MATH 2740 – 03
Matrix methods

Julien Arino
University of Manitoba
julien.arino@umanitoba.ca

Fall 2024

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

- Least squares problems
- QR factorisation
- Singular values decomposition (SVD)
- Principal component analysis (PCA)
- Support vector machines

Least squares problems

QR factorisation

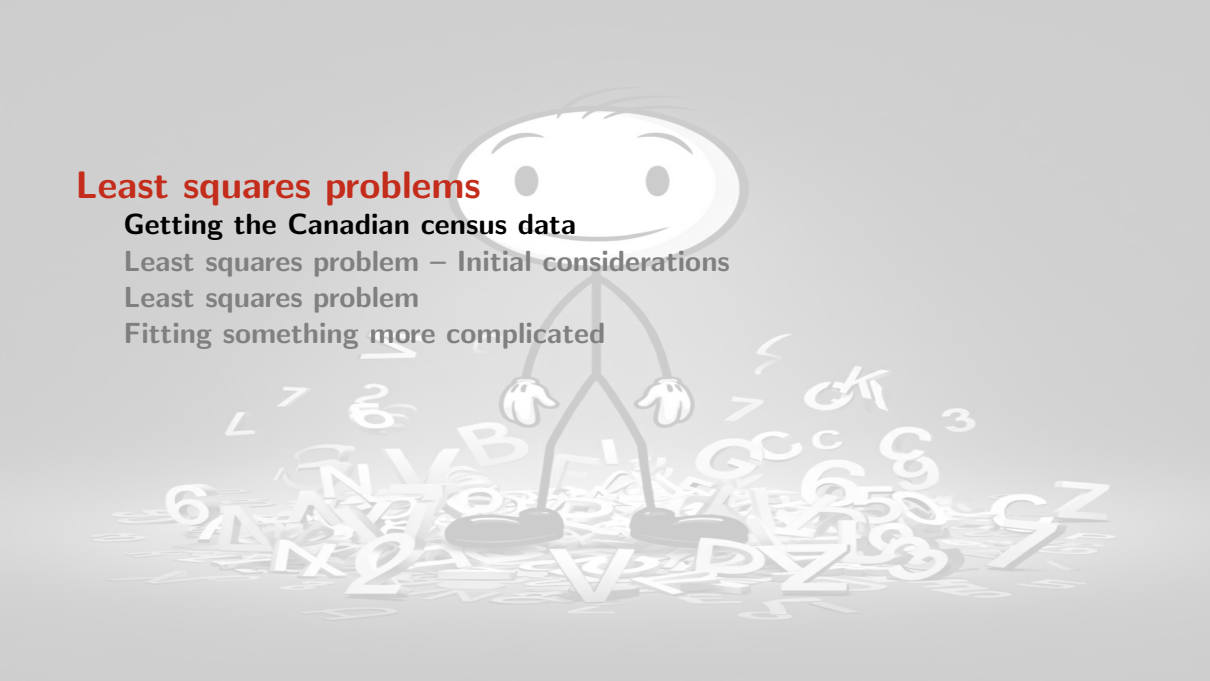
Singular values decomposition (SVD)

Principal component analysis (PCA)

Support vector machines

Least squares problems

- **Getting the Canadian census data**
- Least squares problem – Initial considerations
- Least squares problem
- Fitting something more complicated



Grabing the Canadian census data

We want to consider the evolution of the population of Canada through time

For this, we grab the Canadian census data

Search for (Google) "Canada historical census data csv", since csv (comma separated values) is a very easy format to use with R

Here, we find a csv for 1851 to 1976

We follow the link to Table A2-14, where we find another link, this time to a csv file. This is what we use in R

Grabing the Canadian census data

The function `read.csv` reads in a file (potentially directly from the web)
Assign the result to the variable data. We then use the function head to show the first few lines in the result.

```
data_old = read.csv("https://www150.statcan.gc.ca/n1/en/pub/11-516-x/sectio
head(data_old)

##    X Series.A2.14.
## 1 NA
## 2 NA          Year
## 3 NA
## 4 NA
## 5 NA
## 6 NA
##   Population.of.Canada..by.province..census.dates..1851.to.1976 X.1
## 1                                                                NA
## 2                                                        Canada  NA New
## 3                                                                NA
```

Obviously, this does not make a lot of sense. This is normal: take a look at the first few lines in the file. They take the form

```
head(data_old)
```

```
##    X Series.A2.14.
## 1 NA
## 2 NA          Year
## 3 NA
## 4 NA
## 5 NA
## 6 NA
##   Population.of.Canada..by.province..census.dates..1851.to.1976 X.1
## 1                                                                NA
## 2                                                        Canada  NA New
## 3                                                                NA
## 4                                                                NA
## 5                                                             2  NA
## 6                                                                NA
```

The first line here does this; it is easy to deal with this: the function read.csv takes the optional argument skip=, which indicates how many lines to skip at the beginning The second line is also empty, so let us skip it too

```
data_old = read.csv("https://www150.statcan.gc.ca/n1/en/pub/11-516-x/sectio
                    skip = 2)
head(data_old)

##    X Year     Canada X.1 Newfound.  Prince X.2    Nova       New    Quel
## 1 NA                NA      land  Edward  NA  Scotia Brunswick
## 2 NA                NA            Island  NA
## 3 NA               2  NA         3       4  NA       5         6
## 4 NA                NA                    NA
## 5 NA 1976 22,992,604  NA   557,725 118,229  NA 828,571   677,250 6,234,4
## 6 NA                NA                    NA
##     Ontario  Manitoba X.3 Saskat. X.4   Alberta X.5   British X.6     Yu
## 1                    NA  chewan  NA           NA  Columbia  NA Terri
## 2                    NA          NA           NA            NA
## 3         8          9  NA      10  NA        11  NA        12  NA
```

