Case Study 1; Task 1: Fibonacci numbers

Jonas Tatzberger

2024-04-03

a) Defining functions with for & while loops

At first the function calculating the ratio using a for loop:

```
r_for <- function(i) {
  f_i <- 0
  f_ip1 <- 1
  result <- c()
  for (x in 1:i) {
    f_ip1_new <- f_i+f_ip1
    f_i <- f_ip1
    f_ip1 <- f_ip1_new
    result <- append(result, f_ip1/f_i)
  }
  return (result)
}
```

Running the function using a test output:

```
test_for <- r_for(10)
print(test_for)
```

```
##  [1] 1.000000 2.000000 1.500000 1.666667 1.600000 1.625000 1.615385 1.619048
##  [9] 1.617647 1.618182
```

Next the function using a while-loop:

```
r_while <- function(i) {
  f_i <- 0
  f_ip1 <- 1
  result <- c()
  x=0
  while(x < i) {
    f_ip1_new <- f_i+f_ip1
    f_i <- f_ip1
    f_ip1 <- f_ip1_new
    result <- append(result, f_ip1/f_i)
    x = x+1
  }
  return (result)
}
```

Running the function using a test output:

```
test_while <- r_while(10)
print(test_while)
```

```
##  [1] 1.000000 2.000000 1.500000 1.666667 1.600000 1.625000 1.615385 1.619048
##  [9] 1.617647 1.618182
```

As we can see, both functions produce the same outputs, which seem to nicely converge towards the golden ratio.

b) Benchmarking the function to see which is more efficient

To test the functions, the microbenchmark package is used. When applying the function, the standard values are used (the number of iterations for the expression is set to 100 here). This results in a statistic comprised of n=100 tests.

```
library(microbenchmark)
```

```
## Warning: Paket 'microbenchmark' wurde unter R Version 4.3.3 erstellt
```

Testing for n=200:

```
microbenchmark(r_for(200))
```

```
## Unit: microseconds
##        expr   min     lq    mean median     uq    max neval
##  r_for(200) 351.3 470.35 535.357 517.25 567.85 1058.4   100
```

```
microbenchmark(r_while(200))
```

```
## Unit: microseconds
##          expr   min     lq    mean median    uq    max neval
##  r_while(200) 335.6 365.25 433.498 394.65 458.4 1122.2   100
```

Here we can see that the mean time that the "r_for" function took (397.028us) is lower than the "r_while" function (423.145us). For n=200 the for loop is faster.

Testing for n=2000:

```
microbenchmark(r_for(2000))
```

```
## Unit: milliseconds
##         expr     min       lq     mean   median       uq     max neval
##  r_for(2000) 12.0583 13.23715 17.44319 14.72065 24.86885 29.6573   100
```

```
microbenchmark(r_while(2000))
```

```
## Unit: milliseconds
##           expr    min      lq     mean  median       uq     max neval
##  r_while(2000) 6.5969 12.9654 15.82451 14.7827 17.05375 28.5939   100
```

Again we can see that the mean time that the "r_for" function took (18.295ms) is lower than the "r_while" function (20.847ms). For n=2000 also the for-loop is faster.

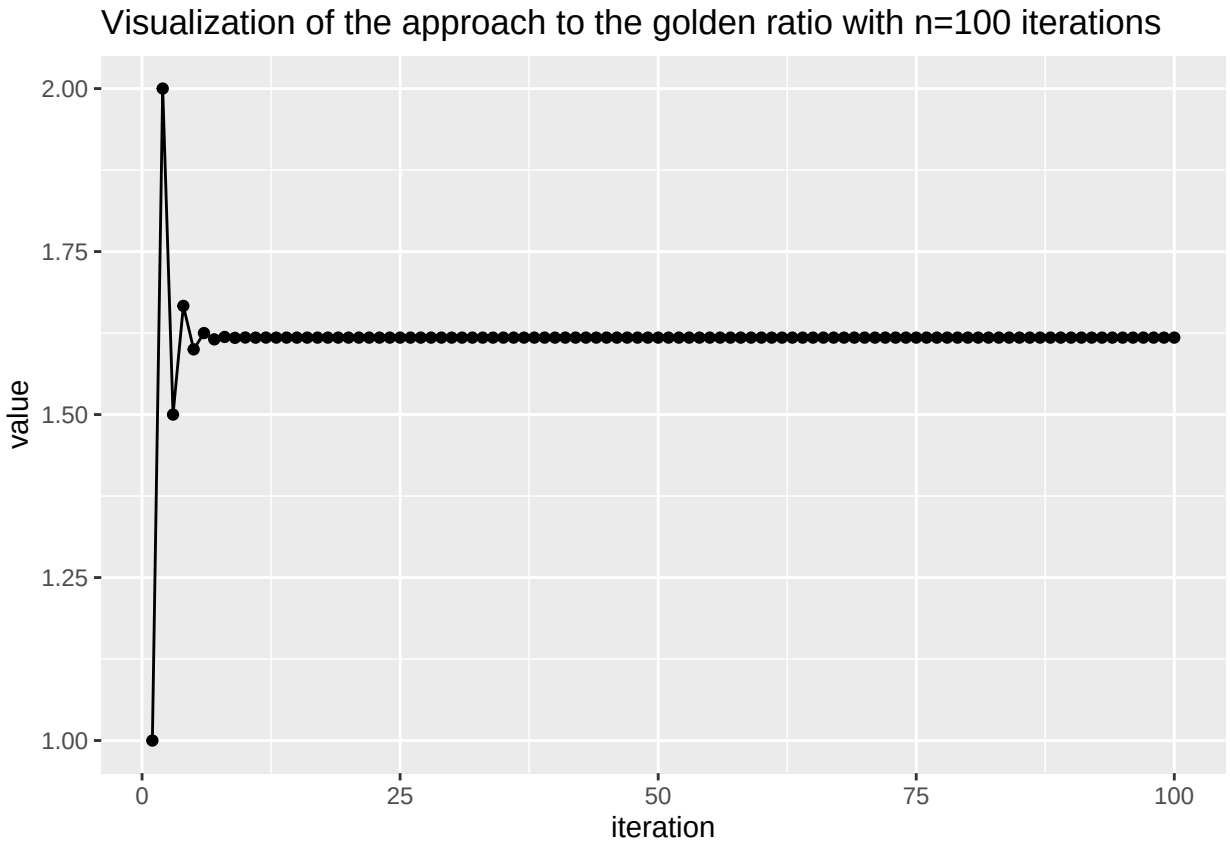
c) Plotting for n=100:

In order to visualize the function, the package "ggplot2" is used. To work with the data it is transformed into a data-frame and an iteration column is added.

```
library(ggplot2)
```

```
## Warning: Paket 'ggplot2' wurde unter R Version 4.3.2 erstellt
```

```
value = r_for(100)
r_100 <- data.frame(value)
r_100$iteration <- 1:nrow(r_100)
ggplot(r_100, aes(x=iteration, y=value)) + geom_point() + geom_line() + ggtitle("Visualization of the ap
```



From the plot, one can deduce that the function converges at around the 6th iteration, from this point on no significant changes are visible.