Computing a system's characteristic from a set of observations

Physics Labs in R

Dmitry Gorodnichy (Email: dg@ivim.ca)

'Prepared for R Ottawa, December 2020 (URL: https://rpubs.com/ivim/r101-w-physics01)'

Contents

Preface

Raison d'être

It's COVID outside. University students do all their studies online on a computer from home, including the "physical experiments". You would wonder, how is it possible to do "physical experiments" online ? - Very easy. Using a software, called Logger Pro.

So, what students do then in their Physics labs ? Instead of actually doing any physical experiments, they learn the software, and then use it to do "all experiments" and all related computations and visualizations.

How useful is it ? Maybe it is, if there's really no other way to do experiments yourself… But why not to teach students free data science tools such as R, which has become so powerful and so well supported by community over the past decade, to do all their data processing, visualization and modeling tasks, instead of learning some a proprietary "black box" software (which they may never understand what it actually does inside, and which they may never use again anyway after they are done with these Physics labs).

That was a chain of thought of that brought me to write this tutorial,and it all started with my daughter askeing me:
“Dad, I don’t understand what this software is doing when it draws a line through my points. Why can’t I just take the average of observations? Why do I have to use some kind of”software" to compute it for me (i.e., the line, the slope of which will be my answer)? What this software actually does? And, Why can’t I just do it myself?"

By that time, I’ve been already organizing the “R Lunch and Learn” series for my fellow GC colleagues interested in learning R. What a great way, I thought, to refresh my basics in Physics and Data Science, while helping others to learn Physics and Data Science too as well as to learn R at the same time!

So this is how it started.

How to make the best use of this article?

There’s more in this article than meets the eye. It is generated by a computer from the R Markdown script. The source of this script is made available in GitHub. It has all R language codes used to generate the results shown below. It has also plenty of English language “between the lines” thoughts (so called, “comments”) that you can’t see below, which you may also find interesting

It is posted there so that you can learn and use R Markdown yourself! - Just open https://rstudio.cloud, create a New R Markdown file there and paste there this script and *compile* it by clicking the “Knit” button and that’s it ! (note, when you compile for first time, it will ask you if you want d to install the three libraries that we use in the code - confirm yes - you’ll learn about them in codes below)

Disclaimer

This article presents a personal, hence rather subjective and opinionated, albeit good-intention-driven, view on how Data Science, R, Mathematics and Physics, could be taught for greater benefit of the students wanting to learn those disciplines and skills. It occasionally injects author’s thoughts and opinions on other subjects not related to the matters of the study, along with with the memories from his diverse cultural and educational past. If any of these appears politically, theoretically or practically incorrect to you – despite author’s efforts to get it all correct – please let the author know. I’m still learning myself. . .

Problem definition

One of distinctive abilities of homo sapiens that distinguishes us from animals is the ability to think *analytically*. - We can make conclusions about *something that we do not see* from other *things that we can see* - and we do it all the time, either intuitively or using scientific methods. Here we’ll talk about how to do scientifically - using physics, data science, and computer programming language called R. Click on the tabs below to read more about each of these three sides of the problem.

Physics side

In physics (as in a science that aims to understand the materialistic nature of life through reasoning and experiment), one of the main tasks is calculating a system’s characteristic that cannot be measured directly from other characteristics that can be measured directly. Some examples:

a. Finding the speed of a eco-friendly bicycle $V = \frac{d}{t}$ by measuring its mileage d over various periods of time t. In the same way for your environment polluting car, we can seek to find to the car gas consumption: *liters per 100 km (l/km)*, or *miles per gallon (mpg)* if you live in one of those few countries that still use the Imperial systems of measurements instead of the International System of Units (SI).

b. Finding the gravity acceleration g knowing that $h = g\frac{t^2}{2}$, where h is the height from which the object falls and t is the time when it reaches the ground.

c. Finding the initial speed of a soccer ball $V0$, knowing the angle β at which it was kicked and the distance d_x it reached, and the formula $V0 = \sqrt{\frac{gd_x}{sin2\beta}}$, which can be easily derived from the above two equations - by combining formulas for horizontal and vertical displacements:

- $d_x = t * V_x = t * V0 * cos\beta$, and $0 = d_y = V0 - g\frac{t^2}{2} = V0 * sin\beta - g\frac{t^2}{2}$, and knowing that
- $2\sin\beta\cos\beta = \sin 2\beta$ (If you don't know where this formula comes from, see how easily it can be obtained using this image!)

Mathematically speaking (or, as scientists say, "mathematically formalized"), all of these tasks are the same, described as follows:

Given:
- a set of observations $(x_i, y_i), i \in \{1, ..., N\}$ and
- the relationship that links the output (response) variable y to the input (trigger or factor) variable $y = K * x$,
Find: K;

Or, more specifically,
Find: such K that explains (or "fits") the best the observed data;

Or, even more specifically,
Find: such K that produces the least error between the observed values y_i and the "predicted" values $y_i^* = K * x_i : \sum_i^N (Y_i - Y_i^*)$.

Below we show how this can be done in R.

Data science side

Actually, there is not such science as "Data Science" (joke :). The term was coined just recently - in the second decade of XXI-st century, whereas people have been working with data for over a century prior to that.

There are however many other types of science that work with data: Natural Sciences (…), Social Sciences (…). And then there is a special type of "Science", which does not even belong in some universities to Science schools (e.g., at uOttawa and many others across North America), but belongs to the school of Engineering. It is called "Computer science" or "Computing science", or "Informatique" in French. This is where they study computer languages and how to build complex algorithms and computer systems, including now a very notorious thing called "Artificial Intelligence".

We'll talk more about that later. In meanwhile, go a head download the .Rmd source of this file from https://github.com/IVI-M/R-Ottawa/tree/master/physics-labs. Open it in R Studio (if you don't have R Studio installed on your machine, just go to https://rstudio.cloud/), Run it (by clicking "Knit" button), and voila ! - You just have built a "computer system" that is doing a job for you - loading data, computing formulas, and showing this article with results and comments!

You can now save your result - the created .html file, and email it to your friend or publish it on your personal page (e.g. in github) or RPubs - which is what we did with this article.

Congratulations - done! You are now almost a Computer Scientist! Almost . . . Anyway, you can safely start calling yourself a Data Scientist. - You now know how to manipulate data, visualize it and get the knowledge from it - which is what Data Science is commonly defined.

R side

This article is written in RStudio in R Markdown (.Rmd file) with

a) bits of LaTex code for writing mathematical formula, and
b) bits (they call it "chunks", since there rather big bits) of R code

This is how you may wish to write any scientific article - for peer reviewing and just for yourself (as it allows you to not lose any of your great ideas or results - by linking directly to the data and commenting out everything that is not ready for public eye yet)

In fact, any major belletristic or thoughtful writing will benefit greatly from this free editor and this meta-data driven form of composing and editing your work, not only scientific. For example, I also use it for my poetry translation and composing projects.