Here, there is the further issue that to make things legible, the table authors used 3 rows (from 2 to 4) to encode for long names (e.g., Prince Edward Island is written over 3 rows). Note, however, that 'read.csv' has rightly picked up on the first row being the column names.
(You could also use the function 'read_csv' from the package 'readr' to read in the file. This function is a bit more flexible than 'read.csv' and can handle such cases more easily. However, it is not part of the base R package, so you would need to install it first.)
Because we are only interested in the total population of the country and the year, let us simply get rid of the first 4 rows and of all columns except the second (Year) and third (Canada)

```
data_old = data_old[5:dim(data_old)[1], 2:3]
head(data_old, n=4)

##   Year     Canada
## 5 1976 22,992,604
## 6
## 7 1971 21,568,311
```

Still not perfect:
- there are some empty rows; - the last few rows need to be removed too, they contain remarks about the data; - the population counts contain commas; - it would be better if years were increasing.
Let us fix these issues.
For 1 and 2, this is easy: remark that the Canada column is empty for both issues.
Now remark as well that below Canada (and Year, for that matter), it is written `<chr>`. This means that entries in the column are characters. Looking for empty content therefore means looking for empty character chains.
So to fix 1 and 2, we keep the rows where Canada does not equal the empty chain.
To get rid of commas, we just need to substitute an empty chain for ",".
To sort, we find the order for the years and apply it to the entire table.
Finally, as remarked above, for now, both the year and the population are considered as character chains. This means that in order to plot anything, we will have to indicate that these are numbers, not characters.

```r
data_old = data_old[which(data_old$Canada != ""),]
data_old$Canada = gsub(",", "", data_old$Canada)
order_data = order(data_old$Year)
data_old = data_old[order_data,]
data_old$Year = as.numeric(data_old$Year)
data_old$Canada = as.numeric(data_old$Canada)
data_old
```

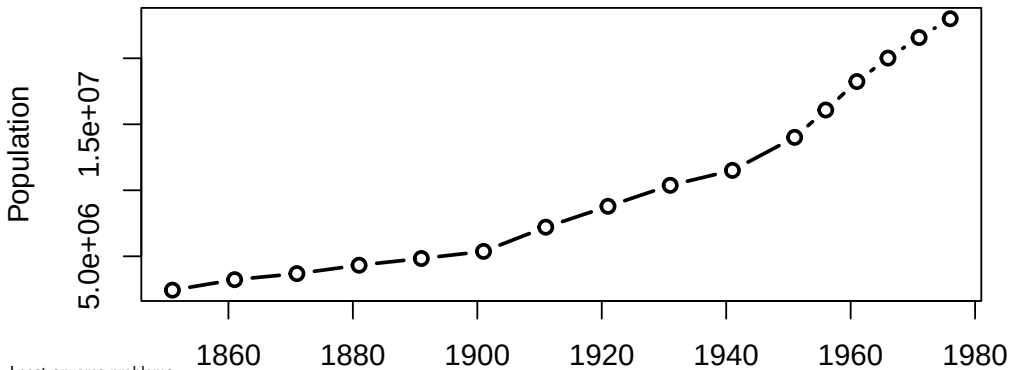
```
##    Year   Canada
## 23 1851  2436297
## 22 1861  3229633
## 21 1871  3689257
## 20 1881  4324810
## 19 1891  4833239
## 17 1901  5371315
## 16 1911  7206643
## 15 1921  8787949
## 14 1931 10376786
```

Row numbers are a little weird, so let us fix this.

```
row.names(data_old) = 1:dim(data_old)[1]
data_old
```

```
##    Year   Canada
## 1  1851  2436297
## 2  1861  3229633
## 3  1871  3689257
## 4  1881  4324810
## 5  1891  4833239
## 6  1901  5371315
## 7  1911  7206643
## 8  1921  8787949
## 9  1931 10376786
## 10 1941 11506655
## 11 1951 14009429
## 12 1956 16080791
## 13 1961 18238247
```

```r
plot(data_old$Year, data_old$Canada,
    type = "b", lwd = 2,
    xlab = "Year", ylab = "Population")
```



But wait, this is only to 1976..! Looking around, we find another table here. There's a download csv link in there, let us see where this leads us. The table is 720KB, so surely there must be more to this than just the population. To get a sense of that, we dump the whole data.frame, not just its head.

```
data_new = read.csv("https://www12.statcan.gc.ca/census-recensement/2011/dp
head(data_new, 10)

##    GEOGRAPHY.NAME            CHARACTERISTIC YEAR.S. TOTAL FLAG_TOTAL
## 1          Canada Population (in thousands)    1956 16081
## 2          Canada Population (in thousands)    1961 18238
## 3          Canada Population (in thousands)    1966 20015
## 4          Canada Population (in thousands)    1971 21568
## 5          Canada Population (in thousands)    1976 22993
## 6          Canada Population (in thousands)    1981 24343
## 7          Canada Population (in thousands)    1986 25309
## 8          Canada Population (in thousands)    1991 27297
## 9          Canada Population (in thousands)    1996 28847
## 10         Canada Population (in thousands)    2001 30007
```

