Programming in Finance

Thomas Schmelzer
thomas.schmelzer@gmail.com
University of Liechtenstein, Vaduz

October 3, 2018

Contents

1 Preface

My goal is to show you programming paradise or at least the door to it or the place at the start of the track leading towards the door (or the ... you get recursion now). Let's not be too ambitious given we only have a few hours together.

Obviously there is a short-term goal lurking. You all want to maximize the likelihood of passing the exam. There will be only one exam. Don't forget that exams are rare moments in your existence to shine. You should be very excited.

There's a simple and well tested strategy to maximize this likelihood. You have to attend all the lectures. Attendance is not enough though. You have to participate and engage. You do the exercises, you ask questions and you discuss with your fellow students. You even start writing your own little programs and code the nights away.

Before we meet you may want to install R and RStudio on your private computer. Please download it from CRAN. You can also run R on the computers provided by the University.

This little script is not a replacement for your physical presence in my lectures. It is not self-contained. It shall help you to prepare for the exam and will point you to literature. I certainly can't do a better job than many of the books on R and programming have done.

In particular I would like to draw your attention to Introduction to Statistical Learning. You may want head straight to Chapter 2.3 (page 42) in the book where the authors give an introduction to R. If you are not into books you may prefer Hastie on R. In this video Trevor John Hastie himself (one of the gods of statistics) is sharing his passion for R with us mortals.

I should admit that I have never been a frequent R user and I don't consider myself a programmer. I am not going to teach you the latest R perks. The goal is to develop a broader understanding of the ideas and complexity underlying programming and in particular I will point to many applications in finance. I hope you will integrate programming in your work flow as it is the most powerful tool for analysing data, optimising portfolios or filling your PowerPoint slides with actual content.

Some of the announced content will be slightly modified and adjusted (almost on the fly) but the fundamental goal remains the same. By the end you will be able to create your very own programs in R! We will also dig into your dark days as undergraduates and revisit some of the tedious problems you had to solve. We will have fun with R! Happy Computing!

Für Gott, Fürst und Vaterland

Thomas Schmelzer

2 Functions in R

I guess it could be discussed whether R qualifies for the status of a programming language but it is certainly a feasible tool for many applications in particular in statistics, finance and economics. We have not made a big fuss about R vs. . RStudio is the (Integrated Development Environment) for R. R could happily live without RStudio but not vice versa.

You find far too many details about RStudio here. R can be driven using the R console only. Though the typical modus operandi would be to create a R script (e.g. a *.R file) containing definitions of the functions you are using. This is similar to Matlab's *.m files.

If you have never used RStudio before you may find the YouTube channel of the IPZ most helpful.

2.1 Hello World

When learning a new language it's a good tradition to create a small `Hello World` application. We could open a console a finish this task with

```
> print("Hello␣World")
```

We hit enter and should see

```
> print("Hello␣World")
[1] "Hello␣World"
```

One approach to programming is to think in terms of functions, a large number of functions. Functions calling other functions. Each function is a specialist for a certain task. Here we get away with only one function `hello` that concatenates two strings. R comes with a special command for string concatenation:

```
paste("Hello", "World")
```

We make this into a proper function

```
hello <- function(name)
{
    # note that the return object/value has to be given in () brackets
    return(paste("Hello", name))
}
```

It might be tedious to specify an argument every time we call this function. Hence we can define an *optional* argument.

```
hello <- function(name="World")
{
    return(paste("Hello", name))
}
```

We save this function in *.R script file and *source* the file to make it visible within the Console pane. In the console you then call the function we have just created

```
> hello()
[1] "Hello␣World"
> hello(name="Thomas")
[1] "Hello␣Thomas"
```

For such simple function you could also use the alternative syntax

```
hello <- function(name="World") paste("Hello",name)
```

A complex program could consist of hundreds of functions calling each other.

2.2 Converting units

Creating functions that return concatenated strings is probably not exactly exciting. A very appropriate application for programming is to perform dull conversions. Whenever you rent a car in the US the fuel consumption is cited in miles per gallon, rather than in metric liters per 100 kilometers.

Programming doesn't mean you should now search in the vast set of R functions for a function doing exactly that (and maybe create a dependency on a package containing it). Rather we create our very own function doing it. The conversion is not exactly rocket science but requires some thoughts that are best done on a sheet of paper. Your program is hopefully just an accurate reflection of them. If you mess it up on your paper you won't be able to create a clean function.

Let's maybe start with the hull of the function we intend to write

```
f_mpg <- function(mpg)
{
    # convert miles per gallon into liters per $100$ kilometers.
    # mpg       number of miles per gallon
    # return    liters per $100$ kilometers

    # check that mpg is positive
    stopifnot(mpg > 0)

    return(...)
}
```

Note that we make sure the user enters a positive number for the miles per gallon. Here `f_mpg` is the name of the function and `mpg` is the only argument.