The question often asked - Why R? Not, Python - another popular language used in Data Science ? (According to latest observations, Python is still much widely used in GC than R, despite the fast growing base R users worldwide).

We'll get back on this later ! In brief, R is much younger than Python. - It grew out of Python, and has everything that Python, plus much more, including:

- Outstanding collection of graphic tools with `ggplot2` package and its many extenstions
- Outstandingly fast, memory and code efficient data processing of using `data.table`package
- Outstandingly helpful for code readability and algorithm design pipe operator `%>%`
- Free on-cloud IDE (Integrated Development Environment) for writing your R and other codes: rstudio.cloud
- Project-oriented design of RStudio, with embedded support for source control (Github) and on-line publication
- Free on-cloud portals for publishing and hosting your R -built solutions and web applications: . . .
- Ability to build and deploy interactive and data processing Web Apps that can be programmed to work with live data. You can be dead, but your code will run forever updating the world live on anything you want it to be updated after your death. . .
- Outstanding collection of free and interactive tutorials, many built in the RStudio
- Interactive and reactive data science and data visualization with Shiny
- R Markdown functionality that allows to mix codes and text any way you want
- Text editor that can be used for any epistolary works (not only your data science projects) and convert them to beautiful, ready to publish, documents
- Overall, RStudio appears more user friendly and faster to compiling and editing the code (based on my personal comparisons)

I have to admit here thought that could be unconsciously biased in making the above statements, as I stopped following Python development back in 2015, when, after playing for a year with both R and Python, I stopped using the latter and decided from now on to dedicate myself to what appeared to me then as a "clear winner" in terms of everything mentioned above.

At the same time, it is important to highlight that all languages are good and can be very powerful in the hands of professionals, of which, historically, Python has more. Therefore, it is often common to use both languages (and also C++) in the same project. In fact, RStudio allows you to easily write and compile codes in any of these languages and mingle them with each other anyway you want!

Warm-up excercises

Prior to building solutions to the physics problems using R, let's set the foundation for what is called "robust scalable programming" in R.

Key R concepts

R is the language that Computer understands.
So, you need to translate your knowledge of the problem and the tasks you want Computer to do for you into this language in such a way:

- firstly, Computer understands you and does exactly what you want it to (and without any "biases"!), and - secondly, your co-workers can understand it too (including yourself - when you come next week or next year!)

As they say, bad habits are worse than bad mistakes. Everyone can strum guitar strings, but if you want to *play guitar* instead of just making sounds, you need to learn the proper hand posture, master the theory and train your ear. The same with Computer programming - there a few skills that you'll need to master, and then writing any code (whether in R, Python, C++, or Java) will be a breeze fro you - just like playing any Christmas song by ear without needing to see its chords.

This article (with all R code chunks inside it) is carefully crafted in such a way as to help you to master these skils.

Click on the tab at right to start building the right habits for Computer programming - the foundations of what is called "robust scalable" coding. Click on [Code] buttons below to see the codes for everything that is computer generated in the article.

R foundations: Your first code describing a real physical system

This code describes the Computer your knowledge of the laws of gravity and asks the Computer to compute the trajectory of an object.

Open the code to see what it does. It introduces you to the following concepts:

- commenting in and out the codes
- three actually needed libraries (there are hundreds of others - sometimes very useful, but you actually don't really need them - when you start learning. Only when you master these three and the concepts described here, should you start adding other libraries)
- constants vs. simple variables, their type,
- importance of properly naming: "Hungarian" notation"
- array of simple variables
- concept of a complex variable (data.table) that "discribes" a physical system

- functions (aka procedures)
- viewing the data with printing and plotting

```r
# Warm up exercise to help you to write your first R code:
# Experiment 1: examine  trajectory of a ball kicked  by soccer player

# These are three MOST needed libraries:

library(magrittr);  # to chain :  1:10 %>% sin %>% round (2)  instead of  round(sin(1:10), 2)
library(data.table);# to use tables: dt <- data.table(x=1:10)  %>% .[, y:= sin(x) %>% round(2)]
options(datatable.print.class = TRUE)  # for printing data.tables with extra information (OPTIONAL)
library(ggplot2);   # to plot: ggplot(dt) + geom_line(aes(x,y))

# These are constants:

g <-  9.8 # m/s^2

# These are variables and their values.

# NB: Unless variables are used locally  (i.e only once and here) or are obvious, they should be named

# Simple variable:

time_start <- 0   # time you start / finish observing (secs). It is numeric (i.e. floating point) numbe
time_end <- 3         # Better name :  fTimeEnd
nObservations <- 5L  # number of observations (inter). It is integer number.

# Array of simple variables:

# aObservations <- 0:nObservations;
aTimes <- seq(time_start, time_end, (time_end-time_start)/nObservations); # Better name: afTimes
aTimes %>% print()  # NB:  you don't need print(x), can just type x, unless  inside  function or loop!
```

```
## [1] 0.0 0.6 1.2 1.8 2.4 3.0
```

```r
aTimes %>% length()
```

```
## [1] 6
```

```r
V0 <- 30 # initial ball speed: min/sec
alpha <- 45 * pi / 180 # angle over horizon: degrees converted to radians
h0 <- 0 #initial heiaght: meter

# This is a function
mps2kmph <- function (speed){  # Another good name:  convertMinPerSec2KmPerHour()
  speed * ( 1 / 1000) / (1 / 3600);  # 1 km = 1000 m; 1 h = 3600
}
mps2kmph(V0)
```

```
## [1] 108
```

```r
# Complex variable: data.table (also known as "improved data.frame"")
# NB: Complex variable describes a complex system. - Each row represents a system state at time t

dtKickedBall <- data.table(t = aTimes);
dtKickedBall %>%
  .[, observation := 1:.N] %>%
  .[, ':='(
    x = V0 * cos(alpha) * t,
    y = h0 + V0 * sin(alpha) * t - g * t^2 / 2
    # Vx = V0 * cos(alpha),
    # Vy = V0 * sin(alpha) - g * t
  ) ]
# NB 1: There's no assignment operator ' <- ' ! New columns are added to existing data.table !
# NB 2: Note use of piping and multiple (vs. single) column modification
# NB 3: columns names in data.table do not need  follow naming conventions - you  print them anyway
dtKickedBall
```

```
##        t observation        x        y
##    <num>       <int>    <num>    <num>
## 1:   0.0           1  0.00000  0.00000
## 2:   0.6           2 12.72792 10.96392
## 3:   1.2           3 25.45584 18.39984
## 4:   1.8           4 38.18377 22.30777
## 5:   2.4           5 50.91169 22.68769
## 6:   3.0           6 63.63961 19.53961
```

```
# Now we can do investigation about this system:

dtKickedBall[, c("t","y")] # show ("select", extract, or subset vertically)  data
```

```
##        t        y
##    <num>    <num>
## 1:   0.0  0.00000
## 2:   0.6 10.96392
## 3:   1.2 18.39984
## 4:   1.8 22.30777
## 5:   2.4 22.68769
## 6:   3.0 19.53961
```

```
# dtKickedBall[, c(1,4)]   # the same
# dtKickedBall[, .(t,y)]   # the same (NOTE the use of . when referring to columns wo "")

# dtKickedBall$y       # show ("pull", extract)  column as array
dtKickedBall[["y"]]  # the same
```

```
## [1]  0.00000 10.96392 18.39984 22.30777 22.68769 19.53961
```

```
# dtKickedBall[2] # show the state at second row
dtKickedBall[c(1:2,(.N-2):.N)] # show ("slice", subset horisontally) the first and last two rows
```

```
##        t observation        x        y
##    <num>       <int>    <num>    <num>
## 1:   0.0           1  0.00000  0.00000
## 2:   0.6           2 12.72792 10.96392
## 3:   1.8           4 38.18377 22.30777
## 4:   2.4           5 50.91169 22.68769
## 5:   3.0           6 63.63961 19.53961
```