Haha, this looks quite nice but has way more information than we need: we just want the population of Canada and here we get 9960 rows. Also, the population of Canada is expressed in thousands, so once we selected what we want, we will need to multiply by 1,000.
There are many ways to select rows. Let us proceed as follows: we want the rows where the geography is "Canada" and the characteristic is "Population (in thousands)". Let us find those indices of rows that satisfy the first criterion, those that satisfy the second; if we then intersect these two sets of indices, we will have selected the rows we want.

```
idx_CAN = which(data_new$GEOGRAPHY.NAME == "Canada")
idx_char = which(data_new$CHARACTERISTIC == "Population (in thousands)")
idx_keep = intersect(idx_CAN, idx_char)
head(idx_keep, n = 8)

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

Yes, this looks okay, so let us keep only these

```
data_new = data_new[idx_keep,]
head(data_new, n = 8)
```

```
##   GEOGRAPHY.NAME            CHARACTERISTIC YEAR.S. TOTAL FLAG_TOTAL
## 1         Canada Population (in thousands)    1956 16081
## 2         Canada Population (in thousands)    1961 18238
## 3         Canada Population (in thousands)    1966 20015
## 4         Canada Population (in thousands)    1971 21568
## 5         Canada Population (in thousands)    1976 22993
## 6         Canada Population (in thousands)    1981 24343
## 7         Canada Population (in thousands)    1986 25309
## 8         Canada Population (in thousands)    1991 27297
```

We want to concatenate this data.frame with the one from earlier

To do this, we need the two data frames to have the same number of columns and, actually, the same column names and entry types (notice that `YEAR.S.` in `data_new` is a column of characters)

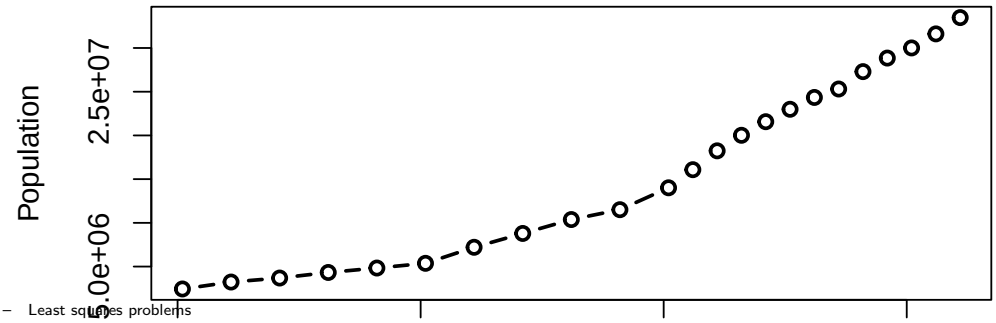
What remains to do

- Rename the columns in the pruned old data (data_pruned) to `year` and `population`. Personally, I prefer lowercase column names.. and `population` is more informative than `Canada`
- Keep only the relevant columns in `data_new`, rename them accordingly and multiply population by 1,000 there
- Transform year in `data_new` to numbers
- We already have data up to and including 1976 in `data_old`, so get rid of that in `data_new`
- Append the rows of `data_new` to those of `data_pruned`

```r
colnames(data_old) = c("year", "population")
data_new = data_new[,c("YEAR.S.","TOTAL")]
colnames(data_new) = c("year", "population")
data_new$year = as.numeric(data_new$year)
data_new = data_new[which(data_new$year>1976),]
data_new$population = data_new$population*1000

data = rbind(data_old,data_new)
```

Let us plot the result

```
plot(data$year, data$population,
    type = "b", lwd = 2,
    xlab = "Year", ylab = "Population")
```



Save the processed data

In case we need the data elsewhere, we save the data to a `csv` file

```
write.csv(data, file = "../CODE/Canada_census.csv")
```

Using `readr` saves the data without row numbers (by default), so we can do this instead

```
readr::write_csv(data, file = "../CODE/Canada_census.csv")
```

Least squares problems

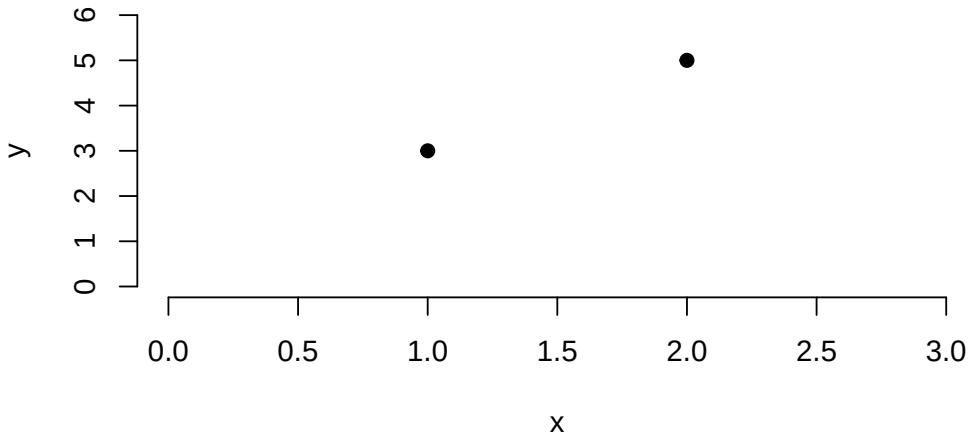
We just collected the census data for Canada

Suppose we want to predict the population of Canada in 20 years given the historical population growth seen in the previous plot. What can we do?

If there were just two points, we could easily "drive" a line through these two points. However, we have much more than two points, so we will use *fitting*, *i.e.*, try to make a curve come as close to possible to the points

We start with a line, giving rise to **linear least squares**

Least squares approximation – A trivial case



We want to find the equation of a line $y = a + bx$ that goes through these two points, i.e., we seek a and b such that

$$3 = a + b$$
$$5 = a + 2b$$

i.e., they satisfy $y = a + bx$ for $(x, y) = (1, 3)$ and $(x, y) = (2, 5)$

This is a linear system with 2 equations and 2 unknowns a and b

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

We know from the "famous" linear algebra in a nutshell theorem that this system has a unique solution if the matrix

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$$

is invertible

$\det(M) = 1$, so we are good, we'll find a and b easily..

Now let's add another point

