Presentation of the course and setting up R

MATH 2740 – Mathematics of Data Science – Lecture 01

Julien Arino
julien.arino@umanitoba.ca

Department of Mathematics @ University of Manitoba

Fall 202X

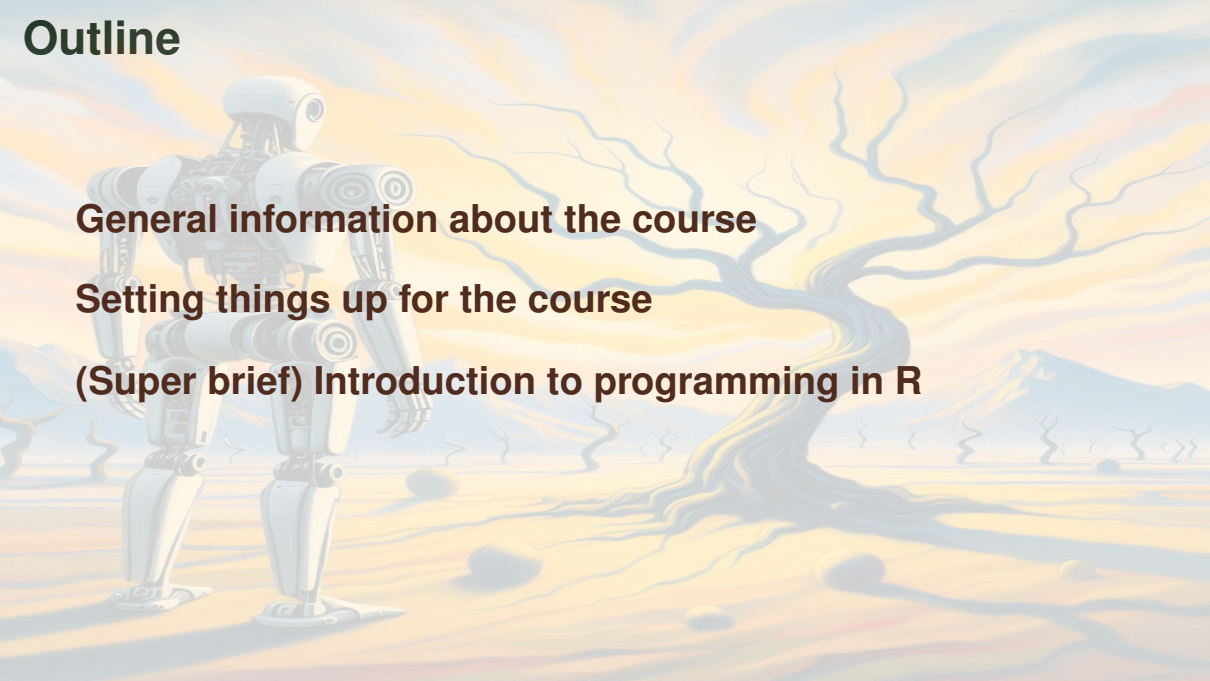
The University of Manitoba campuses are located on original lands of Anishinaabeg, Ininew, Anisininew, Dakota and Dene peoples, and on the National Homeland of the Red River Métis. We respect the Treaties that were made on these territories, we acknowledge the harms and mistakes of the past, and we dedicate ourselves to move forward in partnership with Indigenous communities in a spirit of Reconciliation and collaboration.

Outline

General information about the course

Setting things up for the course

(Super brief) Introduction to programming in R



General information about the course

Setting things up for the course

(Super brief) Introduction to programming in R

Foreword

- "Numerical dates" are in the form YYYY-MM-DD (e.g, these slides were compiled on 2025-11-04)
- Times are 24h (HHMM)
- Units are SI

In case you want to know, the slides in the course are `Rnw` compiled using `R`. All (including source code) are available on GitHub here

Getting in touch

- julien.arino@umanitoba.ca
- Please use your `myumanitoba` email address. Use a tag such as `[MATH 2740]` in your subject line, if you want to be read..
- Word of warning: I am bad with email! I do answer questions after class or during office hours

Office hours

- Because of the ongoing renovation of Machray Hall, I am sharing an office with 8 other colleagues. Next door are offices shared by another 8 and 4 colleagues

- It is therefore **impossible** for me to see you in my office

- I have booked 238 St Paul's from 1430 to 1600 on Tuesday and Thursday for office hours

  Seeing you outside of office hours is impossible without ample warning, since I actually need to book a room

Course website - UMLearn

- All information about the course is posted on UMLearn
- It is **your responsibility** to check the UMLearn site regularly: *Announcements* is how I normally communicate with you about the course
- (Remember to hit the link at the top of the page that says *MATH-2740-A01 - Mathematics of Data Science*, sometimes UMLearn takes you directly to Content, which is not where Announcements are)