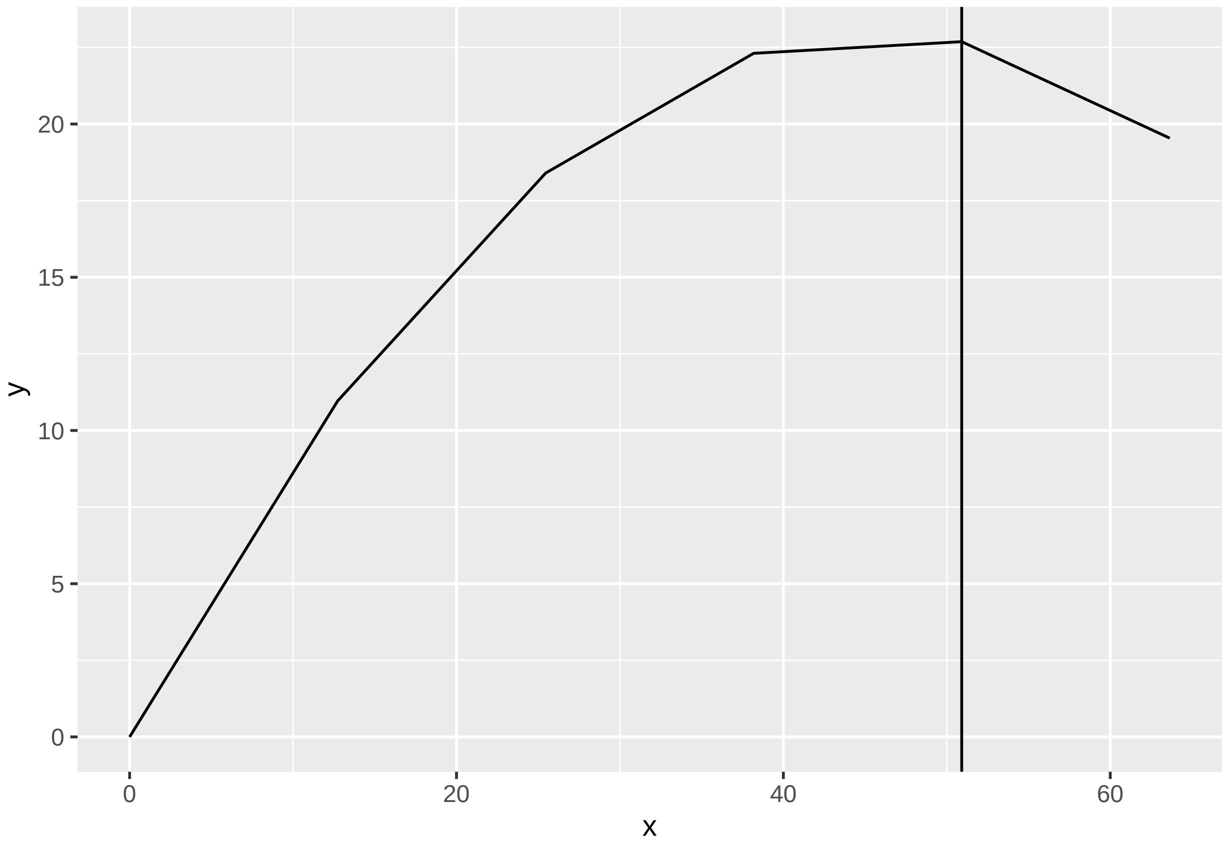
```
dtKickedBall[y>0] # show ("filter") all states above the ground
```

```
##        t observation        x        y
##    <num>       <int>    <num>    <num>
## 1:   0.6           2 12.72792 10.96392
## 2:   1.2           3 25.45584 18.39984
## 3:   1.8           4 38.18377 22.30777
## 4:   2.4           5 50.91169 22.68769
## 5:   3.0           6 63.63961 19.53961
```

```
dtKickedBall[which.max(y)] # find the state when the ball is at its heighest spot
```

```
##        t observation        x        y
##    <num>       <int>    <num>    <num>
## 1:   2.4           5 50.91169 22.68769
```

```
# dtKickedBall[which.min(abs(Vy))] # ... when the ball started to descend - the same as above


# ... and plot the results

g <- ggplot (dtKickedBall) +
  geom_line(aes(x,y)) +
  geom_vline(xintercept =dtKickedBall[which.max(y)]$x)
g # or print(g)
```



```
# NOTE the use of "aes(...)" when plotting data.table internal variables x and y!


# FINAL step:
# Want to plot the system with different settings ? -
# Encapsulate everything above in a function that depends on the variables you want to change !

fKickedBall <- function(time_start, time_end, nObservations, V0, alpha, h0) {
  # NB: to print/plot  from inside a function, you need to  print(...)`
}
```