```
points = list()
points$x = c(1,2,3)
points$y = c(3,5,4) # So the points are (1,3), (2,5) and (3,4)
plot(points$x, points$y,
     pch = 19, cex = 2, bty = "n",
    xlim = c(0, 3.5), ylim = c(0,6), xlab = "x", ylab = "y")
```

These points are clearly not colinear, so there is not one line going through the 3

We end up with an *overdetermined* system

$$
\begin{aligned}
3 &= a + b \\
5 &= a + 2b \\
4 &= a + 3b
\end{aligned}
$$

i.e.,

$$
\begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \\ 4 \end{pmatrix}
$$

We have verified visually that the points are not colinear, so this system has no solution.
(If you had to do it for good, you consider two vectors stemming from these 3 points and compute the angle between them or check that one is a multiple of the other).
So let us instead try to find the line that comes "closest" to the 3 points.

```
A = matrix(c(1,1,1,2), nr = 2, nc = 2, byrow = TRUE)
rhs = matrix(c(3,5), nr = 2, nc =1)
coefs = solve(A,rhs) # To invert A, in R, you use solve(A), to solve Ax=b,
plot(points$x, points$y,
     pch = 19, cex = 2, bty = "n",
    xlim = c(0, 3.5), ylim = c(0,6), xlab = "x", ylab = "y")
abline(coef = coefs, lwd = 2)
```

Obviously, not sensational..

```
plot(points$x, points$y,
     pch = 19, cex = 2, bty = "n",
    xlim = c(0, 3.5), ylim = c(0,6), xlab = "x", ylab = "y")
abline(coef = coefs, lwd = 2)
abline(a = 3, b = 0.5, lwd = 2, col = "red")
```

How do we find "how far away"?
- We could use projections onto the line (which we know minimises the distance) - However, this will be a problem if we later decide that rather than a straight line, we want to use something more "funky" like a quadratic or an exponential

So instead, we compare, for a given value x, the distance between the true value y and the value of y obtained using the curve (line, here) that we use to fit the data
Let (x_i, y_i) be the data points, i.e., here, $(x_1, y_1) = (1, 3)$, $(x_2, y_2) = (2, 5)$ and $(x_3, y_3) = (3, 4)$
Now suppose we use a line with equation $y = a + bx$ and that we pick a value for a and b. Then at x_1,

$$\tilde{y}_1 = a + bx_1 = a + b$$

at x_2

$$\tilde{y}_2 = a + bx_2 = a + 2b$$

and at x_3,

$$\tilde{y}_3 = a + bx_3 = a + 3b$$

Consider x_1, for instance. The error we made by using the line with coefficients (a, b) is $\overrightarrow{(x_1, y_1)(x_1, \tilde{y}_1)}$.

For future use, let us create a function for $y = a_0 + a_1 x$.

```
my_line = function(x, a_0, a_1){
    return(a_0 + a_1*x)
}
```

Functions are super useful when programming

```
my_line(1,2,3)
```

```
## [1] 5
```

```
my_line(a_0 = 2, a_1 = 3, x = 1)
```

```
## [1] 5
```

```
my_line(x = c(1,2,3), a_0 = 2, a_1 = 3)
```

```
## [1]  5  8 11
```

```r
a = 3
b = 0.5 # The line has equation y=a+bx
plot(points$x, points$y,
     pch = 19, cex = 2, bty = "n",
    xlim = c(0, 3.5), ylim = c(0,6), xlab = "x", ylab = "y")
abline(a = a, b = b, lwd = 2)
abline(v = c(1,2,3))  # If we used abline(h=c(0,1)), we would get horizont
p = my_line(c(1,2,3), a, b)
points(c(1,2,3), p, pch = 19, cex = 2, col = "red")
```

Let us return to the error

$$\overrightarrow{(x_1, y_1)(x_1, \tilde{y}_1)}$$

We have

$$\overrightarrow{(x_1, y_1)(x_1, \tilde{y}_1)} = (x_1 - x_1, y_1 - \tilde{y}_1) = (0, y_1 - \tilde{y}_1)$$

Let us call

$$\varepsilon_1 = y_1 - \tilde{y}_1$$

We can compute ε_2 and ε_3 too. And we can then form the **error vector**

$$\mathbf{e} = (\varepsilon_1, \varepsilon_2, \varepsilon_3)^T$$

The norm of $\mathbf{e}$, $\|\mathbf{e}\|$, then tells us how much error we are making for the choice of (a, b) we are using

The norm of $\mathbf{e}$, $\|\mathbf{e}\|$, tells us how much error we are making for the choice of (a, b) we are using

So our objective is to find (a, b) such that $\|\mathbf{e}\|$ is minimal

We could use various norms, but the Euclidean norm has some very interesting properties, so we use

$$\|\mathbf{e}\| = \sqrt{\varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2}$$

The linear least squares problem

Given a collection of data points $(x_1, y_1), \ldots, (x_n, y_n)$, find the coefficients a, b of the line $y = a + bx$ such that

$$\|\mathbf{e}\| = \sqrt{\varepsilon_1^2 + \cdots + \varepsilon_n^2} = \sqrt{(y_1 - \tilde{y}_1)^2 + \cdots + (y_n - \tilde{y}_n)^2}$$

is minimal, where $\tilde{y}_i = a + bx_i$, for $i = 1, \ldots, n$

Let us first hack a brute force solution! (For the example we have been using this far) We have our three points in the list 'points', the function `my_line` that computes the value $\tilde{y}$ given x and a, b, so let us make a new function that, given a, b, computes **e** We'll also pass the points 'points'

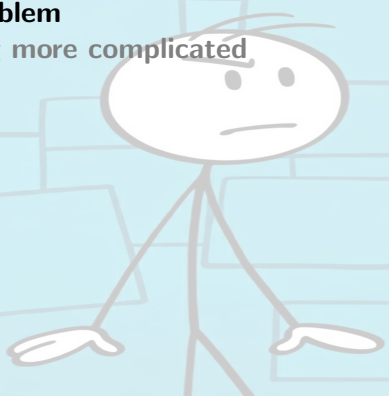
```
error = function(a_0, a_1, points) {
    y_tilde = my_line(points$x, a_0 = a_0, a_1 = a_1)
    e = points$y - y_tilde
    return(sqrt(sum(e^2)))
}
error(a_0 = 2, a_1 = 3, points)

