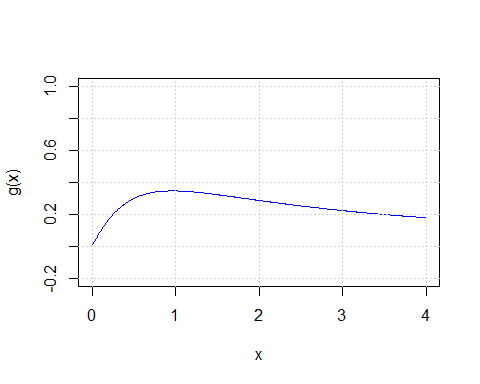
Christian Kammerer, Jakob Lindner

# Report for Computer Lab 1 in Computational Statistics

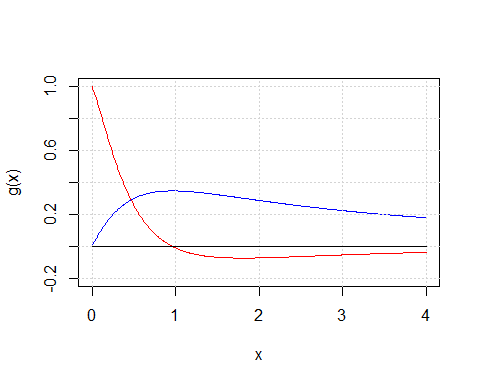
## Question 1: Maximization of a function in one variable

### a. Plot the function g(x) in the interval [0,4]. What is your guess for the maximum point?



The maximum point seems to be between 0.9 and 1

### b. Compute the derivative g’(x) of g(x). Plot g’(x) in [0, 4], and add a horizontal reference line at 0 to the plot.



### c. Write your own R function applying the bisection method to g′ to find a local maximum of g for a user-selected starting interval.

bisection <- function(fun, a, b, threshold){  
 #' computes the maximum of a function using the bisection method  
 #'   
 #' params:  
 #' fun: derivative of the function  
 #' a: starting point of initial interval  
 #' b: last point of initial interval   
 #' threshold: convergence criterion  
   
 # check if criterion for starting interval is met  
 stopifnot(fun(a)\*fun(b) < 0)  
   
 xt <- (a + b)/2  
   
 it <- 1 # start iterations counter with 1 as line above is first iteration  
   
 # improve approximation until convergence criterion is met  
 while(TRUE){  
 it <- it + 1  
   
 # decide which end of the interval must be updated  
 if(fun(a) \* fun(xt) <= 0){  
 b <- xt   
 }  
 else{  
 a <- xt  
 }  
 xt\_next <- (a + b)/2  
   
 # check for convergence  
 if(abs((xt\_next - xt)) < threshold){  
 xt <- xt\_next  
 break  
 }  
   
 else{  
 xt <- xt\_next  
 }  
 }  
   
 return(c(xt, it))  
}

### d. Write your own R function applying the secant method to g′ to find a local maximum of g for a

user-selected pair of starting values.

secant <- function(fun, x0, x1, threshold){  
 #' computes the maximum of a function using the bisection method  
 #'   
 #' params:  
 #' fun: derivative of the function  
 #' x0: initial x0 value for secant  
 #' x1: initial x1 value for secant   
 #' threshold: convergence criterion  
   
 it <- 0 # iterations counter  
   
 while(TRUE){  
 it <- it + 1  
 # apply formula to get next value for secant  
 x\_next <- x1 - fun(x1) \* (x1 - x0)/ (fun(x1) - fun(x0))  
   
 if(abs((x1 - x0)) < threshold){  
 break  
 }  
 else{  
 x0 <- x1  
 x1 <- x\_next  
 }  
 }  
   
 return(c(x1, it))  
}

### e. Run the functions in c. and d. for different starting intervals/pairs of starting values and check when they converge to the true maximum and when not. Discuss why. Compare the two methods also in terms of number of iterations used and programming effort required.

bisection(dg, 0.5,1.2,0.0001) # approximates good

## [1] 0.9609985 13.0000000

# bisection(dg, 0.5,0.8,0.0001) # does not work as both dg(x)are negative  
bisection(dg, 0.1,3.5,0.0001) # works also for bigger starting interval

## [1] 0.9610504 16.0000000

secant(dg, 0.5, 1.2, 0.0001) # finds a good approximation

## [1] 0.9610603 7.0000000

secant(dg, 2.5, 3.2, 0.0001) # does not find the correct maximum as it approaches 0 for x -> inf

## [1] 6.156488e+102 1.002000e+03

secant(dg, 1.1, 1.3, 0.0001) # approximates the value well even though both dg(x) are negative (difference to bisection method)

## [1] 0.9610603 7.0000000

secant(dg, 0.1, 0.3, 0.0001) # finds the correct approximation

## [1] 0.96106 8.00000

**Comparison in terms of iterations and programming effort:** The bisection method takes more (roughly twice as many) iterations compared to secant (13-16 compared to 7-8) while the programming effort was similar for both methods.

