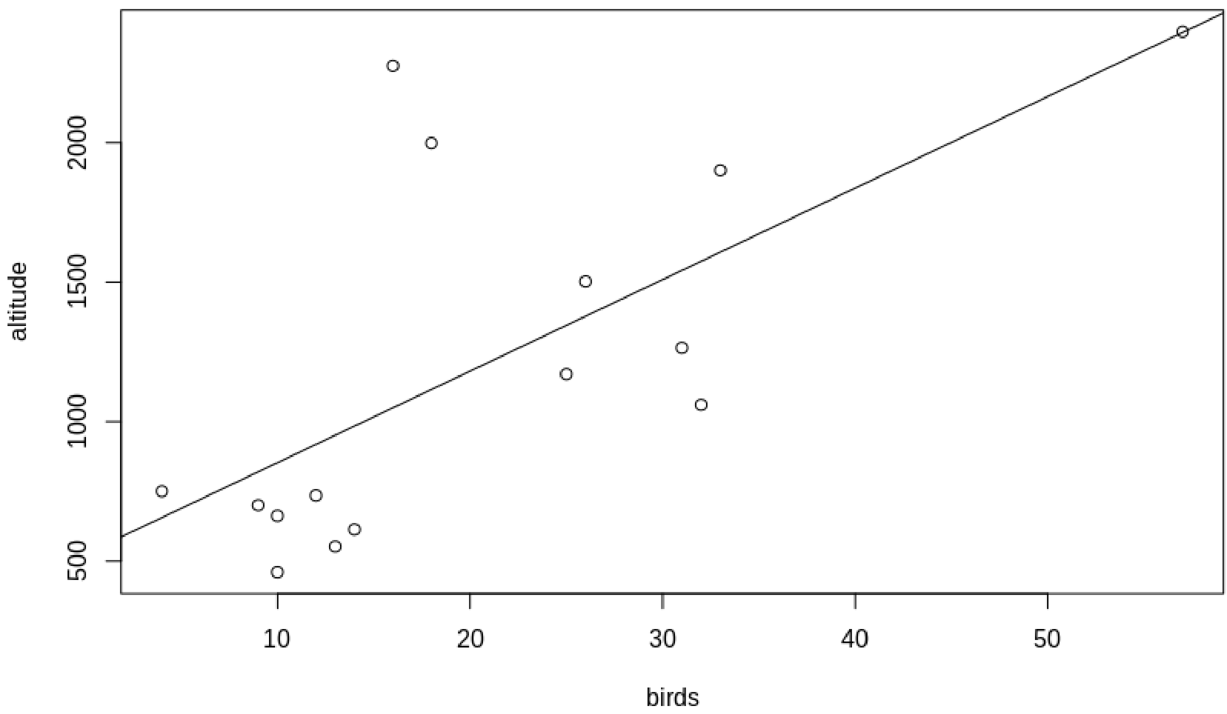
UE Introduction to Computational Statistics - Assignment 2 - 15.10.21

Task 1

birds<-c(57,31,25,32,33,10,18,9,12,14,10,26,4,16,13);altitude<-
c(2397,1264,1170,1060,1900,460,1998,700,735,613,662,1502,750,2275,552)
plot(birds,altitude); linearMod <- lm(altitude~birds)
print(linearMod)
abline(linearMod)