## [1] 7.874008

error(a_0 = 3, a_1 = 0.5, points)

## [1] 1.224745

error(a_0 = 3.1, a_1 = 0.48, points)

## [1] 1.220471
```

Genetic algorithms

Let's use something cool: a *genetic algorithm*
- Genetic algorithms are a stochastic *optimisation* method. There are other types, e.g., gradient descent (deterministic) - The idea is to use a mechanism mimicking evolution's drive towards higher fitness - The function value is its fitness - We try different genes (here, different values of a, b) and evaluate their fitness.. keep the good ones - We mutate or crossover genes, throw in new ones, etc. - We keep doing this until we reach a stopping criterion - We then return the best gene we found

```r
if (!require("GA", quietly = TRUE)) {
  install.packages("GA")
  library(GA)
}
GA = ga(type = "real-valued",
        fitness = function(x) -error(a_0 = x[1], a_1 = x[2], points),
        suggestions = c(a_0 = 2, a_1 = 3),
        lower = c(-10, -10), upper = c(10, 10),
        popSize = 200, maxiter = 150)
# plot(GA)
GA

## An object of class "ga"
##
## Call:
## ga(type = "real-valued", fitness = function(x) -error(a_0 = x[1],     a
##
## Available slots:
```

Least squares problems



The least squares problem (simplest version)

Definition 1

Given a collection of points $(x_1, y_1), \ldots, (x_n, y_n)$, find the coefficients a, b of the line $y = a + bx$ such that

$$\|\mathbf{e}\| = \sqrt{\varepsilon_1^2 + \cdots + \varepsilon_n^2} = \sqrt{(y_1 - \tilde{y}_1)^2 + \cdots + (y_n - \tilde{y}_n)^2}$$

is minimal, where $\tilde{y}_i = a + bx_i$ for $i = 1, \ldots, n$

We just saw how to solve this by brute force using a genetic algorith to minimise $\|e\|$, let us now see how to solve this problem "properly"

For a data point $i = 1, \ldots, n$

$$\varepsilon_i = y_i - \tilde{y}_i = y_i - (a + bx_i)$$

So if we write this for all data points,

$$\begin{aligned} \varepsilon_1 &= y_1 - (a + bx_1) \\ &\vdots \\ \varepsilon_n &= y_n - (a + bx_n) \end{aligned}$$

In matrix form

$$\boldsymbol{e} = \boldsymbol{b} - A\boldsymbol{x}$$

with

$$\boldsymbol{e} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}, A = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \boldsymbol{x} = \begin{pmatrix} a \\ b \end{pmatrix} \text{ and } \boldsymbol{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

The least squares problem (reformulated)

Definition 2 (Least squares solutions)

Consider a collection of points $(x_1, y_1), \ldots, (x_n, y_n)$, a matrix $A \in \mathcal{M}_{mn}$, $\boldsymbol{b} \in \mathbb{R}^m$. A **least squares solution** of $A\boldsymbol{x} = \boldsymbol{b}$ is a vector $\tilde{\boldsymbol{x}} \in \mathbb{R}^n$ s.t.

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \quad \|\boldsymbol{b} - A\tilde{\boldsymbol{x}}\| \leq \|\boldsymbol{b} - A\boldsymbol{x}\|$$

Needed to solve the problem

Definition 3 (Best approximation)

Let V be a vector space, $W \subset V$ and $\mathbf{v} \in V$. The **best approximation** to $\mathbf{v}$ in W is $\tilde{\mathbf{v}} \in W$ s.t.

$$\forall \mathbf{w} \in W, \mathbf{w} \neq \tilde{\mathbf{v}}, \quad \|\mathbf{v} - \tilde{\mathbf{v}}\| < \|\mathbf{v} - \mathbf{w}\|$$

Theorem 4 (Best approximation theorem)

Let V be a vector space with an inner product, $W \subset V$ and $\mathbf{v} \in V$. Then $\mathrm{proj}_W(\mathbf{v})$ is the best approximation to $\mathbf{v}$ in W

Let us find the least squares solution

$\forall \boldsymbol{x} \mathbb{R}^n$, $A\boldsymbol{x}$ is a vector in the **column space** of A (the space spanned by the vectors making up the columns of A)

Since $\boldsymbol{x} \in \mathbb{R}^n$, $A\boldsymbol{x} \in \text{col}(A)$