Important – I **DO NOT** answer emails that ask about things easy to find on UMLearn

Lectures

- TR 1130-1245 in 200 Fletcher Argue

- Videos for the course as I taught it in 2021 are available as a YouTube playlist. There is no guarantee that that the content will be the same this year, but there will be commonalities for sure

Tutorials

- In tutorials, you will review some of the mathematical content and work on computations
- Short (15 minutes) quizzes will be held

Section	Day and time	Location
B01	W 0830-0920	301 Biological Sciences
B02	W 0930-1020	301 Biological Sciences
B03	W 1130-1220	301 Biological Sciences
B04	W 0830-0920	505 Tier

Evaluation

- 1 final examination during the Examination period (**35%**)
- 1 midterm on Monday 27 October 1900–2100 (**25%**)
- 4 quizzes during tutorials (**5%** each for a total **20%**)
- 4 assignments (**5%** each for a total **20%**)

Quizzes

- Held during tutorials
- 15 minutes long
- One of the timeslots (0830, 0930 or 1130) writes on a given week
- Tests your capacity to do simple proofs or remember definitions or results

Assignment schedule

Assignment number	Handed out	Due
Assignment 1	19 September	3 October
Assignment 2	10 October	24 October
Assignment 3	31 October	21 November
Assignment 4	21 November	5 December

Assignments

- Assignments will be computational
- **No tolerance for late assignments** and no make-up for missed assignments: any assignment not returned by the deadline will result in a mark of **zero**
- Computer assignments will need to be handed back in `R` (Python $\Rightarrow$ 0)

CS students (in particular), beware!

- Use Rmarkdown to generate a **notebook**. See more on this in Lecture 02
- Notebooks mix formatted text and code. They are executable and should be submitted as source, not as pdf or html or whatever. Only files in .Rmd are accepted
- Notebooks are not straight code. Submitting straight R code in a notebook with commented code $\Rightarrow$ 0)

Returning assignments

- Assignments go to UMLearn
  - R language only (Python $\Rightarrow$ 0)
  - Needs to be RMarkdown file (.Rmd)
  - Single file
  - Can submit several times but only the latest file will be used

- Your code must run! It must also use the "Be friendly to others" method in these slides

In assignments/quizzes/midterm/final..

- Explain what you are doing
- Math or code without explanation will lose marks

Self-declaration of absences

- You can self-declare an absence of less than 120 hours (5 days)
- Self-declarations are intended for occasional and unforeseen circumstances $\Longrightarrow$ I **will not accept more than one** during the term
- Self-declarations must be filed less than 48 hours after the event you are using it for
- There is no make up for anything missed because of a self-declared absence: any percentage will be moved to the final examination
- See the details and the form here (link). You **must** use the form on that page to self-declare your absence

Academic dishonesty

- Feel free to discuss work with others, but solutions must be your own!
- Markers will be on the lookout for this
- Paraphrasing my computer code = academic dishonesty !
- stack overflow is a fantastic resource but if you use a solution from there, cite it (in a notebook, that's easy)
- ChatGPT, GitHub Copilot, etc. are wonderful tools, but you must use them wisely. Pure unaltered LLM production $\Rightarrow$ AD
- FYI: my PhD student who is marking your computer code and some of your math has been working with LLMs for quite a while now. Their LLM detection radar is finely tuned

Definitions are colour coded

Memorising the definitions is part of the course. To help, definitions are colour coded

Definition 1 (Definitions)

These definitions are important, you need to know them

Definition 2 (Less important definitions)

These definitions are a little less important, you will not be asked to state them (although it is a good idea to know them anyway)

Results are colour coded

Memorising some of the results is part of the course. To help, results are colour coded

Theorem 3 (Theorems)

Theorems in blue boxes are worth knowing but you will not be asked to reproduce them

Theorem 4 (Important theorems)

Theorems in red boxes are important, you should know them and be able to reproduce them

You must know how to do some proofs

There are a few proofs (not many!) that I want you to know how to do

Such proofs appear on slides like the present one, with a red background

General information about the course

Setting things up for the course

(Super brief) Introduction to programming in R

We will be programming quite a bit

As already indicated, Data Science is a very hands-on discipline. We will be programming quite a bit in this course

Indeed, we can work out some of the examples "by hand", but to make things interesting, we typically need to consider larger examples where hand calculations are not pleasant or not even feasible

R versus Python

Slightly different take on life :)

In short: `Python` is more CS, `R` is more Stats/Math

Both are good languages for data science

In this course, assignments **must** use `R`

R was originally for stats but is now more

- Open source version of S
- Appeared in 1993
- Now version 4.5
- Uses a lot of C and Fortran code. E.g., `deSolve`:

  The functions provide an interface to the FORTRAN functions 'lsoda', 'lsodar', 'lsode', 'lsodes' of the 'ODEPACK' collection, to the FORTRAN functions 'dvode', 'zvode' and 'daspk' and a C-implementation of solvers of the 'Runge-Kutta' family with fixed or variable time steps

- Very active community on the web, easy to find solutions

Getting your computer ready for the course

All computer coding is in R; assignments also need to be returned in R

$\Longrightarrow$ you need to find a way to run R. Next slide: some methods, from the easiest to the most challenging.