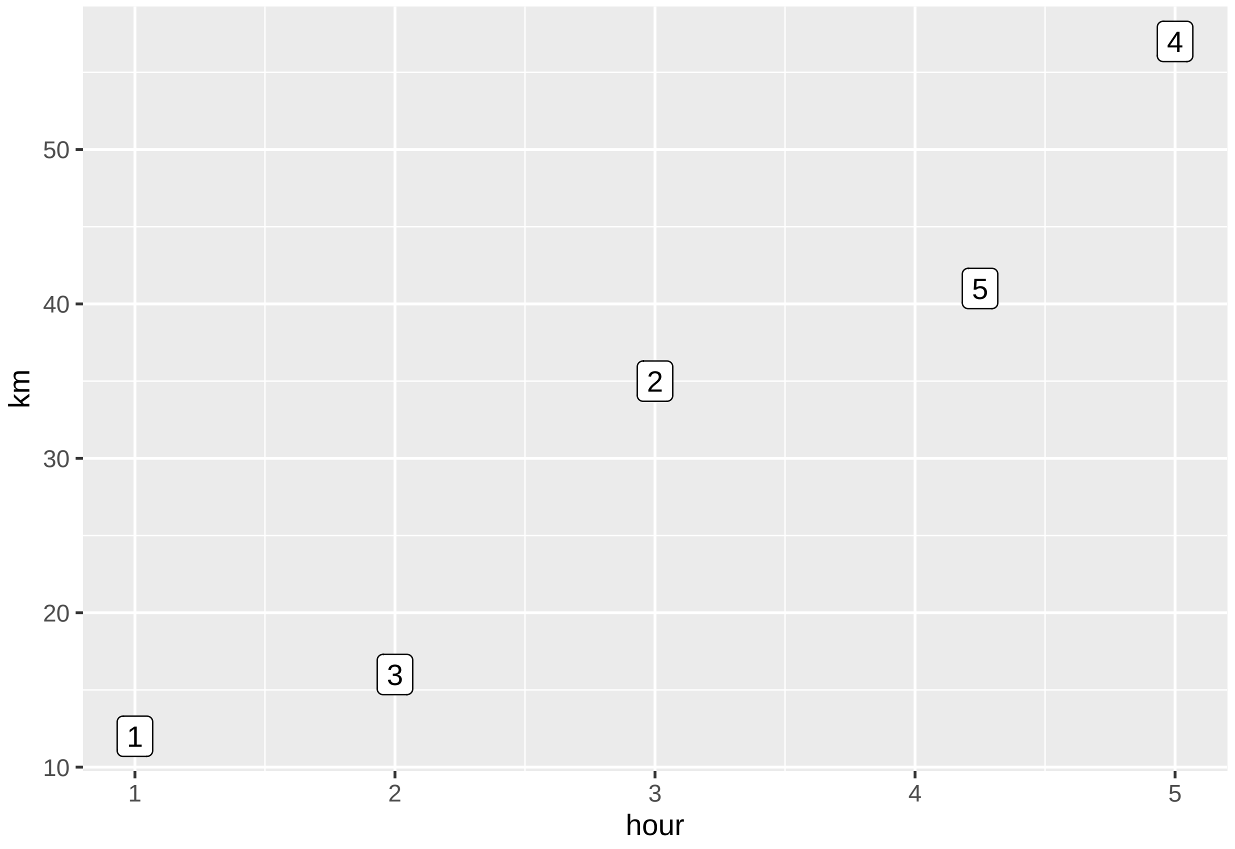
```
# To learn more, goto: https://github.com/IVI-M/R-Ottawa/blob/master/resources.md
```

Problem 1: Finding average speed (or miles per gallon)

Lets say, we made 5 measurements ("km" and "hour") for a moving object, and need find the average car speed $V = km/hour$.

The first step in any analysis is plotting the measurements:

```
library(data.table);
library(magrittr);
hour <- c(1,  3,  2, 5, 4.25)
km <- c(12, 35, 16, 57, 41)
dt <- data.table(km=km, hour=hour)

dt[, observation:=1:.N]

g <- ggplot(dt) +  geom_label(aes(x=hour,y=km, label=observation))
g
```



How do we go now to compute the object speed, asssuming that a) the speed was the constant, and knowing that b) the measurements in real life are never perfect.

Approach 1 - Average

One way of computing the object speed is to compute V_i as $V = km_i/hour_i$ for each measurement and then take the average of it:

```
dt$V <- round( dt$km / dt$hour, 1)
# dt %>% kable()
dt
```

```
##        km  hour observation     V
##     <num> <num>       <int> <num>
## 1:     12  1.00           1  12.0
## 2:     35  3.00           2  11.7
## 3:     16  2.00           3   8.0
## 4:     57  5.00           4  11.4
## 5:     41  4.25           5   9.6
```

```
V.ave1 = mean(dt$V) %>% round(1); V.ave1
```

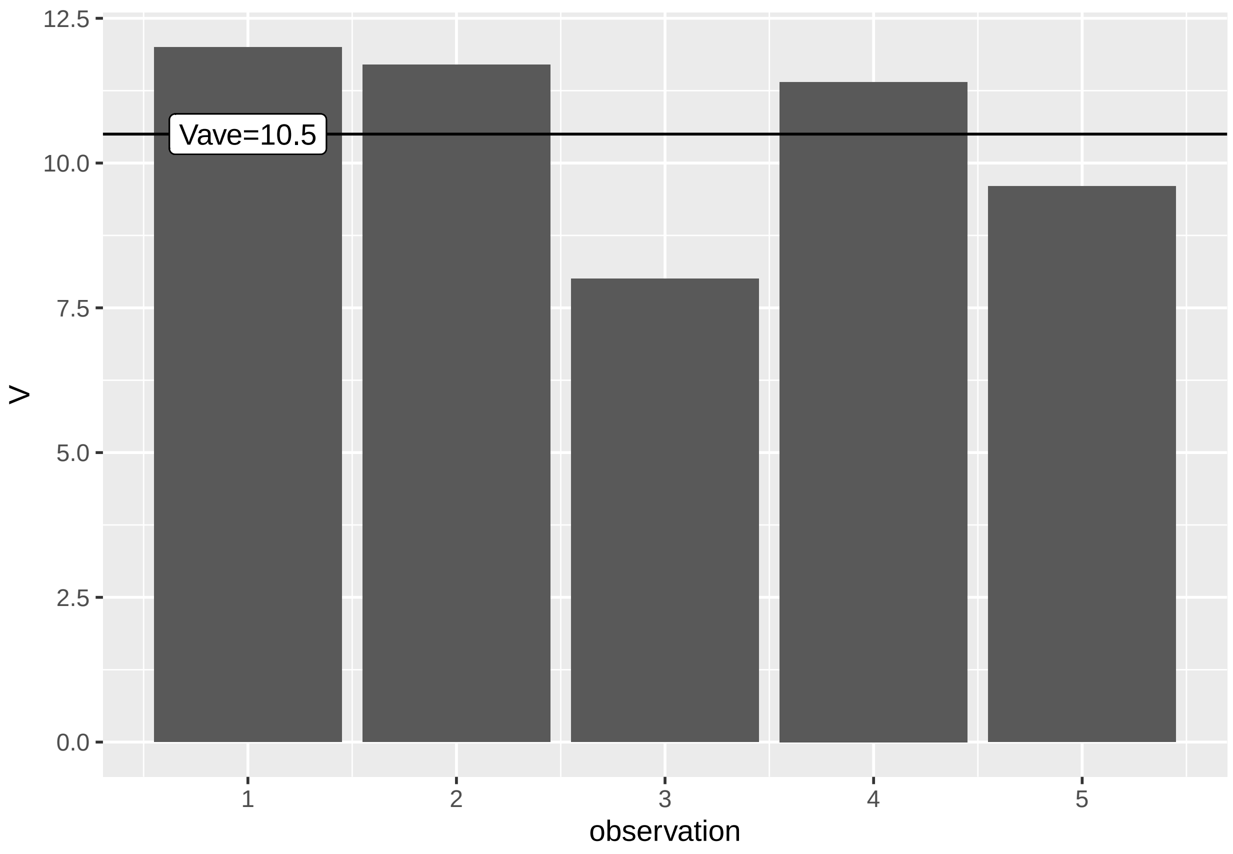
```
## [1] 10.5
```

```
V.sd1 = sd(dt$V) %>% round(1); V.sd1
```

```
## [1] 1.7
```