$\implies$ least squares solution of $A\boldsymbol{x} = \boldsymbol{b}$ is a vector $\tilde{\boldsymbol{y}} \in \text{col}(A)$ s.t.

$$\forall \boldsymbol{y} \in \text{col}(A), \quad \|\boldsymbol{b} - \tilde{\boldsymbol{y}}\| \leq \|\boldsymbol{b} - \boldsymbol{y}\|$$

This looks very much like Best approximation and Best approximation theorem

Putting things together

We just stated: The least squares solution of $A\boldsymbol{x} = \boldsymbol{b}$ is a vector $\tilde{\boldsymbol{y}} \in \text{col}(A)$ s.t.

$$\forall \boldsymbol{y} \in \text{col}(A), \quad \|\boldsymbol{b} - \tilde{\boldsymbol{y}}\| \leq \|\boldsymbol{b} - \boldsymbol{y}\|$$

We know (reformulating a tad):

Theorem 5 (Best approximation theorem)

Let V be a vector space with an inner product, $W \subset V$ and $\mathbf{v} \in V$. Then $\text{proj}_W(\mathbf{v}) \in W$ is the best approximation to $\mathbf{v}$ in W, i.e.,

$$\forall \mathbf{w} \in W, \mathbf{w} \neq \text{proj}_W(\mathbf{v}), \quad \|\mathbf{v} - \text{proj}_W(\mathbf{v})\| < \|\mathbf{v} - \mathbf{w}\|$$

$\implies$ $W = \text{col}(A)$, $\boldsymbol{v} = \boldsymbol{b}$ and $\tilde{\boldsymbol{y}} = \text{proj}_{\text{col}(A)}(\mathbf{b})$

So if $\tilde{\boldsymbol{x}}$ is a least squares solution of $A\boldsymbol{x} = \boldsymbol{b}$, then

$$\tilde{\boldsymbol{y}} = A\tilde{\boldsymbol{x}} = \text{proj}_{\text{col}(A)}(\mathbf{b})$$

We have

$$\boldsymbol{b} - A\tilde{\boldsymbol{x}} = \boldsymbol{b} - \text{proj}_{\text{col}(A)}(\mathbf{b}) = \text{perp}_{\text{col}(A)}(\mathbf{b})$$

and it is easy to show that

$$\text{perp}_{\text{col}(A)}(\mathbf{b}) \perp \text{col}(A)$$

So for all columns $\boldsymbol{a}_i$ of A

$$\boldsymbol{a}_i \cdot (\boldsymbol{b} - A\tilde{\boldsymbol{x}}) = 0$$

which we can also write as $\boldsymbol{a}_i^T(\boldsymbol{b} - A\tilde{\boldsymbol{x}}) = 0$

For all columns $\boldsymbol{a}_i$ of A,

$$\boldsymbol{a}_i^T(\boldsymbol{b} - A\tilde{\boldsymbol{x}}) = 0$$

This is equivalent to saying that

$$A^T(\boldsymbol{b} - A\tilde{\boldsymbol{x}}) = \mathbf{0}$$

We have

$$\begin{aligned} A^T(\boldsymbol{b} - A\tilde{\boldsymbol{x}}) = \mathbf{0} &\iff A^T\boldsymbol{b} - A^TA\tilde{\boldsymbol{x}} = \mathbf{0} \\ &\iff A^T\boldsymbol{b} = A^TA\tilde{\boldsymbol{x}} \\ &\iff A^TA\tilde{\boldsymbol{x}} = A^T\boldsymbol{b} \end{aligned}$$

The latter system constitutes the **normal equations** for $\tilde{\boldsymbol{x}}$

Least squares theorem

Theorem 6 (Least squares theorem)

$A \in \mathcal{M}_{mn}$, $\boldsymbol{b} \in \mathbb{R}^m$. Then

1. *$A\boldsymbol{x} = \boldsymbol{b}$ always has at least one least squares solution $\tilde{\boldsymbol{x}}$*
2. *$\tilde{\boldsymbol{x}}$ least squares solution to $A\boldsymbol{x} = \boldsymbol{b}$ $\iff$ $\tilde{\boldsymbol{x}}$ is a solution to the normal equations $A^T A\tilde{\boldsymbol{x}} = A^T\boldsymbol{b}$*
3. *A has linearly independent columns $\iff$ $A^T A$ invertible.*
   In this case, the least squares solution is unique and

$$\tilde{\boldsymbol{x}} = \left(A^T A\right)^{-1} A^T \boldsymbol{b}$$

We have seen 1 and 2, we will not show 3 (it is not hard)

Least squares problems

Suppose we want to fit something a bit more complicated..

For instance, instead of the affine function

$$y = a + bx$$

suppose we want to do the quadratic

$$y = a_0 + a_1 x + a_2 x^2$$

or even

$$y = k_0 e^{k_1 x}$$

How do we proceed?

Fitting the quadratic

We have the data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ and want to fit

$$y = a_0 + a_1 x + a_2 x^2$$

At (x_1, y_1),

$$\tilde{y}_1 = a_0 + a_1 x_1 + a_2 x_1^2$$

$\vdots$

At (x_n, y_n),

$$\tilde{y}_n = a_0 + a_1 x_n + a_2 x_n^2$$

In terms of the error