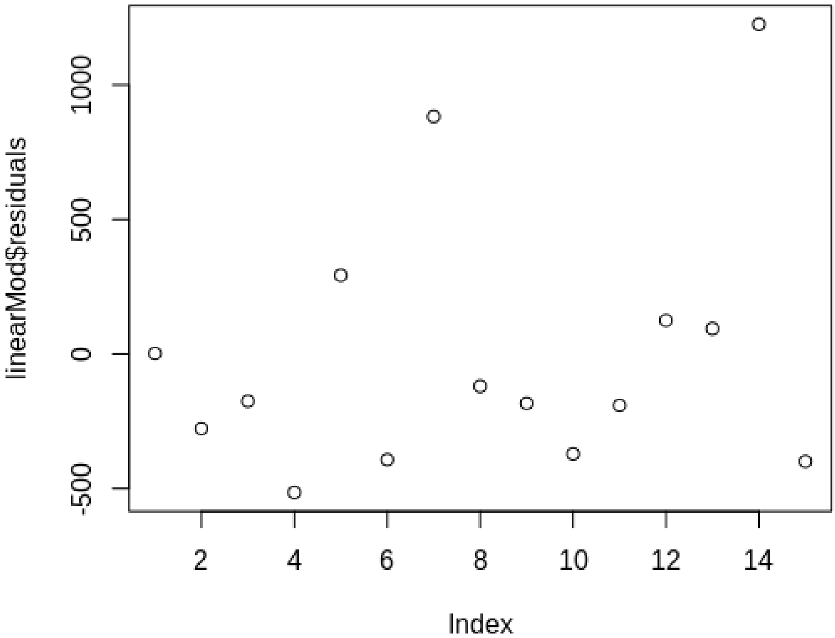
summary(linearMod)

```
Call:
lm(formula = altitude ~ birds)

Residuals:
   Min     1Q Median     3Q    Max 
-514.4 -324.2 -174.7  109.4 1225.6 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)   
(Intercept)  524.489    241.974   2.168   0.0493 * 
birds         32.809      9.873   3.323   0.0055 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 

Residual standard error: 503.8 on 13 degrees of freedom
Multiple R-squared:  0.4593,	Adjusted R-squared:  0.4177 
F-statistic: 11.04 on 1 and 13 DF,  p-value: 0.005498

> 
```

k11923286

Residual Standard Error 503.8 on 13 degrees of freedom
Estimate (Intercept): 524.489 → a
Estimate birds: 32.809 → b
→ bx + a + error = 32.809 x + 524.489 + error = y

Residuals:
plot(linearMod$residuals)



(task 2 next page)

Task 2

a) Approximate exp(x) with help of the Taylor series in R:

```
taylor_function <-function(x)
    {i <- 0
    not_matters <-3
    sum_old <- 0
    sum_now<-0
    while(not_matters<4)
        {next_summand<-x^i/factorial(i)
        sum_now<- sum_old + next_summand
        i<-i+1
        if(abs(next_summand)<(.Machine$double.eps^(1/2)*abs(sum_old)))
            {not_matters<-5}
        sum_old<-sum_now}
    return(sum_now)}
```

b) Comparison of taylor_function from a) and exp(x) function in R:

```
comparison <- function(x)
    {print(paste(taylor_function(x),"vs.",exp(x)))
    print(paste("Same?", taylor_function(x)==exp(x)))}
```

```
> comparison(2)
[1] 7.389056
[1] "vs."
[1] 7.389056
> comparison(1)
[1] 2.718282
[1] "vs."
[1] 2.718282
> comparison(2)
[1] 7.389056
[1] "vs."
[1] 7.389056
> comparison(8)
[1] 2980.958
[1] "vs."
[1] 2980.958
> comparison(15)
[1] 3269017
[1] "vs."
[1] 3269017
> comparison(20)
[1] 485165193
[1] "vs."
[1] 485165195
```

→ Difference, pocket calculator has same result as exp(20) but with one decimal place!

```
> comparison(22)
[1] 3584912824
[1] "vs."
[1] 3584912846
> comparison(24)
[1] 26489121934
[1] "vs."
[1] 26489122130
> comparison(28)
[1] 1.446257e+12
[1] "vs."
[1] 1.446257e+12
> comparison(40)
[1] 2.353853e+17
[1] "vs."
[1] 2.353853e+17
```

→ Both not very precise, pocket calculator has more precision

```
> comparison(60)
[1] 1.142007e+26
[1] "vs."
[1] 1.142007e+26
```

c) Give range of x where reasonable approximation. Make modification to extend this range.

