Code boxes for Delta Method Tutorial

This tutorial requires an R version >= 4.1.0 and Python version >= 3.5

The following code will be used throughout the complete tutorial

```
#This should point to **your** Python path as explained in last section
Sys.setenv(RETICULATE_PYTHON = "/usr/local/Caskroom/miniconda/base/envs/DeltaMethod/bin/python")

library(caracas)
library(MASS)

#Parser for Sympy (you need sympy version 1.10 or 1.9 development)
#check sympy_version() to see you have the right one
sympy_version()
```

```
## [1] '1.9'
```

```
#Create parsers to find the functions check sympy's documentation
#and substitute dots for $ in https://docs.sympy.org/latest/index.html
sympy            <- get_sympy()
Parse            <- sympy$parsing$sympy_parser$parse_expr
RandomSymbol     <- sympy$stats$rv$RandomSymbol
Variance         <- sympy$stats$Variance
Covariance       <- sympy$stats$Covariance
Symbol           <- sympy$Symbol
Lambdify         <- sympy$utilities$lambdify
Simplify         <- sympy$simplify
Derivative       <- sympy$derive_by_array
Taylor           <- sympy$series
LaTeX            <- sympy$latex
Function         <- sympy$Function
```

Mean (classical)

```
#Declaration of constants (Symbol) and random variables (RandomSymbol)
mu     <- Symbol('mu')
sigma     <- Symbol('sigma', positive = TRUE) #Variance of x
n      <- Symbol('n', positive = TRUE, integer = TRUE)
xbar   <- RandomSymbol('xbar')

#List variables for parse
variable_list <- list('mu' = mu, 'xbar' = xbar)

#We are working with the variance of the ratio
phi           <- Parse("mu", local_dict = variable_list)

#Obtain the gradient of log RR
phi_prime     <- Derivative(phi,  list(mu))
```

```r
variable_list <- append(variable_list, list("phi_prime" = phi_prime))

#Direction vector (horizontal vector)
v <- Parse("xbar - mu", local_dict = variable_list)

#Get hadamard (directional) derivative
hadamard <- v #phi_prime = 1 so it doesn't change the result

#Variance
var_phi <- Variance(hadamard)$expand() |> Simplify()

#Recall the covariance is 0 due to independence
var_phi <- var_phi$subs(Variance(xbar), "sigma**2/n")

print(var_phi)
```

```
## sigma**2/n
```

Ratio of two means

To estimate the ratio of two means we need to define it as a function of symbols

```r
mu_x     <- Symbol('mu_x')
mu_y     <- Symbol('mu_y')
n        <- Symbol('n', positive=T) #Sample size of x
sigma_x     <- Symbol('sigma_x', positive = T) #Standard deviation of x
sigma_y     <- Symbol('sigma_y', positive = T) #Standard deviation of y
sigma_xy    <- Symbol('sigma_xy') #Covariance
xbar     <- RandomSymbol('xbar')
ybar     <- RandomSymbol('ybar')

#List variables for parse
variable_list <- list('mu_x' = mu_x, 'mu_y' = mu_y, 'xbar' = xbar,
                      'ybar' = ybar, 'n'  = n, 'sigma_xy' = sigma_xy,
                      'sigma_x' = sigma_x, 'sigma_y' = sigma_y)

#We are working with the variance of the ratio
mean_ratio    <- Parse("mu_x/mu_y", local_dict = variable_list)
```

We then obtain the derivative:

```r
#Obtain the gradient of log RR
g_mean        <- Derivative(mean_ratio,  list(mu_x, mu_y))

#Update variable list
variable_list <- append(variable_list, list("g_mean" = g_mean))
g_mean        <- Parse("Matrix(g_mean)", local_dict = variable_list)

#Direction vector (horizontal vector)
v <- Parse("Matrix([xbar - mu_x, ybar - mu_y])", local_dict = variable_list)

#Compute inner (dot) product
hadamard <- g_mean$dot(v)
print(hadamard)
```

```
## -mu_x*(-mu_y + ybar)/mu_y**2 + (-mu_x + xbar)/mu_y
```

The variance of that gradient is given as follows:

```
#Get the variance of gradient
var_mean_ratio <- Variance(hadamard)$expand() |> Simplify()
print(var_mean_ratio)
```

```
## (mu_x**2*Variance(ybar) - 2*mu_x*mu_y*Covariance(xbar, ybar) + mu_y**2*Variance(xbar))/mu_y**4
```