Now you should know that

```
  # a mile is 1.60934 km
  # a gallon is 3.78541 litre
```

We can then say that the kilometers per litre are

$$\frac{mpg \times 1.60934}{3.78541}.$$

The inverse are the litres per kilometer

$$\frac{3.78541}{mpg \times 1.60934}.$$

But since we are interested in the consumption for 100 kilometers we get

$$\frac{100 \times 3.78541}{mpg \times 1.60934} = \frac{235.215}{mpg}$$

Hence we now complete our simple function

```
f_mpg <- function(mpg)
{
    # convert miles per gallon into liters per $100$ kilometers.
    # mpg       number of miles per gallon
    # return liters per $100$ kilometers

    # check that mpg is positive
    stopifnot(mpg > 0)

    # a mile is 1.60934 km
    # a gallon is 3.78541 litre
    return(235.215/mpg)
}
```

A function could have multiple arguments, e.g. think of the computation of the BMI (Body-Mass Index) which depends on height and weight.

It is also correct R to drop the return function and write instead

```
f_mpg <- function(mpg)
{
    ... (as before)
    # a gallon is 3.78541 litre
    235.215/mpg
}
```

By convention R will return the result of the last operation before it left the function. However, I would recommend to keep your code as explicit as possible. Always use `return` even if you save a few the time hitting your keyboard to type `return`. In this context you may find an the angry guide to R most entertaining.

2.3 Creating vectors

Vectors are the most basic container type (or data structure) in R. You may remember them from your days as undergraduate students where a vector was some sort of arrow or an element of a vector space. For R you should forget both ideas. Don't go crazy on mixing different datatypes in the very same vector. Keep it clean and civilized. In R we call such vectors *atomic*. A typical vector of characters could be

```
> a <- c("London","Berlin","Paris")
> a[2]
[1] "Berlin"
```

or a vector of integers

```
> a <- c(1,5,2)
> a[2]
[1] 5
> class(a[2])
[1] "numeric"
```

No! That was a trap. R interprets numbers as floats unless you specify them explicitly as Integers

```
> a <- c(1L,5L,2L)
> a[2]
[1] 5
> class(a[2])
[1] "integer"
```

Obviously there are many applications in which it would be tedious to explicitly list all elements of a vector. There are plenty of auxiliary functions to create them, e.g.

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(1,10,2)
[1] 1 3 5 7 9
```

Note that the function from the previous section also accepts vectors, e.g

```
> f_mpg(25:30)
[1] 9.408600 9.046731 8.711667 8.400536 8.110862 7.840500
```

This feature will be important later when we apply functions on entire tables.

2.4 Approximating definite Integrals

Yet we haven't seen an application that requires the use of a programming language. Everything could have been done (faster) using Excel. A first glimpse on the power of R is offered by Monte Carlo methods to approximate definite integrals:

$$I = \int_a^b f(x)dx$$

Now you may remember those problems in which you had to compute the anti-derivative F of f and then $I = F(b) - F(a)$. Good old times. The sad truth is that you could do this only for the most simple functions. Leaving questions of existence for I aside it is usually impossible to compute an anti-derivative. There are powerful numerical ideas to approximate I but their application requires a much deeper understanding of numerical analysis than we (or you) have.

A good idea would be to evaluate f at thousands (or maybe even millions) of points. Take an average and multiply with the length of the interval we have integrated over, e.g. we approximate f with one big rectangle over the interval $[a, b]$. Later you may see very similar ideas at work when we approximate the price of European options.

Let's be brave for a moment and take the elliptical integral $f(x) = \sqrt{1 - x^2}$ over $[a = 0, b = 1]$. The advantage is that we know the explicit solution is $\pi/4$.

Note that the digits of π are not defined in some magic book listing them. The number π is defined as the area of the unit circle and hence it makes perfectly sense to use an integral for a quarter of the unit circle, hence

$$\pi = 4 \times \int_0^1 \sqrt{1 - x^2}dx$$

We construct 10^6 uniformly distributed random points. Imagine for a second how much pain that would be in Excel. Imagine a sheet with 10^6 rows. That might be painful.

```
integral <- function(f, a=0, b=1, n=1e6)
{
  # make sure b >= a
  stopifnot(a<=b)
  # create n uniformly distributed random numbers in [a,b]
  x <- runif(n=n, min=a, max=b)
  # average height of f * width of the rectangle
  return(mean(f(x))*(b-a))
}
```

Here the integral is a function of not less than 4 arguments of which 3 are optional. The function `integral` even expects a function f as argument!

We can call the function integral with a definition of f in the argument, e.g.