To solve
solve(exp(x),taylor_function(x))

```
> comparison(0.00000002)
[1] "1.00000002 vs. 1.00000002"
[1] "Same? FALSE"
> comparison(0.00000001)
[1] "1.00000001 vs. 1.00000001"
[1] "Same? TRUE"
>
```

→ Range (-0.0000001;00000001]

Exp(20) seems to be more precisely than taylor_function(20). Also exp(20)>taylor_function(20), same relation for other shown values of 20 <= x < 24.

Stopping condition in taylor_function(x): $\frac{x^i}{i!} < .Machine\$double.eps^{1/2} \cdot \sum_{j=0}^{j=i-1} \frac{x^j}{j!}$

→ Modifying `.Machine$double.eps^(1/2)`… …bigger → Stop earlier
…smaller → stop later

Stop later because exp(20)>taylor_function(20). Therefore, the final sum can get larger.

```
taylor_function_modified <-function(x, smaller_fac)
    {i <- 0
    not_matters <-3
    sum_old <- 0
    sum_now<-0
    while(not_matters<4)
        {next_summand<-x^i/factorial(i)
        sum_now<- sum_old + next_summand
        i<-i+1
        if(abs(next_summand)<
        (.Machine$double.eps^(smaller_fac/2)*abs(sum_old)))
            {not_matters<-5} # break while loop
        sum_old<-sum_now}
    return(sum_now)}
```

```
comparison_modified <- function(x, smaller_fac)
    {print(paste(taylor_function_modified(x, smaller_fac),"vs.",exp(x)))
    print(paste("Same?", taylor_function_modified(x, smaller_fac)==exp(x)))}
```

```
> taylor_function_modified(20,2)
[1] 485165195
> exp(20)
[1] 485165195
>
```

```
> taylor_function_modified(20,1.01)
[1] 485165193
> taylor_function_modified(20,1.1)
[1] 485165195
>
```

```
> taylor_function_modified(20,1.04)
[1] 485165195
> taylor_function_modified(20,1.03)
[1] 485165193
```

(Task 3 next page)

Task 3

Taylor Series formula: $f(x)=\sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$

a) Try out values in R:

```
task3a <- function(x)
    {return((sqrt(1+x)-1)/x)}
```

```
> task3a(1)
[1] 0.4142136
> task3a(10^(-4))
[1] 0.4999875
> task3a(10^(-32))
[1] 0
```

b) Use Taylor series for sqrt(1+x)

Determine the taylor series in R for sqrt(1+x):

```
taylor_series3b <- function(x,x_0,f) # please write f down as: expression(formula(xe))
# not allowed for 3c: x_0 = 0; x = 0
{

  x_0<-x_0;i <- 1; not_matters <-3; sum_now<-0
  to_differentiate <- f

  # First of all taylor sum starts with the 0-th derivative:
  xe <- x_0
  next_summand <- eval(f)/factorial(0) * (x-x_0)^0
  rm(xe)
  sum_old <- next_summand

  while(not_matters<4)
  {
    current_derivative <- D(to_differentiate, "xe")

    to_differentiate2 <- current_derivative

    back_x <- x
    xe <- x_0 # to evaluate the derivative by x_0

    next_summand <- eval(current_derivative)/factorial(i) * (x-x_0)^i

    if(is.nan(next_summand))
        {print("calculation got too long");break}
```

```
    rm(xe)
    print(paste('next s:', next_summand))
    x <- back_x
    sum_now <- sum_old + next_summand
    print(paste("sum now",sum_now))
    print('############')
    if(abs(next_summand)<(.Machine$double.eps^(1/2)*abs(sum_old)))
        {print('done');break}

    to_differentiate <- to_differentiate2
    sum_old <- sum_now
    i <-i +1
  }

  print("final sum")
  return(sum_now)
}
```

Slightly modify the function from a):

```
formula <-  expression(sqrt(xe+1))
task3b <- function(x,x_0,formula)
    {result = taylor_series3b(x, x_0,formula)
    return((result-1)/x)}
```