Recall that $\bar{X}$ and $\bar{Y}$ are random variables with the following variances:

$$\text{Var}\big[\bar{X}\big] = \frac{\sigma_X^2}{n}, \qquad \text{and} \qquad \text{Var}\big[\bar{Y}\big] = \frac{\sigma_Y^2}{n}.$$

which are specified as follows:

```
#To establish a power (say x^2) use ** function instead of ^
var_xbar      <- Parse("sigma_x**2/n", local_dict = variable_list)
var_ybar      <- Parse("sigma_y**2/n", local_dict = variable_list)
cov_xbar_ybar <- Parse("sigma_xy/n", local_dict = variable_list)
```

Further simplifications are allowed that result in a cleaner expression:

```
#Recall the covariance is 0 due to independence
var_mean_ratio <- var_mean_ratio$subs(Covariance(xbar, ybar), cov_xbar_ybar)

#Assign the variances of p1_hat and p2_hat
var_mean_ratio <- var_mean_ratio$subs(Variance(xbar), var_xbar)
var_mean_ratio <- var_mean_ratio$subs(Variance(ybar), var_ybar)

#This is the final expression for the variance
var_mean_ratio <- var_mean_ratio |> Simplify()
print(var_mean_ratio)
```

```
## (mu_x**2*sigma_y**2 - 2*mu_x*mu_y*sigma_xy + mu_y**2*sigma_x**2)/(mu_y**4*n)
```

We transform the expression into an R function:

```
#Transform the symbolic algebra into an R function
variable_list     <- append(variable_list,
                            list("var_mean_ratio" = var_mean_ratio))
var_mean <- Parse("lambdify((mu_x, mu_y, sigma_x, sigma_xy, sigma_y, n), var_mean_ratio)",
                           local_dict = variable_list)
```

The function can be evaluated for different values:

```
#You can use the variance function to estimate the IF with data
sample           <- mvrnorm(10, c(3,4), matrix(c(1,0.3,0.3,2), ncol = 2))
colnames(sample) <- c("X","Y")

#We obtain the sample means and variances
xbar   <- sample[,"X"] |> mean()
ybar   <- sample[,"Y"] |> mean()

sigma_x   <- sample[,"X"] |> sd()
sigma_y   <- sample[,"Y"] |> sd()
sigma_x_y <- cov(sample[,"X"], sample[,"Y"])
```

```r
var_mean(mu_x = xbar, mu_y = ybar, sigma_x = sigma_x,
         sigma_xy = sigma_x_y, sigma_y = sigma_y, n = 10)
```

```
## [1] 0.008869031
```

Relative Risk

To estimate the relative risk we need to define it as a function of symbols

```r
#In "Sympy" you need to declare variables for algebra as symbols
#and random variables as random symbols
p1     <- Symbol('p1', positive=T)
p2     <- Symbol('p2', positive=T)
N      <- Symbol('N', positive=T, integer=T) #Sample size
p1_hat <- RandomSymbol('p1_hat')
p2_hat <- RandomSymbol('p2_hat')

#List variables for parse
variable_list <- list('p1' = p1, 'p2' = p2, 'p1_hat' = p1_hat,
                      'p2_hat' = p2_hat, "N" = N)
```

Recall that in the case of estimating the Relative Risk: $RR = \frac{p_1}{p_2}$ we use the estimator:

$$\widehat{RR} = \frac{\hat{p}_1}{\hat{p}_2}.$$

As $\hat{p_1}$ and $\hat{p_2}$ are random variables thay have the following variance:

$$\text{Var}\big[\hat{p}_i\big] = \frac{p_i(1 - p_i)}{N} \qquad \text{for } i = 1, 2.$$

which is specified as follows:

```r
var_p1_hat <- Parse("p1*(1 - p1)/N", local_dict = variable_list)
var_p2_hat <- Parse("p2*(1 - p2)/N", local_dict = variable_list)
```