```
> 4*integral(f=function(x) sqrt(1-x^2),n=1e6)
[1] 3.141372
```

The value you will observe will deviate as the results rely on a random number generator. Note that the function `function(x) sqrt(1-x^2)` has no name. It is an anonymous function. It is defined on the fly.

For the first time in this lecture we compute an approximation although there may even exist an exact solution. Exact solutions are often out of reach or it may take ages to compute them.

Think about the integral

$$I = \int_0^1 \sin^2(\tan(\tan(\pi x)))dx.$$

Bornemann and Schmelzer have shown that

$$I = e^{-\tanh 1}\sinh(\tanh(1)) = 0.390992162151530\dots$$

The paper may give you a first impression how hard it would be to solve integrals analytically, e.g. trying to construct an exact solution. It is typically simpler and yet it provides enough accuracy to come up with some sort of approximation or even a sequence of them using a program. We will return to this discussion in the context of linear systems.

We also don't mimic an approach here we did with pen and pencil before. We apply for the first time a method we would fail to run without serious computing power. None of us would generate 10^6 random numbers by hand or throw a dice like a million times. Having R at hand such tasks become trivial and fun.

2.5 Sorting a vector

Given a vector of random numbers

```
> set.seed(seed=0)
> x<-sample(1:10)
> x
 [1]  9  3 10  5  6  2  4  8  7  1
```

the short answer to sort this vector would be to call `sort(x)`. However, it's a worthwhile exercise to understand at least one sorting algorithm. Tony Hoare's Quicksort is my first choice. It's an amazing idea taking full advantage of recursion and is among the greatest algorithms of the previous century despite or maybe because of the brevity[1]. In computer science we would call this algorithm beautiful and elegant.

```
qsort <- function(x)
{
  # what if there is no data or only one point...
  if (length(x) <= 1) return(x)

  # make now three arrays (only the area in the middle is sorted).
  left <- x[x < x[1]]
  middle <- x[x == x[1]]
  right <- x[x > x[1]]
```

[1] I bet you also came across Dantzig's Simplex algorithm in a course on operations research or linear programming.

```
  # qsort works also for an empty vector!

  # build a new vector in the middle the pivot element(s)
  # and on the right the larger elements
  # and on the left the smaller elements
  return(c(qsort(left), middle, qsort(right)))
}
```

Note that we are using *Boolean indexing* here.

```
> x
 [1]  9  3 10  5  6  2  4  8  7  1
> x < 4
 [1] FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
> x[x < 4]
[1] 3 2 1
```

Let's study the fate of the vector given above in the quicksort algorithm. The first pivot element `x[1]` is 9. We get

```
left <- c(3,5,6,2,4,8,7,1)
middle <- c(9)
right <- c(10)
```

Note that a vector may contain multiple copies of the same element. Hence `middle` could be a vector of length larger than 1. Sorting the vector `right` is trivial. The vector `left` is unsorted but shorter than the initial vector. In `qsort(left)` we get

```
left <- c(2,1)
middle <- c(3)
right <- c(5,6,4,8,7)
```

We drill deeper and sort `left` and `right`. We divide and conquer!

2.6 Experimenting with the Collatz conjecture

Any introduction to Programming should discuss the Fibonacci numbers. We don't. Instead we introduce the open Collatz conjecture. You could be instantaneously world-famous by solving it. Here's how it works. We define a function f such that $f(n) = n/2$ if n is even and $f(n) = 3n + 1$ if n is odd. We then define $a_0 = n$ and $a_i = f(a_{i-1})$ where $i \geq 1$

An example may clarify the situation:

$$13 \mapsto 40 \mapsto 20 \mapsto 10 \mapsto 5 \mapsto 16 \mapsto 8 \mapsto 4 \mapsto 2 \mapsto 1.$$

We stop iterating once we have reached 1. The conjecture claims that this iteration will always end at 1 for every $n \in \mathbb{N}$. Experiments have confirmed that this is true for all numbers smaller than 10^{22}.

Let's create this function

```
f <- function(n)
{
    # construct the entire Collatz path starting from n
    if (n==1) return(1)
    if (n %% 2 == 0) return(c(n, f(n/2)))
    return(c(n, f(3*n + 1)))
}
```

Some developers may argue against recursion in such a context. Note in particular that we are building a vector here without allocating space for it before. It's dynamically growing in size. This tends to be ineffective and slow.

Writing functions in the most effective way possible is a topic far beyond those lecture notes. Avoid the temptation of premature optimization in your code. You may sacrifice any elegance on the altar of efficiency.

2.7 Summary

Functions are central building blocks of programming (in R). R comes with numerous functions out-of-the-box but being able to develop functions is crucial for any work going beyond simple console computations.

We have seen functions as arguments to other functions (e.g. in the Monte Carlo integration), a function calling itself(recursion as in QuickSort or in the Collatz problem), multiple and optional arguments and the stopifnot construct to check feasible input parameters.

Functions should be kept short, well documented and tested. It is even possible to define functions within functions. In that case it is not possible to call the functions from outside the hosting function. In professional software development it's a good idea to keep the interface you expose to clients (e.g. other programmers) as small as possible.

2.8 Practical Hints

You may have to write your own function or your own set of functions in your exam or for your thesis. Functions always create some objects (e.g. matrices, vectors (quicksort) or scalars (definite integrals, miles per gallon). A function can not mutate any of its arguments as it deals with copies of them.

Programming involves a lot of creativity and there are often numerous approaches possible. The good news is that you don't have to be too concerned how to design a system of a few hundred functions. We are happy if you manage to create one or two of them.

You have seen already multiple examples and they all share a common syntax. A function may rely on a list of arguments. Some of them might be optional.

Please resist the idea of creating your very own syntax. It's always