$$\varepsilon_1 = y_1 - \tilde{y}_1 = y_1 - (a_0 + a_1 x_1 + a_2 x_1^2)$$
$$\vdots$$
$$\varepsilon_n = y_n - \tilde{y}_n = y_n - (a_0 + a_1 x_n + a_2 x_n^2)$$

i.e.,

$$\boldsymbol{e} = \boldsymbol{b} - A\boldsymbol{x}$$

where

$$\boldsymbol{e} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}, A = \begin{pmatrix} 1 & x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix}, \boldsymbol{x} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} \text{ and } \boldsymbol{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Theorem 6 applies, with here $A \in \mathcal{M}_{n3}$ and $\boldsymbol{b} \in \mathbb{R}^n$

Fitting the exponential

Things are a bit more complicated here

If we proceed as before, we get the system

$$\begin{aligned} y_1 &= k_0 e^{k_1 x_1} \\ &\vdots \\ y_n &= k_0 e^{k_1 x_n} \end{aligned}$$

$e^{k_1 x_i}$ is a nonlinear term, it cannot be put in a matrix

However: take the ln of both sides of the equation

$$\ln(y_i) = \ln(k_0 e^{k_1 x_i}) = \ln(k_0) + \ln(e^{k_1 x_i}) = \ln(k_0) + k_1 x_i$$

If $y_i, k_0 > 0$, then their ln are defined and we're in business..

$$\ln(y_i) = \ln(k_0) + k_1 x_i$$

So the system is

$$\boldsymbol{y} = A\boldsymbol{x} + \boldsymbol{b}$$

with

$$A = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \boldsymbol{x} = \begin{pmatrix} k_1 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} \ln(k_0) \end{pmatrix} \text{ and } \boldsymbol{y} = \begin{pmatrix} \ln(y_1) \\ \vdots \\ \ln(y_n) \end{pmatrix}$$

Least squares problems

QR factorisation

Singular values decomposition (SVD)

Principal component analysis (PCA)

Support vector machines

QR factorisation

Matrix factorisations

Matrix factorisations are popular because they allow to perform some computations more easily

There are several different types of factorisations. Here, we study just the QR factorisation, which is useful for many least squares problems

Definition 7 (Orthogonal set of vectors)

The set of vectors $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\} \in \mathbb{R}^n$ is an **orthogonal set** if

$$\forall i, j = 1, \ldots, k, \quad i \neq j \implies \boldsymbol{v}_i \bullet \boldsymbol{v}_j = 0$$

Theorem 8

$\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\} \in \mathbb{R}^n$ with $\forall i,\ \boldsymbol{v}_i \neq \boldsymbol{0}$, orthogonal set $\implies$ $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\} \in \mathbb{R}^n$ linearly independent

Definition 9 (Orthogonal basis)

Let S be a basis of the subspace $W \subset \mathbb{R}^n$ composed of an orthogonal set of vectors. We say S is an **orthogonal basis** of W

Proof of Theorem 8

Assume $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ orthogonal set with $\boldsymbol{v}_i \neq \boldsymbol{0}$ for all $i = 1, \ldots, k$. Recall $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ is LI if

$$c_1 \boldsymbol{v}_1 + \cdots + c_k \boldsymbol{v}_k = \boldsymbol{0} \iff c_1 = \cdots = c_k = 0$$

So assume $c_1, \ldots, c_k \in \mathbb{R}$ are s.t. $c_1 \boldsymbol{v}_1 + \cdots + c_k \boldsymbol{v}_k = \boldsymbol{0}$. Recall that $\forall \boldsymbol{x} \in \mathbb{R}^k$, $\boldsymbol{0}_k \bullet \boldsymbol{x} = 0$. So for some $\boldsymbol{v}_i \in \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$

$$\begin{aligned} 0 &= \boldsymbol{0} \bullet \boldsymbol{v}_i \\ &= (c_1 \boldsymbol{v}_1 + \cdots + c_k \boldsymbol{v}_k) \bullet \boldsymbol{v}_i \\ &= c_1 \boldsymbol{v}_1 \bullet \boldsymbol{v}_i + \cdots + c_k \boldsymbol{v}_k \bullet \boldsymbol{v}_i \end{aligned} \tag{1}$$

As $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ orthogonal, $\boldsymbol{v}_j \bullet \boldsymbol{v}_i = 0$ when $i \neq j$, (1) reduces to