Finally, we create the log Relative Risk and calculate its derivative:

```r
#We are working with the variance of log_RR
log_RR        <- Parse("log(p1/p2)", local_dict = variable_list)

#Obtain the gradient of log RR
g_RR          <- Derivative(log_RR,  list(p1, p2))

#Uodate variable list with the derivative
variable_list <- append(variable_list, list("g_RR" = g_RR))

#Convert to matrix as it is a list
g_RR          <- Parse("Matrix(g_RR)", local_dict = variable_list)

#Direction vector (horizontal vector)
v <- Parse("Matrix([p1_hat - p1, p2_hat - p2])", local_dict = variable_list)

#Compute inner product
hadamard <- g_RR$dot(v)
print(hadamard)
```

```
## -(-p2 + p2_hat)/p2 + (-p1 + p1_hat)/p1
```

The variance of that gradient is given as follows:

```r
#Get the variance of gradient
var_log_rr <- Variance(hadamard)$expand() |> Simplify()
print(var_log_rr)
```

```
## Variance(p2_hat)/p2**2 - 2*Covariance(p1_hat, p2_hat)/(p1*p2) + Variance(p1_hat)/p1**2
```

Further simplifications are allowed that result in a cleaner expression:

```r
#Recall the covariance is 0 due to independence
var_log_rr <- var_log_rr$subs(Covariance(p1_hat, p2_hat), 0)

#Assign the variances of p1_hat and p2_hat
var_log_rr <- var_log_rr$subs(Variance(p1_hat), var_p1_hat)
var_log_rr <- var_log_rr$subs(Variance(p2_hat), var_p2_hat)

#This is the final expression for the variance
print(var_log_rr)
```

```
## (1 - p2)/(N*p2) + (1 - p1)/(N*p1)
```

Finally we transform the symbolic expression into an `R` function:

```r
#Transform the symbolic algebra into an R function
variable_list     <- append(variable_list, list("var_log_rr" = var_log_rr))
variance_function <- Parse("lambdify((p1,p2,N), var_log_rr)",
                           local_dict = variable_list)
```

The function can be evaluated for different values:

```r
#You can use the variance function to estimate the IF with data
variance_function(p1 = 0.7, p2 = 0.4, N = 100)
```

```
## [1] 0.01928571
```

```r
variance_function(p1 = 0.3, p2 = 0.5, N = 500)
```

```
## [1] 0.006666667
```

```r
variance_function(p1 = 0.1, p2 = 0.1, N = 2)
```

```
## [1] 9
```

Counter Example: Attributable Risk

For the counterexample of the function where the Taylor series is zero given by

$$AF = \begin{cases} \frac{e^{\theta/x}-1}{e^{\theta/x}} & \text{if } x \neq 0, \\ 1 & \text{if } x = 0. \end{cases}$$

```r
#Get the variables
x      <- Symbol('x', positive=T)

#Tenth order Taylor
Taylor("(exp(1/x) - 1)/exp(1/x)", x0 = 0, n = 10)$removeO()
```

```
## 0
```

Higher Order Delta Method

Estimate the Taylor series for $p(1-p)$:

```
#Get the variables
p      <- Symbol('p', positive=T)

#3rd order Taylor
Taylor("p*(1-p)", x0 = 1/2, n = 3)$removeO()
```

```
## 0.25 - (p - 0.5)**2
```

Correlation parameters

```
mu_uv    <- Symbol('mu_uv')
mu_u     <- Symbol('mu_u')
mu_v     <- Symbol('mu_v')
mu_u2    <- Symbol('mu_u2', positive=T)
mu_v2    <- Symbol('mu_v2', positive=T)
N        <- Symbol('N', positive=T, integer=T) #Sample size
uvbar    <- RandomSymbol("uvbar")
ubar     <- RandomSymbol("ubar")
vbar     <- RandomSymbol("vbar")
ubar2    <- RandomSymbol("ubar2")
vbar2    <- RandomSymbol("vbar2")

#List variables for parse
variable_list <- list('mu_uv' = mu_uv, 'mu_u' = mu_u, 'mu_v' = mu_v,
                      'mu_u2' = mu_u2, 'mu_v2'  = mu_v2, 'N' = N,
                      'uvbar' = uvbar, 'ubar' = ubar, 'vbar' = vbar,
                      'ubar2' = ubar2, 'vbar2' = vbar2)

#We are working with the variance of the covariance
exp <- "(mu_uv - mu_u*mu_v)/(sqrt(mu_u2 - mu_u**2)*sqrt(mu_v2 - mu_v**2))"
cov_function    <- Parse(exp, local_dict = variable_list)

#Gradient
#Obtain the gradient of log RR
g_cov <- Derivative(cov_function,  list(mu_uv, mu_u, mu_v, mu_u2, mu_v2))
g_cov <- Simplify(g_cov)

print(g_cov)
```

```
## [1/(sqrt(-mu_u**2 + mu_u2)*sqrt(-mu_v**2 + mu_v2)), (mu_u*mu_uv - mu_u2*mu_v)/((-mu_u**2 + mu_u2)**(
```