```
my_fun <- function(arg_1, arg_2, ...)
{
    # I am a comment. This function creates/computes/...
    # arg_1 is ...
    # arg_2 is ...
    stopifnot(arg_1 ...)
    body
}
```

Having this in mind you need to

- Come up with a descriptive name for your function
- Decide about the arguments you would like to give to your function (always aim for a small number). Make arguments optional if they should have a standard value or tend to remain unchanged.
- Use plenty of `stopifnot` to check whether all arguments are feasible.

- Use comments all over the place.
- If you notice that your function is doing multiple things it's time to split your function into functions.
- It is possible to work without the `return` function. Nevertheless, be explicit and use it. Don't forget the brackets `return()`.

Note that it is possible to access variables defined outside the function (e.g. outside the *scope* of the function). But please be extremely careful. That's a recipe for disaster. Keep it civilized. Treat functions as little self-contained units. Don't you dare to access variables not given to or defined within the function!

2.9 Python?

The concept of a function is essentially present in all major programming languages. Sometimes they are obfuscated as some static members of a class (Java), but the process of identifying suitable sub-problems and solve them by creating a function remains. The exact syntax will differ for each of the languages though. Currently the elephant in the room is Python. We can almost translate the R code for the Collatz problem word for word to Python:

```
import numpy as np

def collatz(n):
    if n == 1:
        return np.array([1.0])

    if n % 2 == 0:
        return np.append([n], collatz(n/2))

    return np.append([n], collatz(3*n + 1))
```

I can hear the screams of Python developers! A more Pythonic way would be:

```
def collatz(n):
    assert isinstance(n, int)
    assert n >= 1

    def __colla(n):

        while n > 1:
            yield n

            if n % 2 == 0:
                n = int(n / 2)
            else:
                n = int(3 * n + 1)

        yield 1

    return list([x for x in __colla(n)])
```

Note that we do not use recursion here and no vector is growing dynamically in size. Unfortunately it is not trivial to convert this code back into R as the `yield` construct is not present in R[2]

[2]The author would be delighted if a R expert could correct him.

3 Linear Algebra with R

Linear Algebra is the branch of mathematics underlying almost all quantitative topics in finance. It is also the mathematical home of the author and therefore he may have a bias here. It might be good to review some of the basics before you proceed. We have already seen vectors and introduced them as 1-dimensional container objects or data structures. Closely related to vectors are matrices. We introduce them as 2-dimensional container objects.

My toe nails roll up when I introduce matrices as some sort of rectangular car park with multiple lots. Mathematicians have a much tighter and more solid concept of matrices, which are instances of a linear mapping in a particular basis. Let's not go there (today).

3.1 Matrices

R is somewhat special in the way we construct matrices. We have several options though. We can wrap up a vector, e.g.

```
A <- matrix(rnorm(100), nrow=20, ncol=5)
```

Here we construct a matrix with 20 rows and 5 columns out of a vector of length 100. We call matrices with more rows and columns *high and skinny*. In most applications (in finance) we deal with such matrices.

Constructing and working with matrices in R is discussed in great detail here.

You may interpret a vector as matrix with only one column. Also keep in mind that a $n \times m$ Matrix has n rows and m columns (Zeilen zuerst, Spalten später).

3.2 The Inner Product of vectors

The inner product of two vectors x and y (both of the same length n) is defined as

$$x^T y = \sum_{i=1}^{n} x_i y_i$$

The inner product induces the Euclidean norm via

$$\|x\|_2 = \sqrt{x^T x}$$

We call two vectors x and y orthogonal if $x^T y = 0$. The inner product therefore induces a geometry among the vectors.

In R we would write

```
x <- c(1,0,1)
y <- c(2,1,0)

# for the inner product
t(x) %*% y

# or
sum(x*x)

norm_2 <- function(x) sqrt(sum(x*x))

# Euclidean norm of the vector x
```

```
norm_2(x)