### f. When you just should program one of them: Would you use bisection or secant, here? In general, for another function g(x) to be maximized: When would you switch and use the other algorithm?

As both methods have problems to find the maximum for some starting values but the secant method converges faster, we would choose secant. In general we could start with the bisection method. If we start with a pair that meets the starting criterion it is guaranteed to find a maximum. When the interval is narrowed down we can switch to the secant method for faster convergence. If it fails to converge in this interval we could switch back to the bisection method.

## Question 2: Computer arithmetics (variance)

### a. Write your own R function, myvar, to estimate the variance in this way.

myvar <- function(x){  
 n <- length(x)  
 v <- 1/(n-1) \* (sum(x^2) - 1/n \* (sum(x)^2))  
 return (v)  
}

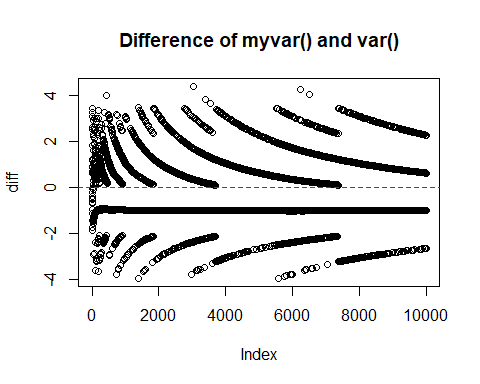
### b. Generate a vector x = (x1, . . . , x10000) with 10000 random numbers with mean 108 and variance 1.

set.seed(42)  
x <- rnorm(10000, mean=10^8, sd=sqrt(1))

### c. For each subset Xi = {x1, . . . , xi}, i = 1, . . . , 10000 compute the difference Yi = myvar(Xi) −

var(Xi), where var(Xi) is the standard variance estimation function in R. Plot the dependence Yi on i. Draw conclusions from this plot. How well does your function work? Can you explain the behaviour?

diff <- numeric(10000)  
for (i in 1:10000){  
 x\_sub <- x[1:i]  
 diff[i] <- myvar(x\_sub) - var(x\_sub)  
}

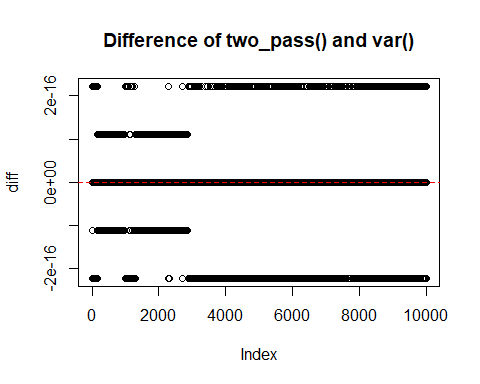


The difference between the result of myvar and the real variance is very big compared to the variance of 1 that the data was created with. Hence myvar does not work well with this dataset.  
The reason can be so-called catastrophic cancellation. It occurs in the substraction of two nearly equal numbers, which we have in our case because of the large mean and little variance, and results in a loss of precision. Due to computer arithmetics the significant bits are mostly cancelled out and the result depends heavily on the least significant and least accurate bits.

### d. How can you better implement a variance estimator? Find and implement a formula that will give the same results as var().

As alternative approach we implement the two-pass algorithm. (source: <https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance>) It may still face some issues on big datasets but the plot of the results will show how well it performs on our data.

two\_pass <- function(x) {  
 n <- length(x)  
 mean\_x <- mean(x)  
 v <- sum((x - mean\_x)^2) / (n - 1)  
 return(v)  
}  
  
two\_pass\_diff <- numeric(10000)  
  
for (i in 1:10000) {  
 Xi <- x[1:i]  
 two\_pass\_diff[i] <- two\_pass(Xi) - var(Xi)  
}  
  
# Plot the comparison for two\_pass  
plot(1:10000, two\_pass\_diff, xlab= "Index", ylab="diff", main = "Difference of two\_pass() and var()")  
abline(h = 0, col = "red", lty = 2)



It gives the same result as the built-in var()-function for many cases but also deviates for some. As the difference is very small though, the two-pass algorithm performs nearly as good as var().