, Sigma)
```

```
## [1] 0.05064452
```

Note that in the function above, I didn't use the `lower` option in `qf` and `pf` so I need to adjust accordingly. Here is an alternative version.

```r
power.hotellingT2.alt = Vectorize(function(delta, n1, n2, Sigma, alpha=0.05) {
  d = rep(delta, p)
  Delta = (n1 * n2) / (n1 + n2) * t(d) %*% solve(Sigma) %*% d
  x = qf(alpha, p, n1 + n2 - p - 1, lower=FALSE)
  pf(x, p, n1 + n2 - p - 1, Delta, lower=FALSE)
}, "delta")
```

We test it by comparing it to the other version and see we get the same answer.

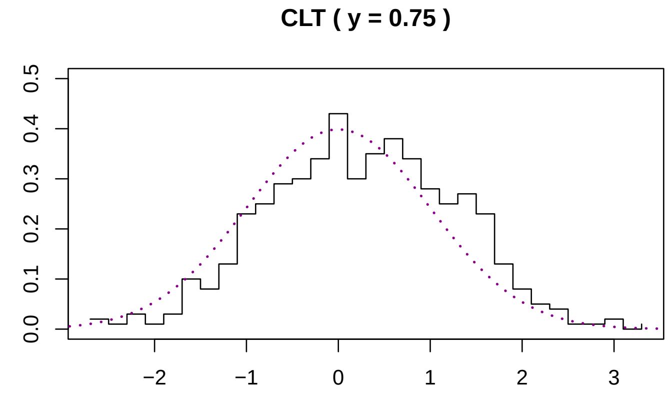
```r
delta = 0.01
p = 10
n1 = 50
n2 = 50
Sigma = diag(rep(1, p))

c(power.hotellingT2(delta, n1, n2, Sigma), power.hotellingT2.alt(delta, n1, n2, Sigma))
```

```
## [1] 0.05064452 0.05064452
```

Notice that I wrapped the above functions in a `Vectorize` function? This is so that the first parameter of the function can be passed in as a vector and the function will be applied to each value of that vector. This is useful for plotting as we can pass in a vector of `deltas`.

```r
alpha = 0.05
deltas = seq(-0.5, 0.5, length.out=40)
# do plot without plotting to set ranges of plot and labels
plot(deltas, power.hotellingT2(deltas, n1, n2, Sigma), type='n',
     ylab=expression(1-beta), xlab=expression(delta))
# Now plot again
abline(h=alpha, col='lightgray')
lines(deltas, power.hotellingT2(deltas, n1, n2, Sigma), type='l', ylab=NA, lwd=2, col='red')
```



(b) Write a function to compute the power of the Hotelling T^2 test using a simulation approach under the same assumptions as (a). Using $m = 5000$ simulations for each δ, plot the closed-form and simulation power curves for $0 \leq \delta \leq 0.5$ on the same figure. Do they match? If not, explain why not.

Solution: We start by writing a function that takes two data matrices (X and Y) and calculates the p-value for Hotelling's T^2 test. We will need this for our simulation.

```
hotellingT2 = function(X, Y) {
  n1 = nrow(X)
  n2 = nrow(Y)
  x.bar = colMeans(X)
  y.bar = colMeans(Y)
  S.X = cov(X)
  S.Y = cov(Y)
  S.pooled = ((n1 - 1) * S.X + (n2 - 1) * S.Y) / (n1 + n2 - 2)
  d = x.bar - y.bar
  T2 = (n1 * n2) / (n1 + n2) * t(d) %*% solve(S.pooled) %*% d
  pf((n1 + n2 - p - 1) / ((n1 + n2 - 2) * p) * T2, p, n1 + n2 - 1 - p, lower.tail = FALSE)
}
```

We can check it on the data from Question 1.

```
hotellingT2(Tibet1, Tibet2)
```

```
##            [,1]
## [1,] 0.09433219
```

To perform simulations, we need to sample from a multivariate normal distribution. You could use the function `mvrnorm` from the `MASS` library, the `rmvnorm` function from the `mvtnorm` library, or the `rmvn` function from the (new) library `mvnfast` which can speed up the sampling if you have multiple processors. I will use the later library.