# Please avoid using the crossprod function! Too ambiguous
```

The inner product also defines the angle between two vectors:

$$x^T y = \|x\|_2 \|y\|_2 \cos(x, y)$$

The inner product is also known as vector scalar product or dot product. In finance it is very relevant as the expected return of a portfolio is the inner product of a weight vector and the vector of expected returns per asset.

More details on the the inner product

3.3 The Matrix-Vector product

Given a $n \times m$ matrix we could interpret the matrix as m column vectors or n row vectors. Or shall we say it's just a grid of numbers? For applications is most helpful to interpret a matrix as m column vectors, e.g

$$A = \begin{pmatrix} a_{,1} & \dots & a_{,m} \end{pmatrix}$$

where $a_{,i}$ is a column vector of length n. Given now a vector x of length m the Matrix-Vector product

$$Ax = x_1 a_{,1} + x_2 a_{,2} + \dots + x_m a_{,m}$$

is the weighted sum of the m column vectors. The length of the vector x has to match the number of columns in A. In R we write

```
A %*% x
```

instead of

```
A*x      # DON'T DO THIS! UNLESS x IS A SCALAR. VERBOTEN!
```

Creating your own function for matrix vector products is a simple exercise also introducing the concept of a `for` loop.

```
mv <- function(A,v)
{
  # check that A has as many columns as v has entries
  stopifnot(ncol(A)==length(v))

  x = 0

  # loop over all columns of A
  for (i in 1:length(v))
  {
    # increment the "matrix" x
    x = x + A[,i]*v[i]
  }

  return(x)
}
```

The *vector space* of all vectors we can reach with A is called the *image* or *range* of A. Some also call it the span of the column vectors.

Keep in mind the matrix

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

If we now look at

$$A \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_1 + w_2 \\ w_1 + w_2 \end{pmatrix}$$

you see that only points (x, y) with $x == y$ are in the image of A.

In some rare situations you may want to interpret a Matrix as n row vectors. The jth element of matrix-vector product

$$Ax = \begin{pmatrix} a_{1,}^T x \\ \vdots \\ a_{n,}^T x \end{pmatrix}$$

is just the inner product of the jth row with the vector x. With that we can define an elegant Matrix-Matrix product

$$(AB)_{i,j} = a_{i,}^T b_{,j}.$$

In R this would be

```
A %*% B
```

3.4 Solving linear systems

We have seen it is trivial to compute $b = Ax$ when both A and x are given. It's just a simple matrix-vector product and b will be in the image of A by construction. In practice we ask for the solution of the inverse problem. Given A and b what was x?

Of course, this may not have a solution at all as b may not be in the image of A. We have found another reason to be happy with approximations. Often there is no solution to our problem.

You may remember the days when you had to solve 3×3 or 2×2 systems by hand. What a tedious task. Tedious stuff is always best outsourced to a computer.

We rarely solve linear systems as either there is no solution or even if there is one it's faster to compute a series of approximations. If you are really keen to solve a linear systems you can use R's solve command:

```
> A<-matrix(c(1,1,1,2,2,0,0,2,2),3,3)
> A
     [,1] [,2] [,3]
[1,]    1    2    0
[2,]    1    2    2
[3,]    1    0    2
> b<-1:3
> x<-solve(A,b)
> A %*% x-b      # if x is the solution this will return a vector of 0
     [,1]
[1,]    0
[2,]    0
[3,]    0
```

If A doesn't have full rank (e.g. there are points that can not be reached by a matrix-vector product) the computer says no.

3.5 Least squares

Here we assume that the matrix $A \in \mathbb{R}^{n \times m}$ has full column rank. This implies in particular that $A^T A \in \mathbb{R}^{m \times m}$ is invertible.

The most common situation we face in finance or economics would be a high and skinny system. There is no hope that b is in the image of A.

Our goal is therefore to come up with an approximation keeping the residual as short as possible, e.g.

$$x^* = \arg \min_{x \in \mathbb{R}^m} \|Ax - b\|_2$$

For a derivation of the Normal Equations or here:

This can only happen if $Ax^* - b$ is orthogonal to every vector in the column space of A, i.e. every vector of the form Ay for some m-dimensional vector y. Thus:

$$(Ay)^T (Ax^* - b) = 0$$

and hence

$$y^T A^T (Ax - b) = 0.$$

This has to be true for every vector y and hence $A^T(Ax^* - b) = 0$ which are the normal equations. Therefore

$$A^T A x^* = A^T b$$

or

$$x^* = (A^T A)^{-1} A^T b$$

Here the matrix $(A^T A)^{-1} A^T$ is the *Moore-Penrose Inverse*. In R:

```
solve(t(A) %*% A, t(A) %*% b)
```

Now where did this orthogonality come from? Given an x satisfying the normal equations, take any other y. Then the Pythagorean theorem gives $\|Ay-b\|^2 = \|A(x-y)\|^2 + \|Ax-b\|^2 \geq \|Ax-b\|^2$. Moreover $\|A(x-y)\|^2 > 0$ so that $\|Ay - b\|^2 > \|Ax - b\|^2$.

3.6 Constrained Least squares and Regularisation

You have seen only the tip of an iceberg in the lecture. Here I point you into further directions for your private research.

In applications we would typically have additional constraints on the vector x^*. Imagine we build a minimum variance portfolio or an index tracker. We want to impose $x^* \geq 0$ (element-wise) or $\sum x_i^* = 1$. Such problems are a lot harder than your typical least-squares problem.

The modern and most flexible way would be to formulate them as a convex program. You may want to experiment with new `CVXR` package.

In many applications it would be rather brave to solve the unconstrained problem, e.g to solve

$$x^* = \arg \min_{x \in \mathbb{R}^m} \|Ax - b\|_2$$

In particular, if A has almost co-linear columns the solution would look extreme (e.g. extremely large entries). To stabilize the problem we could introduce penalty terms, such as

$$x^* = \arg \min_{x \in \mathbb{R}^m} \|Ax - b\|_2 + \lambda \|x\|_2.$$

This approach known as Ridge Regression balances the solution. The free parameter λ acts as a gas pedal. For $\lambda = 0$ you are back to your unconstrained problem. For very large λ you pull the solution towards 0.

You may want to read about

$$x^* = \arg\min_{x \in \mathbb{R}^m} \|Ax - b\|_2 + \lambda \|x\|_1.$$

This approach is known as LASSO or sparse regression. This idea is most suitable for factor selection as the active columns of A are chosen one-by-one for shrinking λ. There are very efficient algorithms for this problem, e.g. LARS (Least Angle Regression).

You can also apply LASSO and Ridge Regression together in the same problem. This is known as Elastic Net.

3.7 Linear Regression

Linear Regression is just a very special case of solving a least squares problem. If you have a bunch of pairs (x_i, y_i) you could setup the matrices

$$A = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \text{ and } b = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

In R there are tons of functions for this and you may want to explore the `lm` and `glm` commands.

```
# the matrix for A (no need to create the constant column...)
A <- c(1, 2, 3, 4)
b <- c(1.1, 1.9, 3.05, 4.1)
x<-lsfit(A,b,intercept=TRUE)
x$coefficients

# Using lm for the same task
x=lm(b~A)
summary(x)

glm(b~A)
```

You may also want to study Lab 3.

3.8 Summary

We have finally seen matrices and discussed in particular the matrix-vector product in great detail. We interpreted matrices as a bunch of column vectors and the matrix-vector product as a weighted sum of those column vectors. Solving linear systems is the inverse problem though. Given a matrix and given a vector b, can we find a vector x such that $Ax = b$. Almost always the answer is that there is no such vector. In applications of interest in finance the matrix A is typically high and skinny. Once more we have to live with approximations. An approximation that keeps the residual $r = Ax - b$ measured in the Euclidean norm as short as possible.

This finally opened the door to least-squares problems and we met the celebrated Normal Equations (normal as in orthogonal).

Unfortunately it turns out that solving unconstrained least squares problems is a somewhat bad and inflexible idea. It's hard to impose constraints on x and the solutions may exhibit a level of sheer craziness. We have addressed these points here in the script and you may want to return to this discussion later in life when you solve regression problems in your thesis.

Once regression has been embedded into the language of convex programming we have a powerful tool for portfolio optimisation at hand.

4 Statistics and Portfolios in R

Portfolios are essentially weighted sums of random variables. Of course, we hope that we would have some edge and that our allocation has a least some sort of positive expectation. Nevertheless, without a grasp of some basic statistics we won't see land in portfolio management[3]

4.1 Elementary statistics

This chapter revisits results you should be familiar with from your days as undergraduates but also embeds statistics into linear algebra as we all love linear algebra.

The mean of a vector $x = \begin{pmatrix} x1 & \dots & x_n \end{pmatrix}$ is defined as

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

The mean is not very robust to outliers and in particular in skewed distribution is often a poor measure for the centre of distribution.

The variance of the very same vector is

$$\mathrm{Var}(x) = \frac{1}{n-1}\sum_{i=1}^{n-1}(x_i - \bar{x})^2$$

For any constant c we get $\mathrm{Var}(x) = \mathrm{Var}(x+c)$. Hence without loss of generality we could centre the vector x, e.g. we define

$$y_i = x_i - \bar{x}$$

Hence

$$\bar{y} = 0$$

and

$$\mathrm{Var}(x) = \mathrm{Var}(y) = \frac{1}{n-1}\sum_{i=1}^{n-1} y_i^2 = \frac{1}{n-1}\|y\|_2^2.$$

The variance is therefore the scaled and squared Euclidean norm of a centered vector. The standard deviation is therefore

$$\mathrm{Sd}(x) = \sqrt{\mathrm{Var}(x)} = \frac{1}{\sqrt{n-1}}\|y\|_2.$$

The covariance of two vectors a and b is defined as

$$\mathrm{Cov}(a,b) = \frac{1}{n-1}\sum_{i=1}^{n}(a_i - \bar{a})(b_i - \bar{b})$$

Again, only the deviation from the mean enters the equation and hence we can w.l.o.g. (without loss of generality) assume that $\bar{a} = \bar{b} = 0$.

$$\mathrm{Cov}(a,b) = \frac{1}{n-1}\sum_{i=1}^{n} a_i b_i = \frac{a^T b}{n-1}$$

So the covariance is the inner product of the centered vectors scaled by $n-1$. The inner product has a geometric definition

$$a^T b = \|a\|_2 \|b\|_2 \cos(a,b).$$

[3]Unless you prefer the wet finger in the air method.

But we have seen before

$$\|a\|_2 = \mathrm{Sd}(a)\sqrt{n-1}$$

and hence

$$\mathrm{Cov}(a,b) = \mathrm{Sd}(a)\mathrm{Sd}(b)\cos(a,b).$$

A trivial consequence for the correlation is that

$$\mathrm{Cor}(a,b) = \cos(a,b) = \frac{a^T b}{\|a\|_2\|b\|_2}$$

where a and b are two centered vectors.

Let's have a look at the matrix $A^T A$ where all columns in A have been centered. As we have seen before the entry $(A^T A)_{i,j}$ is the inner product of the ith column of A with the j the column of A. However the inner product of two centered vectors is

$$a_{,i}^T a_{,j} = (n-1)\mathrm{Cov}(a_{,i}, a_{,j})$$

and hence

$$A^T A = (n-1)\mathrm{Cov}(A).$$

We have met this matrix also in the Moore-Penrose Inverse.