You can print the code in LaTeX as follows:

```
LaTeX(g_cov)
```

$$\left[\begin{matrix}\frac{1}{\sqrt{-\mu_{u}^{2}+\mu_{u2}}\sqrt{-\mu_{v}^{2}+\mu_{v2}}} & \frac{\mu_{u}\mu_{uv}-\mu_{u2}\mu_{v}}{(-\mu_{u}^{2}+\mu_{u2})^{\frac{3}{2}}\sqrt{-\mu_{v}^{2}+\mu_{v2}}} & \frac{-\mu_{u}\mu_{v2}+\mu_{uv}\mu_{v}}{\sqrt{-\mu_{u}^{2}+\mu_{u2}}(-\mu_{v}^{2}+\mu_{v2})^{\frac{3}{2}}} & \frac{\mu_{u}\mu_{v}-\mu_{uv}}{2(-\mu_{u}^{2}+\mu_{u2})^{\frac{3}{2}}\sqrt{-\mu_{v}^{2}+\mu_{v2}}} & \frac{\mu_{u}\mu_{v}-\mu_{uv}}{2\sqrt{-\mu_{u}^{2}+\mu_{u2}}(-\mu_{v}^{2}+\mu_{v2})^{\frac{3}{2}}}\end{matrix}\right]$$

```
#Convert to matrix as it is a list
variable_list  <- append(variable_list, list("g_cov" = g_cov))
```

```
g_cov          <- Parse("Matrix(g_cov)", local_dict = variable_list)

#Direction vector (horizontal vector)
v <- Parse("Matrix([uvbar - mu_uv, ubar - mu_u, vbar - mu_v,
          ubar2 - mu_u2, vbar2 - mu_v2])", local_dict = variable_list)

#Compute inner product
hadamard <- g_cov$dot(v)
```

$$\frac{\left(-\mu_{u}+\bar{u}\right)\left(\mu_{u}\mu_{uv}-\mu_{u2}\mu_{v}\right)}{\left(-\mu_{u}^{2}+\mu_{u2}\right)^{\frac{3}{2}}\sqrt{-\mu_{v}^{2}+\mu_{v2}}}+\frac{-\mu_{uv}+\bar{uv}}{\sqrt{-\mu_{u}^{2}+\mu_{u2}}\sqrt{-\mu_{v}^{2}+\mu_{v2}}}+\frac{\left(-\mu_{v}+\bar{v}\right)\left(-\mu_{u}\mu_{v2}+\mu_{uv}\mu_{v}\right)}{\sqrt{-\mu_{u}^{2}+\mu_{u2}}\left(-\mu_{v}^{2}+\mu_{v2}\right)^{\frac{3}{2}}}+\frac{\left(-\mu_{v2}+\bar{v}_{2}\right)\left(\mu_{u}\mu_{v}-\mu_{uv}\right)}{2\sqrt{-\mu_{u}^{2}+\mu_{u2}}\left(-\mu_{v}^{2}+\mu_{v2}\right)^{\frac{3}{2}}}+\frac{\left(-\mu_{u2}+\bar{u}_{2}\right)\left(\mu_{u}\mu_{v}-\mu_{uv}\right)}{2\left(-\mu_{u}^{2}+\mu_{u2}\right)^{\frac{3}{2}}\sqrt{-\mu_{v}^{2}+\mu_{v2}}}$$

```
#Get the variance of gradient
var_cov <- Variance(hadamard)$expand()

#Simplification process takes minutes be patient!
var_cov <- Simplify(var_cov)

#Further simplification required to delete the Cov(1,1)
var_cov <- Simplify(var_cov)

print(var_cov)
```

```
## (4*(-mu_u**4 + 2*mu_u**2*mu_u2 - mu_u2**2)*(mu_u**2*mu_v*mu_v2*Covariance(vbar, vbar2) - mu_u*mu_uv*m
```