This gives us $V_{ave} = 10.5 \pm 1.7$

```
ggplot(dt) + geom_col( aes(observation,V) ) +
  geom_hline(yintercept = V.ave1) +
  geom_label(aes(1,V.ave1,label=paste0("Vave=", V.ave1)))
```



This was easy enough and can be computed without any computer software!

Yet there's also another way to find V that is "mathematically more precise" but which cannot be done without a computer - it's called regression or building a linear model!

Approach 2 - Regression

Knowing that the relationship between input (km) and output (hour) is linear, we "ask" computer to find the line that "fits" the measurement points "as close as possible" - the process called regression. The slope of such line will be the speed of the object. In R, this is done in one line:

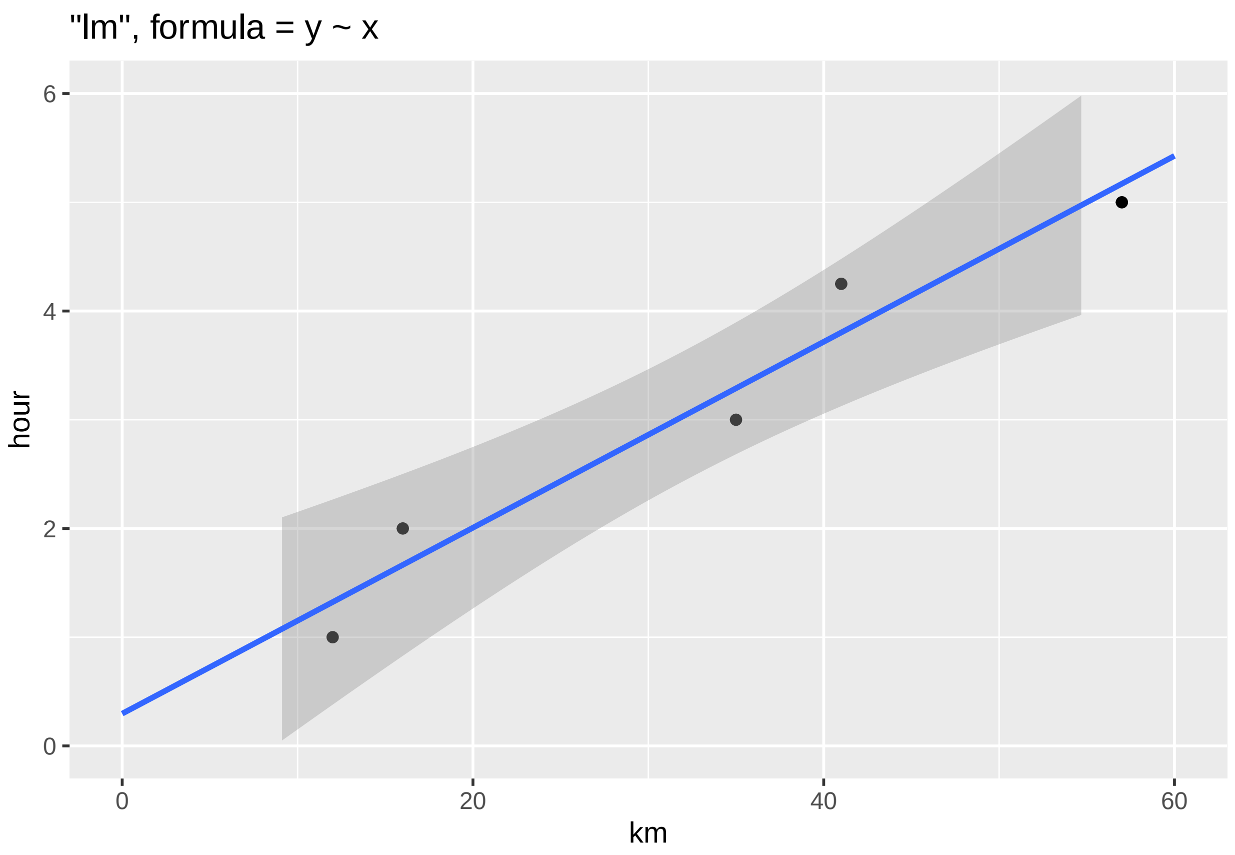
```
# dt <- dt %>% rbind( data.table(0,0,0,0),use.names=FALSE )
# V.ave1 = mean(dt$V) %>% round(1); V.ave1
# V.sd1 = sd(dt$V) %>% round(1); V.sd1
# dt[, observation := 1:.N]

# https://rpubs.com/aaronsc32/regression-through-the-origin

model0 <- lm( dt$km  ~ 0 + dt$hour ) %T>% print # # Adding the 0 term tells the lm() to fit the line th
```

```
##
## Call:
## lm(formula = dt$km ~ 0 + dt$hour)
##
## Coefficients:
## dt$hour
##   10.66
```

```
# str(model0);
# summary(model0)
# model0$coefficients
V.as_pente <- model0$coefficients[1] %>% round(1); V.as_pente
```

```
## dt$hour
##    10.7
```

```
ggplot(dt,  aes(km,hour)) + geom_point( ) +
  geom_smooth(method = "lm", formula = y ~ x, fullrange=TRUE) +
  xlim(0,60) + ylim(0,6) +
  labs(title='"lm", formula = y ~ x')
```