Note that all options described below are Open Source (completely free)

In short...

- Terminal version, not very friendly
- Nicer terminal: radian
- Execute R scripts by using `Rscript name_of_script.R`. Useful to run code in `cron`, for instance
- Use IDEs:
  - RStudio has become the reference
  - RKWard is useful if you are for instance using an ARM processor (Raspberry Pi, some Chromebooks..)
- Integrate into jupyter notebooks

Install R and RStudio

This is probably the best option if you intend to go a little further than what we will do in the course. `R` is available on most platforms, while `RStudio` is available on most platforms except for Linux ARM devices (but can be compiled there)

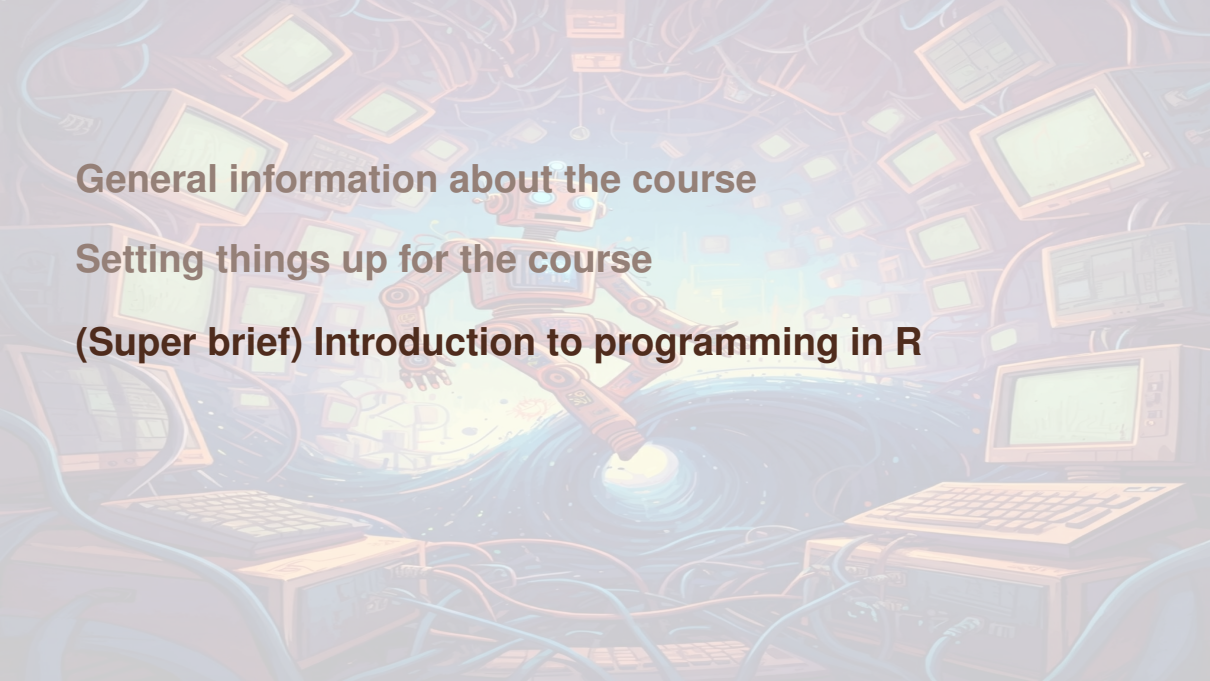
Visit https://www.r-project.org/

Choose your version: Windows or Mac. Under Linux, you can install directly from your package manager (e.g., `sudo apt install R-base` for Debian-based distros)

To install RStudio, see here

Going further

- RStudio server: run RStudio on a Linux server and connect via a web interface
- Shiny: easily create an interactive web site running R code
- Shiny server: run Shiny apps on a Linux server
- Rmarkdown: markdown that incorporates R commands. Useful for generating reports in html or pdf, can make slides as well..
- Sweave/knitr: LaTeX incorporating R commands. Useful for generating reports. Not used as much as Rmarkdown these days (but used for these slides)



General information about the course

Setting things up for the course

(Super brief) Introduction to programming in R

R is a scripted language

- Interactive
- Allows you to work in real time
  - Be careful: what is in memory might involve steps not written down in a script
  - If you want to reproduce your steps, it is good to write all the steps down in a script and to test from time to time running using `Rscript`: this will ensure that all that is required to run is indeed loaded to memory when it needs to, i.e., that it is not already there..