We then create all the variances we are going to substitute:

```
sigma_uv    <- Symbol('sigma_uv',  positive = T)
sigma_u     <- Symbol('sigma_u',   positive = T)
sigma_v     <- Symbol('sigma_v',   positive = T)
sigma_u2    <- Symbol('sigma_u2',  positive = T)
sigma_v2    <- Symbol('sigma_v2',  positive = T)
cov_u_v     <- Symbol('cov_u_v',   positive = T)
cov_uv_v    <- Symbol('cov_uv_v',  positive = T)
cov_uv_u    <- Symbol('cov_uv_u',  positive = T)
cov_v_v2    <- Symbol('cov_v_v2',  positive = T)
cov_u_u2    <- Symbol('cov_u_u2',  positive = T)
cov_uv_u2   <- Symbol('cov_uv_u2',  positive = T)
cov_uv_v2   <- Symbol('cov_uv_v2',  positive = T)
cov_u2_v2   <- Symbol('cov_u2_v2', positive = T)
cov_u2_v    <- Symbol('cov_u2_v',  positive = T)
cov_v2_u    <- Symbol('cov_v2_u',  positive = T)

#We then substitute the variances
variable_list <- append(variable_list,
                        list("sigma_u"   = sigma_u,  "sigma_uv" = sigma_uv,
                             "sigma_v"   = sigma_v,  "sigma_u2" = sigma_u2,
                             "sigma_v2"  = sigma_v2, "cov_u_v" = cov_u_v,
                             "cov_v_v2"  = cov_v_v2, "cov_uv_v" = cov_uv_v,
                             "cov_uv_u"  = cov_uv_u, "cov_uv_u2" = cov_uv_u2,
                             "cov_uv_v2" = cov_uv_v2,
                             "cov_u_u2"  = cov_u_u2, "cov_u2_v2" = cov_u2_v2,
                             "cov_u2_v"  = cov_u2_v, "cov_v2_u" = cov_v2_u))
```

```r
var_ubar     <- Parse("sigma_u**2/N",  local_dict = variable_list)
var_vbar     <- Parse("sigma_v**2/N",  local_dict = variable_list)
var_uvbar    <- Parse("sigma_uv**2/N", local_dict = variable_list)
var_u2bar    <- Parse("sigma_u2**2/N", local_dict = variable_list)
var_v2bar    <- Parse("sigma_u2**2/N", local_dict = variable_list)
covar_u_v    <- Parse("cov_u_v/N", local_dict = variable_list)
covar_u_u2   <- Parse("cov_u_u2/N", local_dict = variable_list)
covar_v_v2   <- Parse("cov_v_v2/N", local_dict = variable_list)
covar_u2_v   <- Parse("cov_u2_v/N", local_dict = variable_list)
covar_v2_u   <- Parse("cov_v2_u/N", local_dict = variable_list)
covar_u2_v2  <- Parse("cov_u2_v2/N", local_dict = variable_list)
covar_uv_u   <- Parse("cov_uv_u/N", local_dict = variable_list)
covar_uv_v   <- Parse("cov_uv_v/N", local_dict = variable_list)
covar_uv_u2  <- Parse("cov_uv_u2/N", local_dict = variable_list)
covar_uv_v2  <- Parse("cov_uv_v2/N", local_dict = variable_list)

var_cov <- var_cov$subs(Variance(ubar),  var_ubar)
var_cov <- var_cov$subs(Variance(vbar),  var_vbar)