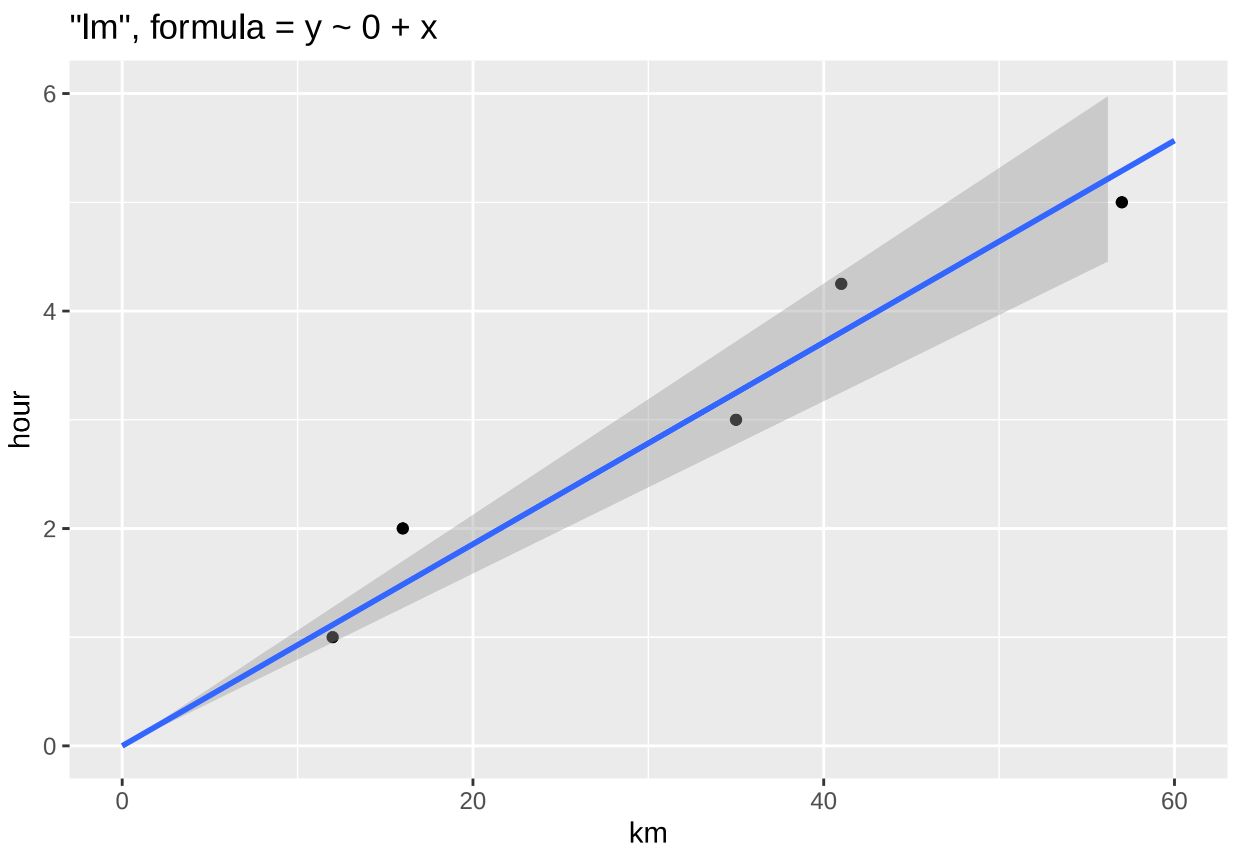
```
model1 <- lm( dt$km  ~ dt$hour )   # # Adding the 0 term tells the lm() to fit the line through the ori
model1
```

```
##
## Call:
## lm(formula = dt$km ~ dt$hour)
##
## Coefficients:
## (Intercept)      dt$hour
##      -1.682       11.109
```

```
# str(model1);
# summary(model1)
# model1$coefficients
V.as_pente1 <- model1$coefficients[2] %>% round(1); V.as_pente1
```

```
## dt$hour
##    11.1
```

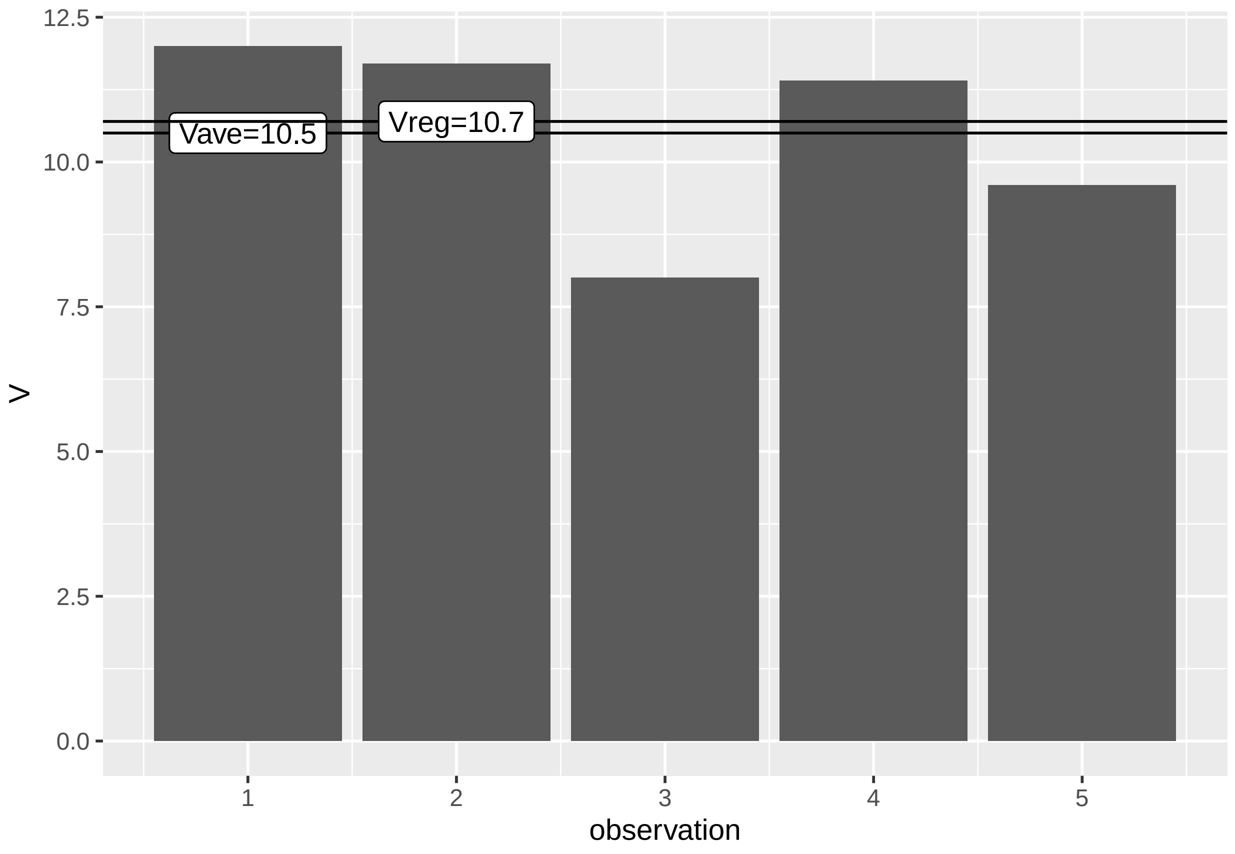
```
ggplot(dt,  aes(km,hour)) + geom_point( ) +
  geom_smooth(method = "lm", formula = y ~ 0 + x, fullrange=TRUE) +
  xlim(0,60) + ylim(0,6) +
  labs(title='"lm", formula = y ~ 0 + x')
```



This gives us $V_{ave} = 10.7$

What did you notice? - The result is different!

```
dt[, observation:=1:.N]
ggplot(dt) + geom_col( aes(observation,V) ) +
  geom_hline(yintercept = V.ave1) +
  geom_label(aes(1,V.ave1, label=paste0("Vave=", V.ave1))) +
  geom_hline(yintercept = V.as_pente) +
  geom_label(aes(2,V.as_pente, label=paste0("Vreg=", V.as_pente)))
```



Why we have different results?