```r
library(mvnfast)
```

We sample data under the alternative ($\mu_1 \neq \mu_2$ with $\mu_1 - \mu_2 = (\delta, \dots, \delta)'$) by setting μ_1 to be a zero vector and μ_2 to be a vector of δ's. We compute the p-value every time we sample, then we calculate the mean number of times that the p-value exceeds α. This gives an estimate of β. We set `nsims` to be the number of simulations (replications) that we do of our experiment.

```r
#set.seed(234234)

nsims = 1000

delta = 0.02
alpha = 0.05

p = 10
n1 = 50
n2 = 50

mu.1 = rep(0, p)
mu.2 = rep(delta, p)
Sigma = diag(rep(1, p))

pvalues = replicate(nsims, {
  X = rmvn(n1, mu.1, Sigma)
  Y = rmvn(n2, mu.2, Sigma)
  hotellingT2(X, Y)
})
mean.beta = mean(pvalues > alpha)
cat("The test power is", (1 - mean.beta),"\n")
```

```
## The test power is 0.062
```

Using the same parameters, we compare to our “closed-form” function.

```r
power.hotellingT2(delta, n1, n2, Sigma)
```

```
## [1] 0.05260095
```

Try running the simulation again, you will notice that every time you run it you will get a different answer unless you use `set.seed`. If you increase the `nsims`, the value should approach the true power value given by `power.hotellingT2`.

We can now write a function that performs a simulation to calculate the power.

```r
sim.power.hotellingT2 = Vectorize(function(delta, n1, n2, Sigma, alpha=0.05, nsims=1000) {
  p = dim(Sigma)[1]
  d = rep(delta, p)
  mu.1 = rep(0, p)
  mu.2 = rep(delta, p)
  pval = replicate(nsims, {
    X = rmvn(n1, mu.1, Sigma)
    Y = rmvn(n2, mu.2, Sigma)
    hotellingT2(X, Y)
  })
  1-mean(pval > alpha)
}, "delta")
```

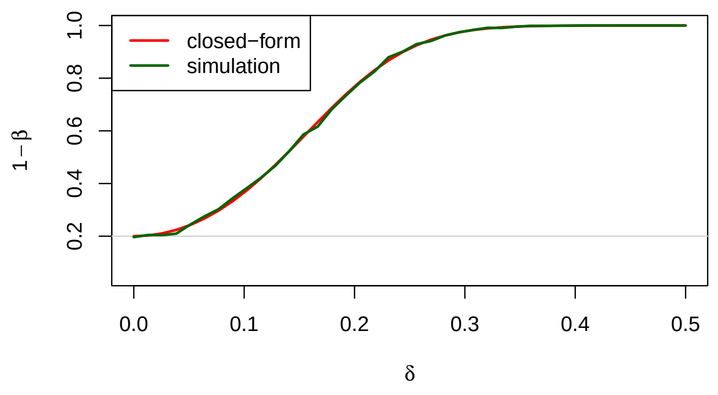
We compare the closed-form and the simulation.

```r
delta = 0.02
c(power.hotellingT2(delta, n1, n2, Sigma), sim.power.hotellingT2(delta, n1, n2, Sigma))
```

```
## [1] 0.05260095 0.06100000
```

We now plot a power curve using the simulation version of the function. Note that this can take quite a bit of time on a slow machine as it needs to perform a simulation for each possible value of δ that we plot on the curve (here, 40 values). Since the power function is symmetric around 0, we only plot $0 \leq \delta \leq 0.5$.