var_cov <- var_cov$subs(Variance(uvbar), var_uvbar)
var_cov <- var_cov$subs(Variance(ubar2), var_u2bar)
var_cov <- var_cov$subs(Variance(vbar2), var_v2bar)
var_cov <- var_cov$subs(Covariance(ubar, vbar),   covar_u_v)
var_cov <- var_cov$subs(Covariance(ubar, ubar2),  covar_u_u2)
var_cov <- var_cov$subs(Covariance(vbar, vbar2),  covar_v_v2)
var_cov <- var_cov$subs(Covariance(ubar2, vbar),  covar_u2_v)
var_cov <- var_cov$subs(Covariance(vbar2, ubar),  covar_v2_u)
var_cov <- var_cov$subs(Covariance(ubar2, vbar2), covar_u2_v2)
var_cov <- var_cov$subs(Covariance(uvbar, ubar),  covar_uv_u)
var_cov <- var_cov$subs(Covariance(uvbar, vbar),  covar_uv_v)
var_cov <- var_cov$subs(Covariance(uvbar, vbar2), covar_uv_v2)
var_cov <- var_cov$subs(Covariance(uvbar, ubar2), covar_uv_u2)

#Further simplification
var_cov <- Simplify(var_cov)
```

```r
#Transform the symbolic algebra into an R function
variable_list     <- append(variable_list, list("var_cov" = var_cov))

exp <- paste0("lambdify((mu_u, mu_v, mu_uv, mu_u2, mu_v2, sigma_u, sigma_v, sigma_uv,",
              "sigma_u2, sigma_v2, cov_u_v, cov_uv_u, cov_uv_v, cov_v_v2,",
              "cov_u_u2, cov_uv_u2,",
              "cov_uv_v2, cov_u2_v2, cov_u2_v, cov_v2_u, N",""), var_cov)")
var_cov <- Parse(exp, local_dict = variable_list)

#You can check the variance function works as intended:
samples <- 1000
rho     <- 0.83
data    <- mvrnorm(n=samples, mu=c(0, 0), Sigma=matrix(c(1, rho, rho, 1), nrow=2))
colnames(data) <- c("U","V")
rho_hat <- cov(data[,"U"], data[,"V"])

variance_rho <- var_cov(
  mu_u     = mean(data[,"U"]),
  mu_v     = mean(data[,"V"]),
```

```
  mu_uv    = mean(data[,"U"]*data[,"V"]),
  mu_u2    = mean(data[,"U"]^2),
  mu_v2    = mean(data[,"V"]^2),
  sigma_u  = sd(data[,"U"]),
  sigma_v  = sd(data[,"V"]),
  sigma_uv = sd(data[,"U"]*data[,"V"]),
  sigma_u2 = sd(data[,"U"]^2),
  sigma_v2 = sd(data[,"V"]^2),
  cov_u_v  = cov(data[,"U"],data[,"V"]),
  cov_uv_u = cov(data[,"U"]*data[,"V"],data[,"U"]),
  cov_uv_v = cov(data[,"U"]*data[,"V"],data[,"V"]),
  cov_v_v2 = cov(data[,"V"],data[,"V"]^2),
  cov_u_u2 = cov(data[,"U"],data[,"U"]^2),
  cov_uv_u2 = cov(data[,"U"]*data[,"V"],data[,"U"]^2),
  cov_uv_v2 = cov(data[,"U"]*data[,"V"],data[,"V"]^2),
  cov_u2_v = cov(data[,"U"]^2,data[,"V"]),
  cov_v2_u = cov(data[,"V"]^2,data[,"U"]),
  cov_u2_v2 = cov(data[,"U"]^2,data[,"V"]^2),
  N = samples
)

#Confidence interval
rho_hat + qnorm(1 - 0.975/2)*sqrt(variance_rho)
```

```
## [1] 0.8367203
```

```
rho_hat - qnorm(1 - 0.975/2)*sqrt(variance_rho)
```

```
## [1] 0.8359433
```