Assignment

Two ways:

```
X <- 10
```

or

```
X = 10
```

First version is preferred by R purists.. I don't really care

Lists

A very useful data structure, quite flexible and versatile. Empty list: `L <- list()`. Convenient for things like parameters. For instance

```
L[1]

## $a
## [1] 10

L[[2]]

## [1] 3

L$a

## [1] 10

L[["b"]]

## [1] 3
```

Vectors

```
x = 1:10
y <- c(x, 12)
y
z = c("red", "blue")
z
z = c(z, 1)
z
```

Note that in z, since the first two entries are characters, the added entry is also a character. Contrary to lists, vectors have all entries of the same type

Matrices

Matrix (or vector) of zeros

```
A <- mat.or.vec(nr = 2, nc = 3)
```

Matrix with prescribed entries

```
B <- matrix(c(1,2,3,4), nr = 2, nc = 2)
B
#      [,1] [,2]
# [1,]    1    3
# [2,]    2    4
C <- matrix(c(1,2,3,4), nr = 2, nc = 2, byrow = TRUE)
C
#      [,1] [,2]
# [1,]    1    2
# [2,]    3    4
```

Remark that here and elsewhere, naming the arguments (e.g., `nr = 2`) allows to use arguments in any order

Matrix operations

Probably the biggest annoyance in R compared to other languages

- The notation `A*B` is the *Hadamard product* $A \circ B$ (what would be denoted `A.*B` in matlab), not the standard matrix multiplication
- Matrix multiplication is written `A %*% B`

Vector operations

Vector addition is also frustrating. Say you write `x=1:10`, i.e., make the vector

```
x
#  [1]  1  2  3  4  5  6  7  8  9 10
```

Then `x+1` gives

```
x+1
#  [1]  2  3  4  5  6  7  8  9 10 11
```

i.e., adds 1 to all entries in the vector

Beware of this in particular when addressing sets of indices in lists, vectors or matrices

Flow control

```
# if (condition is true) {
#   list of stuff to do
# }
```

Even if `list of stuff to do` is a single instruction, best to use curly braces

```
# if (condition is true) {
#   list of stuff to do
# } else if (another condition) {
#   ...
# } else {
#   ...
# }
```

For loops

`for` applies to lists or vectors

```
# for (i in 1:10) {
#   something using integer i
# }
# for (j in c(1,3,4)) {
#   something using integer j
# }
# for (n in c("truc", "muche", "chose")) {
#   something using string n
# }
# for (m in list("truc", "muche", "chose", 1, 2)) {
#   something using string n or integer n, depending
# }
```

lapply

Very useful function (a few others in the same spirit: `sapply`, `vapply`, `mapply`)
Applies a function to each entry in a list/vector/matrix. Because there is a parallel version (`parLapply`) that we will see later, worth learning

```
l = list()
for (i in 1:10) {
        l[[i]] = runif(i)
}
lapply(X = l, FUN = mean)
```

or, to make a vector

```
unlist(lapply(X = l, FUN = mean))
```

or

```
sapply(X = l, FUN = mean)
```

“Advanced” lapply

Can “pick up” nontrivial list entries

```
l = list()
for (i in 1:10) {
        l[[i]] = list()
        l[[i]]$a = runif(i)
        l[[i]]$b = runif(2*i)
}
sapply(X = l, FUN = function(x) length(x$b))
```

gives

```
# [1]  2  4  6  8 10 12 14 16 18 20
```

Just recall: the argument to the function you define is a list entry (`l[[1]]`, `l[[2]]`, etc., here)

Avoid parameter variation loops with expand.grid

```
# Suppose we want to vary 3 parameters
variations = list(
    p1 = seq(1, 10, length.out = 10),
    p2 = seq(0, 1, length.out = 10),
    p3 = seq(-1, 1, length.out = 10)
)

# Create the list
tmp = expand.grid(variations)
PARAMS = list()
for (i in 1:dim(tmp)[1]) {
    PARAMS[[i]] = list()
    for (k in 1:length(variations)) {
        PARAMS[[i]][[names(variations)[k]]] = tmp[i, k]
    }
}
```

Can I have this wrapped up to go?

For each slide set including `R` code, we generate an `R` file in the `CODE` directory with all the code chunks in this `Rnw` file and no LaTeX

Source the file (if you are reading `SLIDES/filename.pdf`, then you want `CODE/filename.R`) to reproduce all results in the slide set without generating a pdf

Some small changes might be required (for instance, in the file for this lecture, quite a lot is commented out)