```
x <- rnorm(100)

# mean of x
mean(x)
# or
sum(x)/length(x)

# variance
var(x)
# or
sum((x - mean(x))^2)/(length(x)-1)
# or
y <- x - mean(x)
sum(y*y)/length(y)-1

# standard deviation
sd(x)
or
sqrt(var(x))

# covariance
cov(x,y)
or
a <- x - mean(x)
b <- y - mean(y)
t(a) %*% b / (length(a) - 1)

# correlation
cor(x,y)
or
a <- x - mean(x)
b <- y - mean(y)
norma = sqrt(sum(a*a))
```

```
normb = sqrt(sum(b*b))
t(a) %*% b / (norma * normb)

# covariance matrix
a <- matrix(rnorm(10),5,2)
cov(a)
or
a <- scale(a, center=TRUE, scale=FALSE)
t(a) %*% a / (nrow(a) - 1)
```

4.2 The sum of random variables

Let's assume a and b are two random vectors.

$$\mathrm{Var}(a+b) = \frac{1}{n-1}\sum_{i=1}^{n}\left(a_i + b_i - \overline{a+b}\right)^2$$

Note that $\overline{a+b} = \bar{a} + \bar{b}$ and hence

$$\begin{aligned}\mathrm{Var}(a+b) &= \frac{1}{n-1}\sum_{i=1}^{n}\left(a_i - \bar{a} + b_i - \bar{b}\right)^2\\ &= \frac{1}{n-1}\sum_{i=1}^{n}(a_i-\bar{a})^2 + \frac{1}{n-1}\sum_{i=1}^{n}\left(b_i-\bar{b}\right)^2 + \frac{2}{n-1}\sum_{i=1}^{n}(b_i-\bar{b})(a_i-\bar{a})\\ &= \mathrm{Var}(a) + \mathrm{Var}(b) + 2\times\mathrm{Cov}(a,b)\end{aligned}$$

In particular if a and b are uncorrelated we get

$$\mathrm{Var}(a+b) = \mathrm{Var}(a) + \mathrm{Var}(b)$$

And hence

$$\mathrm{Sd}(a+b) = \sqrt{\mathrm{Var}(a) + \mathrm{Var}(b)}$$

if a and b are uncorrelated.

4.3 Portfolios

Note that a portfolio is at the end of the day the (weighted) sum of random variables. We only discussed two assets in lecture as it simplifies the use of geometric arguments and visualisations as demonstrated in a video by Man AHL.

We assume we own assets $S_1,\ldots,S_n$. Their return history induces a matrix

$$A = \begin{pmatrix} r_1 & \ldots & r_n\end{pmatrix}$$

where r_i is the time series of returns for asset S_i. Closely related is the centered version of A

$$A_s = \begin{pmatrix} r_1 - \overline{r_1} & \ldots & r_n - \overline{r_n}\end{pmatrix}$$

The covariance matrix can be computed either with