Summary (steps)

We can summarize the previous results in the following steps. Note that the code will not work as no function `f` is specified:

1. Instantiate the estimator $\hat{\theta}$ of θ as random variable and θ as constant.

```
theta         <- Symbol('theta')
theta_hat     <- RandomSymbol('theta_hat')

variable_list <- list("theta" = theta, "theta_hat" = theta_hat)
```

2. Create the function ϕ whose variance is to be estimated. In this example it is θ^2.

```
#This is usually done with Parse if we know the function
phi           <- Parse("theta**2", local_dict = variable_list)
variable_list <- append(variable_list, list("phi" = phi))
```

3. Obtain ϕ's derivative

```
phi_prime     <- Derivative(phi, list(theta))
variable_list <- append(variable_list, list("phi_prime" = phi_prime))
phi_prime     <- Parse("Matrix(phi_prime)", local_dict = variable_list)
```

4. Obtain the direction vector v

```r
v <- Parse("Matrix([theta_hat - theta])", local_dict = variable_list)
```

5. Get the dot product:

```r
hadamard <- phi_prime$dot(v)
```

6. Calculate the variance

```r
#Get the variance of gradient
variance_f <- Variance(hadamard)$expand() |> Simplify()
```

7. Replace the variance of the estimator by the new variable

```r
sigma_thetahat <- Parse("sigma_thetahat**2",  local_dict = variable_list)
variance_f     <- variance_f$subs(Variance(theta_hat), sigma_thetahat)
```

7. Conver to `R` function

```r
variable_list     <- append(variable_list, list("variance_f" = variance_f))
variance_function <- Parse("lambdify((theta,sigma_thetahat), variance_f)",
                           local_dict = variable_list)
```

8. Enjoy!

```r
variance_function(theta = 3, sigma_thetahat = 0.01)
```

```
## [1] 0.0036
```

Installation

Installing `R`

Go to https://cran.r-project.org/ and choose your operating system.

- If using `Windows` select this option, then choose `base` and download the `exe` file.
- If using `Mac` select this option, then choose the option based on your processor (Intel or ARM). To see what processor you have go to `About this Mac` and check the **processor**.
- If using `Linux` the suggested way is to download the binaries from `CRAN` as the ones you get from the package managers (`apt`, `yum`, etc) are usually old.

Suggestion `RStudio`

`RStudio` is an integrated development environment for `R` (*i.e.* its an editor that allows you to work better on your scripts). It needs to be installed separately from `R` (you need both installations). To install go to https://www.rstudio.com/products/rstudio/download/ and choose the `Free` Desktop version.

Installing `R` packages

Open `R` (or `RStudio`) if installed and write `install.packages(c("reticulate","caracas"))` on the console and wait for installation.

Installing Python

Python already comes installed with your operating system; however we suggest an additional (separate) installation using `Anaconda`. To install go to https://www.anaconda.com/products/individual. Anaconda comes bundled with a bunch of data science tools that are not required for this tutorial: if you'd prefer a meager installation you can install Miniconda https://docs.conda.io/en/latest/miniconda.html. For both installations the process that follows is the same. We'll refer to both as `conda`.

Please make sure you are installing a `conda` version corresponding to Python >= 3.5 (*i.e.* any Python version greater than 3.5 like 3.6, 3.7 are great for this.)

Creating a conda environment

Once `conda` is installed open Anaconda Prompt (Windows) or Terminal (Mac). Write:

```
conda create -y --name DeltaMethod python=3.9
```

to create a new environment called `DeltaMethod`. Write

```
conda env list
```

to get the path to your environment. For example my path is `/usr/local/Caskroom/miniconda/base/envs/DeltaMethod`.

We'll use this environment in the `R` code. Before loading library `caracas`, at the beginning of the code, write:

```
Sys.setenv(RETICULATE_PYTHON = "path/to/your/environment/bin/python")
```

for example, in my computer it ends up like this:

```
Sys.setenv(RETICULATE_PYTHON = "/usr/local/Caskroom/miniconda/base/envs/DeltaMethod/bin/python")
```

to verify that the environment is set check its in `python` and `libpython`:

```
reticulate::py_config()
```

```
## python:         /usr/local/Caskroom/miniconda/base/envs/DeltaMethod/bin/python
## libpython:      /usr/local/Caskroom/miniconda/base/envs/DeltaMethod/lib/libpython3.9.dylib
## pythonhome:     /usr/local/Caskroom/miniconda/base/envs/DeltaMethod:/usr/local/Caskroom/miniconda/bas
## version:        3.9.7 | packaged by conda-forge | (default, Sep 29 2021, 19:23:19)  [Clang 11.1.0 ]
## numpy:          /usr/local/Caskroom/miniconda/base/envs/DeltaMethod/lib/python3.9/site-packages/nump
## numpy_version:  1.21.4
##
## NOTE: Python version was forced by RETICULATE_PYTHON
```

Installing Sympy

To install Sympy ensure you are using a version `>= 1.9`. Go to `Anaconda Prompt` (`Terminal`) and activate the environment:

```
conda activate DeltaMethod
```

Finally, install `sympy` and `numpy` in the environment

```
conda install -y -c conda-forge sympy">=1.9" numpy
```

Warning If `R` or `RStudio` was already open before completing installation you'll need to restart them.