Let's understand what's going on.

Effort to find an explanation Let's add FIVE "new" points, which are actually not new, but are the first one repeated FIVE more times, and see what changes.

```
dt <- dt %>% rbind( dt[1]);
dt <- dt %>% rbind( dt[1])
dt <- dt %>% rbind( dt[1])
dt <- dt %>% rbind( dt[1])
dt <- dt %>% rbind( dt[1])

V.ave1 = mean(dt$V) %>% round(1); V.ave1
```

```
## [1] 11.3
```

```
V.sd1 = sd(dt$V) %>% round(1); V.sd1
```

```
## [1] 1.4
```

```
dt[, observation:=1:.N]


model1 <- lm( dt$km ~ dt$hour ); model1
```

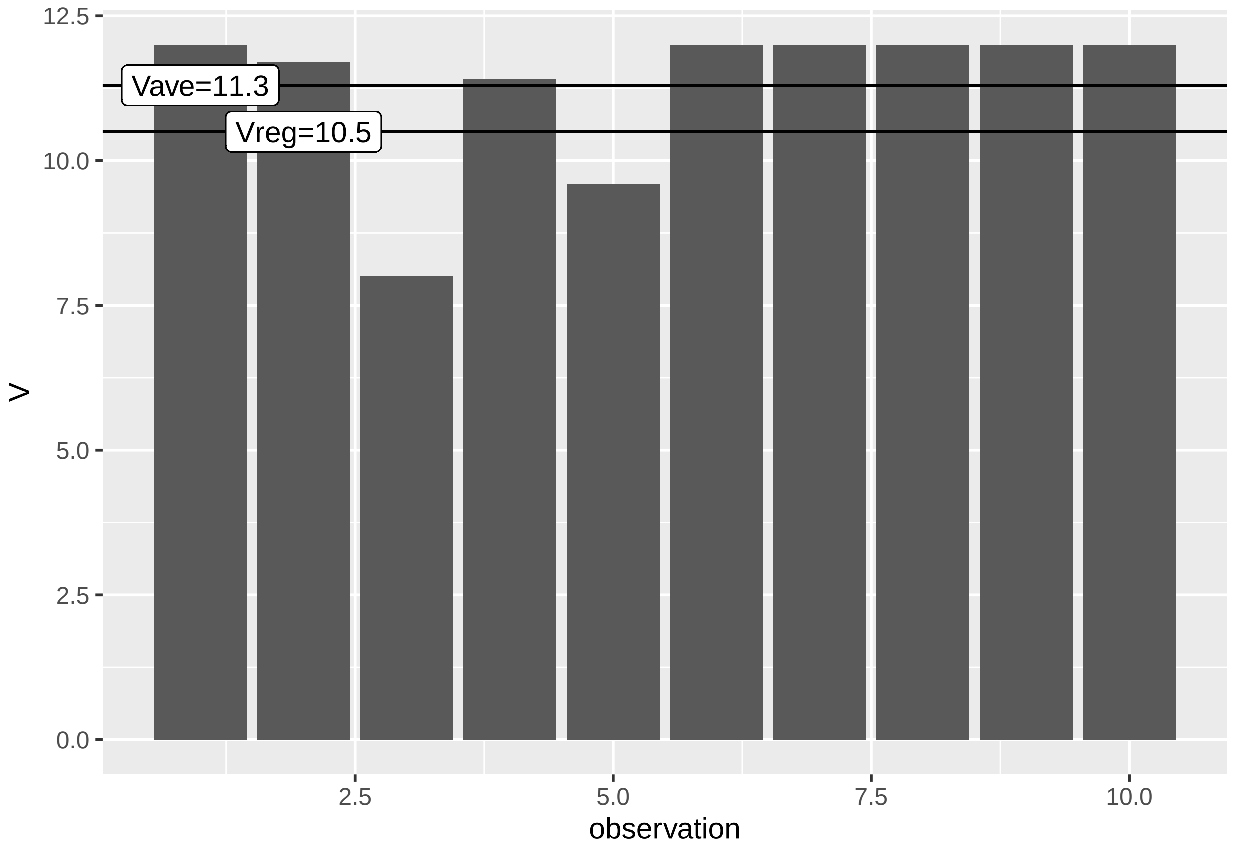
```
##
## Call:
```

```
## lm(formula = dt$km ~ dt$hour)
##
## Coefficients:
## (Intercept)      dt$hour
##      0.8727      10.4826
```

```
V.as_pente <- model1$coefficients[2] %>% round(1); V.as_pente
```

```
## dt$hour
##    10.5
```

```
ggplot(dt) + geom_col( aes(observation,V) ) +
  geom_hline(yintercept = V.ave1) +
  geom_label(aes(1,V.ave1, label=paste0("Vave=", V.ave1))) +
  geom_hline(yintercept = V.as_pente) +
  geom_label(aes(2,V.as_pente, label=paste0("Vreg=", V.as_pente)))
```



We see that $Vreg$ computed using regression , as slope of the fitting line, is (almost) the same, whereas $Vave$ computed as average has become smaller. This is because we added five more times the smallest point, which lifted the average, but did not change the slope! So now you know it.

Remaining Question: The above example describes the case of dependent (or non evenly measured) observations. In this case, indeed regression helps to deal with such dependency (or uneven sample class distribution).

However, what if all new measurements are independent, each one adding a new unbiased piece of evidence - what they call in science, are *iid* (Independent and identically distributed) random variables? How do we compare Average to Regression then?

If you know, please let me know !

Inside the "regression"

Among other questions, you may still wonder about two things:

1. What is actually computer doing in this intimidatingly sounding process called "regression"?, and
2. What if you don't have computer? How then should you compute the average speed ?