```
# covariance matrix
C <- cov(A)
# or more explicitly
```

```
A_s = scale(A, center=TRUE, scale=FALSE)
C <- t(A_s) %*% A_s / (nrow(A) - 1)

# rescale to get to the correlation matrix
cor <- cov2cor(C)

# back to covariance matrix with
# vector of volatilities (per column)
v = apply(A,2,sd)
C <- diag(v) %*% cor %*% diag(v)
```

The volatility of the portfolio is

$$\begin{aligned}\mathrm{Var}\left(\sum w_i S_i\right) &= w^T C w\\ &= \frac{1}{n-1} w^T A_s^T A_s w\\ &= \frac{1}{n-1}\|A_s w\|_2^2\end{aligned}$$

This identity is extremely helpful in asset management. We no longer need to compute the covariance matrix explicitly. Note that $A_s w$ is a time series of weighted excess returns (relative to the mean).

If we try to construct a minimum variance portfolio we could drop the constant factor $\frac{1}{n-1}$ and the square and state

$$\begin{aligned}&\min\|A_s w\|_2\\ &s.t. \sum w = 1\\ &w \geq 0\end{aligned}$$

which is a constrained least-squares problem. If we find the solution is too concentrated we could apply Ridge regression, e.g.

$$\begin{aligned}&\min\|A_s w\|_2 + \lambda\|w\|_2\\ &s.t. \sum w = 1\\ &w \geq 0\end{aligned}$$

Note that for $\lambda \to \infty$ the solution will approach the $1/n$ portfolio, e.g. $w_i = \frac{1}{n}$.

In practice we rarely invest the efforts to centre A. We tend to get away with Aw rather than $A_s w$.

In a world with only two assets the covariance matrix is

$$C = \begin{pmatrix} v_1^2 & \rho v_1 v_2 \\ \rho v_1 v_2 & v_2^2 \end{pmatrix}$$

where v_i is the volatility of the ith asset and hence the variance for the portfolio is:

$$w^T C w = w_1^2 v_1^2 + w_2^2 v_2^2 + 2 w_1 w_2 \rho v_1 v_2.$$

4.4 Summary

Here we first linked linear algebra (and therefore geometry) with statistics. By now we all understand that the correlation of two vectors is the cosine of the angle between them. In particular a correlation ρ is always within $-1 \leq \rho \leq +1$. We then spent some time on the sum of random variables as I have noticed a large fraction of you had problems with it. We can argue that a portfolio is a weighted sum of random variables. We then reformulated simple portfolio allocation problems as constrained least squares problems.

5 Tidyverse and Tidyquant in R

Tidyverse and Tidyquant won't be covered in the exam but might be very helpful in your course on data sourcing.

5.1 Tidyverse

R is an old language and over the years it grew in all sort of directions. Today there are more than 13000 packages available on CRAN. R has never been embraced by the community of computer scientists and accumulate quite some rust over the year. Comparing R with most modern languages you will find R often poorly designed and rather limited.

The situation has changed dramatically when RStudio (the company) came along. RStudio dramatically changed the landscape by introducing RStudio (the IDE), Shiny and several packages.

RStudio's research is led by Hadley Wickham — the R messiah. He gave the world tidyverse. Tidyverse is a set of packages that share a common philosophy. Having said that you could still work without tidyverse but your code will probably be slower, less elegant and less readable. A complete example comparing base R and tidyverse is given here.

5.2 Tidyquant

Tidyquant is a rather new package to access financial data from the web using a common api. A good discussion is given here.

5.3 Summary

Tidyverse and Tidyquant could be very helpful for solving the problems in your data sourcing class.

6 What's next?

You have made it through a first introduction to Programming in R. There are many possible routes for you to continue the process of exploring R.

6.1 Shiny

I have to admit that I was almost blown away when I deployed my first own app on shinyapps.io A simple introduction is given here. Apps can be a powerful tool to communicate your results with your audience.

6.2 Dependency Hell

We have touched the issue in the class room. We had at least 4 different versions of R running live. The way forward is to rely on docker containers to make sure every user is running the experiments in exactly the same environment. There is the Rocker project.

6.3 Convex Programming

I have pointed to the issue before. Convex Programming is the mathematical home for portfolio optimisation and I highly recommend to learn the CVXR package.

6.4 Graphics

I would certainly help your professional or academic career if you are able to create elegant and powerful graphics. Hence invest some time and learn about `ggplot2` or `ggvis`.

6.5 Community

You are now officially a programmer. Congratulations. Share your results with the world. Get your own free account on Github. Study what others have published on Github.

6.6 Collatz reloaded

Please remember to mention my name when you finally manage to prove the Collatz conjecture. Gauss got famous by predicting the trajectory of Ceres. It's your turn now.

Welcome to the world of Programming!

Für Gott, Fürst und Vaterland