Approximate for small x:
x={10^(-4), 10^(-32)}

```
> task3b(10^(-4),0)
[1] "done"
[1] 0.4999875
> task3b(10^(-4),1)
[1] "done"
[1] 0.5000685
> task3b(10^(-32),0)
[1] "done"
[1] 0
> task3b(10^(-32),1)
[1] "done"
[1] 8.113104e+23
```

c) Taylor series for whole function:

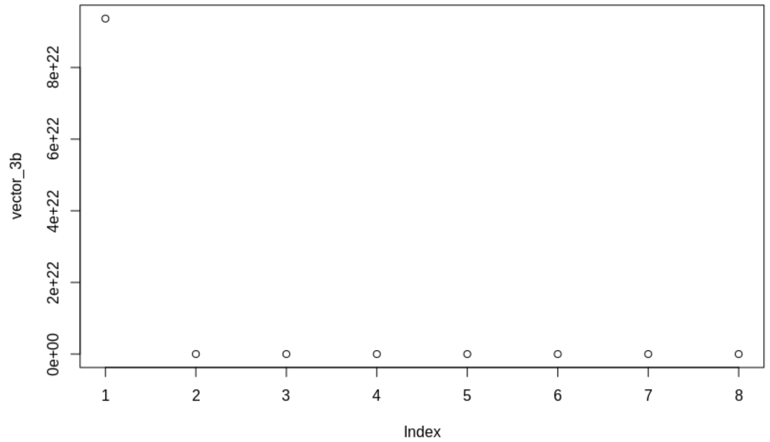
```
formula <-  expression((sqrt(xe+1)-1)/xe)
task3c <- function(x,x_0,formula)
    {return(taylor_series3b(x, x_0, formula))}
```

```
> task3c(10^(-4),0.5,formula)
[1] "next s: 0.041233204028851"
[1] "sum now 0.490722946812029"
[1] "############"
[1] "next s: 0.00721787343157781"
[1] "sum now 0.497940820243607"
[1] "############"
[1] "next s: 0.00154971610027882"
[1] "sum now 0.499490536343886"
[1] "############"
[1] "next s: 0.000369076904170904"
[1] "sum now 0.499859613248056"
[1] "############"
[1] "next s: 9.36480118146529e-05"
[1] "sum now 0.499953261259871"
[1] "############"
[1] "next s: 2.48042807222051e-05"
[1] "sum now 0.499978065540593"
[1] "############"
[1] "next s: 6.77732937868524e-06"
[1] "sum now 0.499984842869972"
[1] "############"
[1] "next s: 1.89599487227379e-06"
[1] "sum now 0.499986738864844"
[1] "############"
[1] "next s: 5.40345312525343e-07"
[1] "sum now 0.499987279210157"
[1] "############"
[1] "next s: 1.56368480760263e-07"
[1] "sum now 0.499987435578638"
[1] "############"
[1] "calculation got too long"
[1] "final sum"
[1] 0.4999874
```