Effort to find an explanation Remember, above we said that regression finds the line that fits the points "as close as possible" ? Mathematically speaking, this means that computer seeks to find such V to minimize the total fitting error, which can be written as follows:

$$\sum_{i \in 1,\dots,N} (km_i * V - hour_i)^2 \to min$$

To find the extremity points of a function, you need to find where its derivative is equal to zero, so lets do it:

$$\sum_{i \in 1,\dots,N} 2km_i * (km_i * V - hour_i) = 0$$

from where we get:

$$V = \frac{\sum 2km_i * hour_i}{\sum 2km_i^2}$$

This is actually the reason, why scientists like measuring errors using a quadratic function $L_2 = (\Delta x)^2$ - because it is the easiest to take the derivative from! They call it the L_2 metric (or norm). However, they are also other kinds of norms,e.g., such as $L_1 = |x|$ which are more preferable for robust analysis of data that have outliers (i.e. points that are far from average) - Can you guess why?

Now, lets check the formula using the lab data:

```r
sum( 2* dt$km  *dt$hour) / sum( 2* dt$km^2)
```

```
## [1] 0.09185567
```

Hmmm ... Something must be wrong ... Help !...

If you know what's going on there, please let me know !!

Problem 2: Finding gravity acceleration

Here we want to calculate g knowing that $h = g\frac{t^2}{2}$, also knowing, lets say find measurements that we made by observing a falling object - using the metholodologies described above.

What's difference of this problem from the previous one?

Two things:

- Relationship between input (trigger) and output (response)variables is not linear
- The value that we seek to find is NOT the average among several possible values a variable can take, but the constant !

TBD

Problem 3: Finding the ball speed

Can you do it yourself now, using solutions from above?

We'll do something different and more practical - we'll apply our knowledge to build an anti- ballistic missile system - one of those developed during the Cold War.

TBD

```
dtProjectile
```

```
##    angle distance     sin2b       V0
##    <num>    <num>     <num>    <num>
## 1:    15      5.1 0.5000000 9.998000
## 2:    30      8.0 0.8660254 9.514648
## 3:    45     10.0 1.0000000 9.899495
## 4:    60      8.5 0.8660254 9.807474
## 5:    75      4.8 0.5000000 9.699485
```

```
V0.ave <- mean(dtProjectile$V0, na.rm = T); V0.ave
```

```
## [1] 9.78382
```

```
V0.sd <- sd(dtProjectile$V0, na.rm = T); V0.sd
```

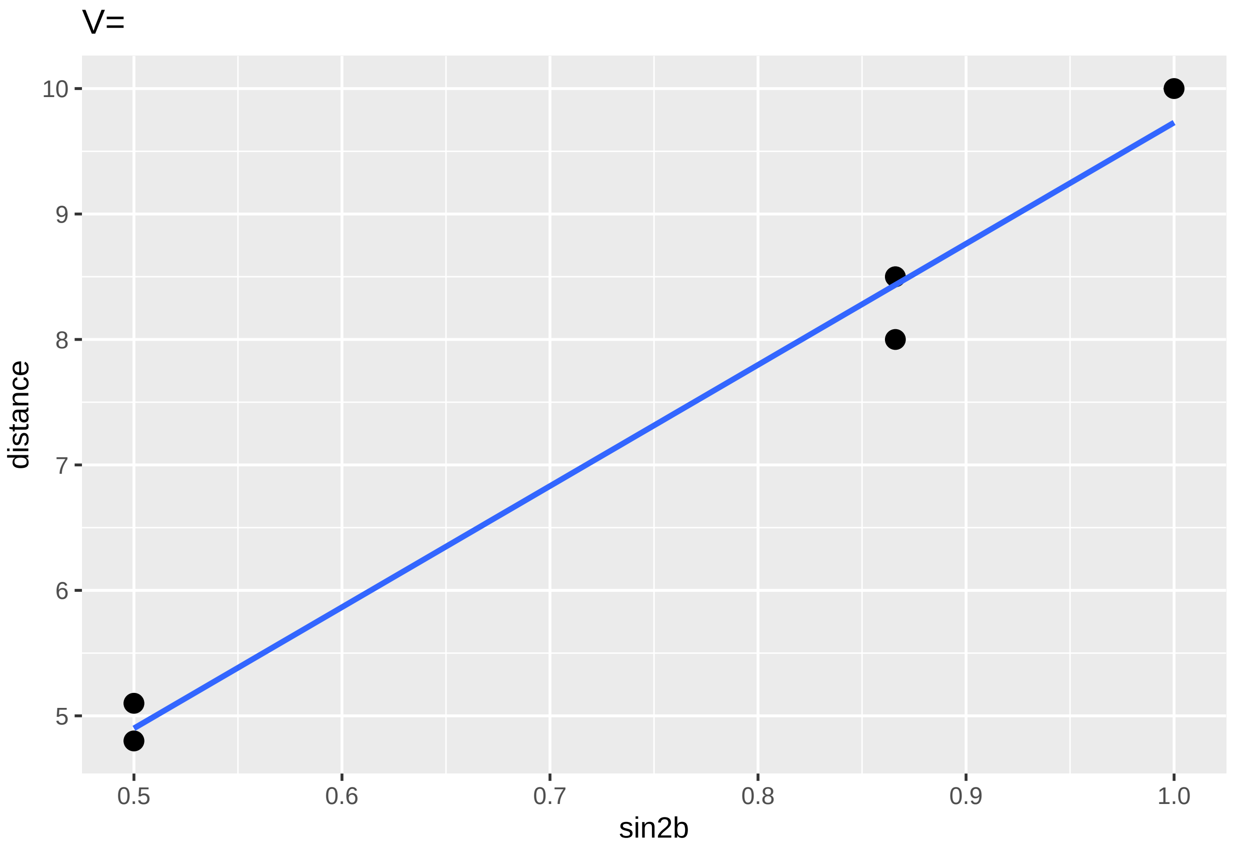
```
## [1] 0.1866677
```

```
V0.se <- V0.sd /sqrt( length(dtProjectile$distance)); V0.se
```

```
## [1] 0.08348035
```

```
# ORqplot(x=sin2b,y=distance, data=dtProjectile, geom="point")
ggplot(dtProjectile, aes(x=sin2b,y=distance)) +
  geom_point(size=3) +
  geom_smooth(method = "lm", se=F) +
  labs(title="V=")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
model1 <- lm( dtProjectile$distance ~  dtProjectile$sin2b)
# str(model1)
model1$coefficients
```

```
##        (Intercept) dtProjectile$sin2b
##         0.07184394         9.65709798
```

```
str(model1$coefficients)
```

```
##  Named num [1:2] 0.0718 9.6571
##  - attr(*, "names")= chr [1:2] "(Intercept)" "dtProjectile$sin2b"
```

```
V0.as_pente <- model1$coefficients[2]; V0.as_pente
```

```
## dtProjectile$sin2b
##           9.657098
```

–>