```r
alpha = 0.2
nsims = 5000
deltas = seq(0, 0.5, length.out=40)
# do plot without plotting to set ranges of plot and labels
plot(deltas, power.hotellingT2(deltas, n1, n2, Sigma), type='n',
     ylab=expression(1-beta), xlab=expression(delta))
# now do it properly
abline(h=alpha, col='lightgray')
lines(deltas, power.hotellingT2(deltas, n1, n2, Sigma, alpha), type='l', ylab=NA, lwd=2, col="red")
lines(deltas, sim.power.hotellingT2(deltas, n1, n2, Sigma, alpha, nsims), type='l', ylab=NA, lwd=2, col='
legend("topleft", c("closed-form", "simulation"), col=c("red", "darkgreen"), lty=1, lwd=2)
```



(c) Redo (b) with $m = 10000$. What does it change?

Solution: Unfortunately, simulations are quite computationally demanding.
One way to accelerate this simulation is to do it the replications in parallel across the number of cores in your machine. The library `future.apply` makes this easy.

```
#install.packages("future.apply")
```

We can create a new version of the function where we replace `replicate` by `future_replicate`.

```
library(future.apply)
```

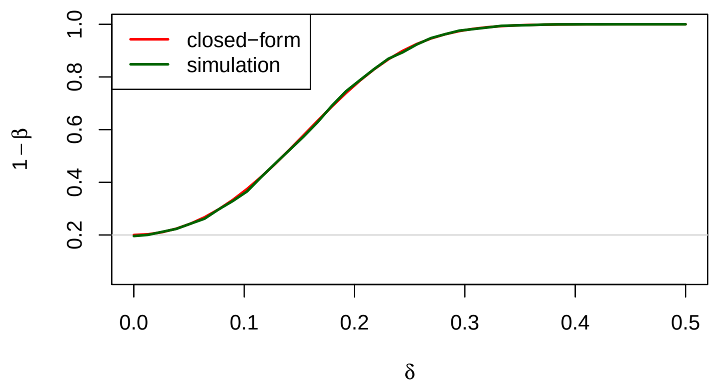
```
## Loading required package: future
```

```
psim.power.hotellingT2 = Vectorize(function(delta, n1, n2, Sigma, alpha=0.05, nsims=1000) {
  p = dim(Sigma)[1]
  d = rep(delta, p)
  mu.1 = rep(0, p)
  mu.2 = rep(delta, p)
  pval = future_replicate(nsims, {
    X = rmvn(n1, mu.1, Sigma)
    Y = rmvn(n2, mu.2, Sigma)
    hotellingT2(X, Y)
  })
  1-mean(pval > alpha)
}, "delta")
```

We can then use the `plan` function to set the number of cores that are used. Make sure you use the `psim.power.hotellingT2` version that you just created. I have a big machine so I am using 36 cores (change yours appropriately). This allows us to increase the number of simulations to 10000.
We expect a better match between the simulation and closed-form curves.

```r
plan(multisession, workers = 36)

alpha = 0.2
nsims = 10000
deltas = seq(0, 0.5, length.out=40)
# do plot without plotting to set ranges of plot and labels
plot(deltas, power.hotellingT2(deltas, n1, n2, Sigma), type='n',
     ylab=expression(1-beta), xlab=expression(delta))
# now do it properly
abline(h=alpha, col='lightgray')
lines(deltas, power.hotellingT2(deltas, n1, n2, Sigma, alpha), type='l', ylab=NA, lwd=2, col="red")
lines(deltas, psim.power.hotellingT2(deltas, n1, n2, Sigma, alpha, nsims), type='l', ylab=NA, lwd=2, col='
legend("topleft", c("closed-form", "simulation"), col=c("red", "darkgreen"), lty=1, lwd=2)
```



References

[A] Anderson (2003). An Introduction to Multivariate Statistical Analysis. Wiley.
[B] James, Witten, Hastie, Tibshirani (2013). An Introduction to Statistical Learning. Springer.
[C] https://online.stat.psu.edu/stat415/lesson/25/25.2
[D] http://www4.stat.ncsu.edu/~davidian/st810a/simulation_handout.pdf
[E] https://stats.stackexchange.com/a/40874
[F] https://en.wikipedia.org/wiki/Hotelling%27s_T-squared_distribution