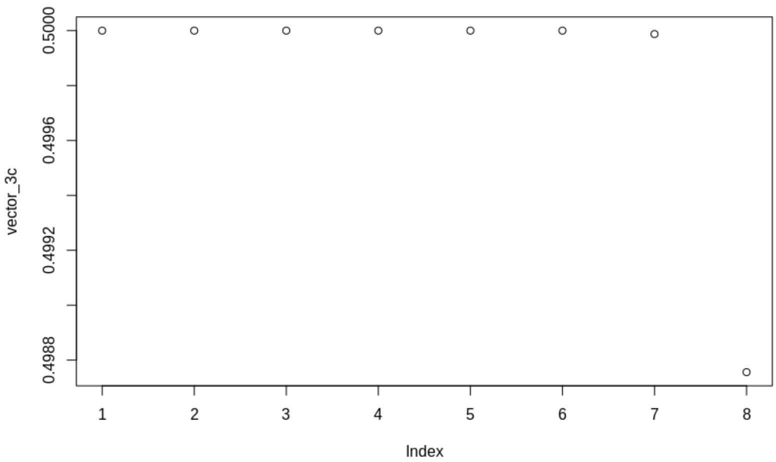
```
> task3c(10^(-32),0.5,formula)
[1] "next s: 0.0412414523193149"
[1] "sum now 0.490731195102493"
[1] "############"
[1] "next s: 0.00722076144732628"
[1] "sum now 0.497951956549819"
[1] "############"
[1] "next s: 0.00155064630199486"
[1] "sum now 0.499502602851814"
[1] "############"
[1] "next s: 0.000369372313384075"
[1] "sum now 0.499871975165198"
[1] "############"
[1] "next s: 9.37417160415066e-05"
[1] "sum now 0.499965716881239"
[1] "############"
[1] "next s: 2.48340667057848e-05"
[1] "sum now 0.499990550947945"
[1] "############"
[1] "next s: 6.78682523498095e-06"
[1] "sum now 0.49999733777318"
[1] "############"
[1] "next s: 1.8990311961232e-06"
[1] "sum now 0.499999236804376"
[1] "############"
[1] "next s: 5.41318907423135e-07"
[1] "sum now 0.499999778123284"
[1] "############"
[1] "next s: 1.56681562007829e-07"
[1] "sum now 0.499999934804846"
[1] "############"
[1] "calculation got too long"
[1] "final sum"
[1] 0.4999999
```

d) Compare results
small_x = c(10^(-32),10^(-28),10^(-22),10^(-18),10^(-12),10^(-8),10^(-4),10^(-2))

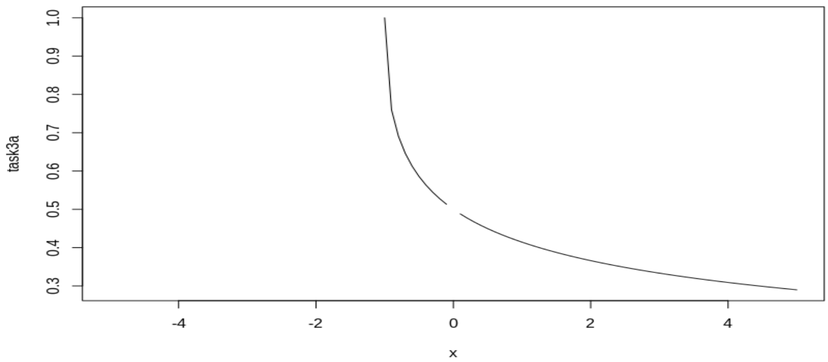
```
vector_3b <- c()
for(one_small_x in small_x)
	{
	formula <-  expression(sqrt(xe+1))
	result <- task3b(one_small_x,0.1,formula)
	vector_3b <- append(vector_3b, result)
	}
plot(vector_3b)
```



```
small_x = c(10^(-32),10^(-28),10^(-22),10^(-18),10^(-12),10^(-8),10^(-4),10^(-2))

vector_3c <- c()
for(one_small_x in small_x)
	{
	formula <-  expression((sqrt(xe+1)-1)/xe)
	result <- task3c(one_small_x,0.1,formula)
	vector_3c <- append(vector_3c, result)
	}
```



Visualize function:
plot(task3a, xlim=c(-5,5))



→ Function is not defined in point 0, however around x=0 the function goes to 0.5 on both sides. Problem: When you only estimate part of function, here sqrt(1+x), it make a difference to estimation of the whole function. In this case for x = 10^(-32) the interim result is 1 which is only rounded! From this it follows for x=0 related to whole function: $\frac{\sqrt{1+x}-1}{x}=\frac{1-1}{\infty}$

However, when we estimate the whole function, we set the numerator in relation to the denominator. Therefore, we end up with ca. 0.5 when the function goes to 0. Thats the correct result.