$$c_i \boldsymbol{v}_i \bullet \boldsymbol{v}_i = 0 \iff c_i \|\boldsymbol{v}_i\|^2 = 0$$

As $\boldsymbol{v}_i \neq 0$ for all i, $\|\boldsymbol{v}_i\| \neq 0$ and so $c_i = 0$. This is true for all i, hence the result □

Example – Vectors of the standard basis of $\mathbb{R}^3$

For $\mathbb{R}^3$, we denote

$$\boldsymbol{i} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{j} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \text{ and } \boldsymbol{k} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

($\mathbb{R}^k$ for $k > 3$, we denote them $\boldsymbol{e}_i$)

Clearly, $\{\boldsymbol{i}, \boldsymbol{j}\}$, $\{\boldsymbol{i}, \boldsymbol{k}\}$, $\{\boldsymbol{j}, \boldsymbol{k}\}$ and $\{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}\}$ orthogonal sets. The standard basis vectors are also $\neq \mathbf{0}$, so the sets are LI. And

$$\{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}\}$$

is an orthogonal basis of $\mathbb{R}^3$ since it spans $\mathbb{R}^3$ and is LI

$$c_1\boldsymbol{i} + c_2\boldsymbol{j} + c_3\boldsymbol{k} = c_1 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + c_2 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + c_3 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

Orthonormal version of things

Definition 10 (Orthonormal set)

The set of vectors $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\} \in \mathbb{R}^n$ is an **orthonormal set** if it is an orthogonal set and furthermore

$$\forall i = 1, \ldots, k, \quad \|\boldsymbol{v}_i\| = 1$$

Definition 11 (Orthonormal basis)

A basis of the subspace $W \subset \mathbb{R}^n$ is an **orthonormal basis** if the vectors composing it are an orthonormal set

$\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\} \in \mathbb{R}^n$ is orthonormal if

$$\boldsymbol{v}_i \bullet \boldsymbol{v}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Projections

Definition 12 (Orthogonal projection onto a subspace)

$W \subset \mathbb{R}^n$ a subspace and $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$ an orthogonal basis of W. $\forall \boldsymbol{v} \in \mathbb{R}^n$, the **orthogonal projection** of $\boldsymbol{v}$ **onto** W is

$$\text{proj}_W(\boldsymbol{v}) = \frac{\boldsymbol{u}_1 \bullet \boldsymbol{v}}{\|\boldsymbol{u}_1\|^2}\boldsymbol{u}_1 + \cdots + \frac{\boldsymbol{u}_k \bullet \boldsymbol{v}}{\|\boldsymbol{u}_k\|^2}\boldsymbol{u}_k$$

Definition 13 (Component orthogonal to a subspace)

$W \subset \mathbb{R}^n$ a subspace and $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$ an orthogonal basis of W. $\forall \boldsymbol{v} \in \mathbb{R}^n$, the **component** of $\boldsymbol{v}$ **orthogonal to** W is

$$\text{perp}_W(\boldsymbol{v}) = \boldsymbol{v} - \text{proj}_W(\boldsymbol{v})$$

What this aims to do is to construct an orthogonal basis for a subspace $W \subset \mathbb{R}^n$

To do this, we use the *Gram-Schmidt orthogonalisation process*, which turn s a basis of W into an orthogonal basis of W

Gram-Schmidt process

Theorem 14

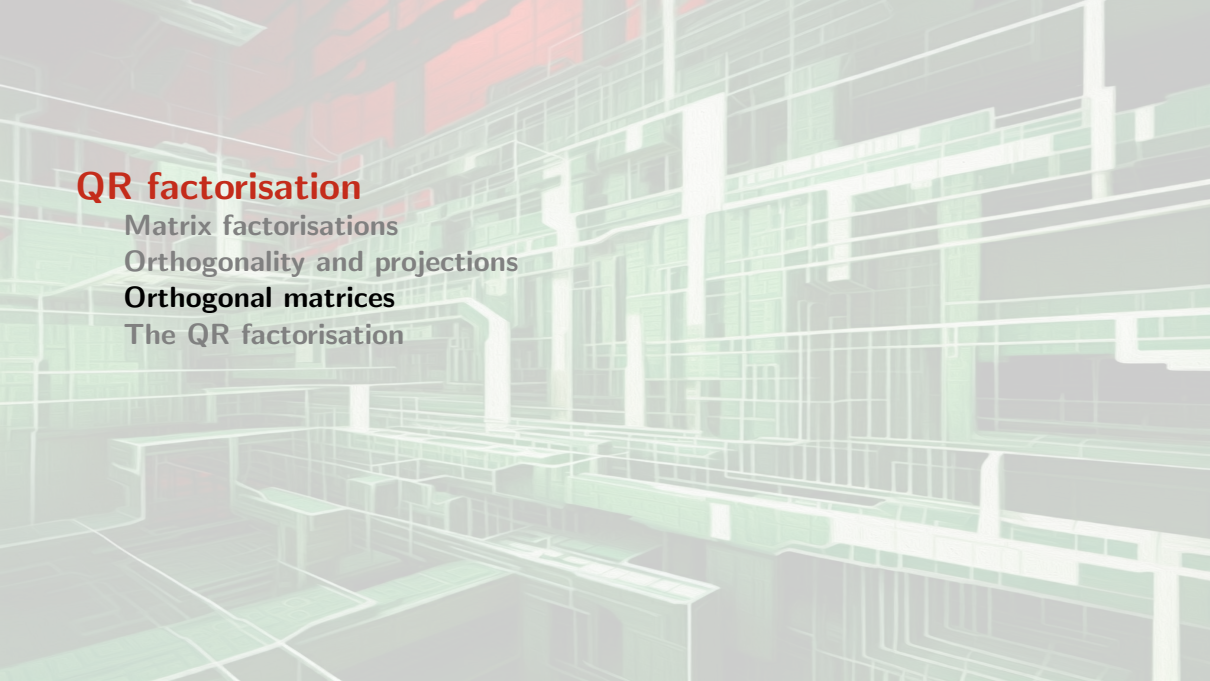
$W \subset \mathbb{R}^n$ a subset and $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ a basis of W. Let

$$\begin{aligned}
\boldsymbol{v}_1 &= \boldsymbol{x}_1 \\
\boldsymbol{v}_2 &= \boldsymbol{x}_2 - \frac{\boldsymbol{v}_1 \bullet \boldsymbol{x}_2}{\|\boldsymbol{v}_1\|^2}\boldsymbol{v}_1 \\
\boldsymbol{v}_3 &= \boldsymbol{x}_3 - \frac{\boldsymbol{v}_1 \bullet \boldsymbol{x}_3}{\|\boldsymbol{v}_1\|^2}\boldsymbol{v}_1 - \frac{\boldsymbol{v}_2 \bullet \boldsymbol{x}_3}{\|\boldsymbol{v}_2\|^2}\boldsymbol{v}_2 \\
&\vdots \\
\boldsymbol{v}_k &= \boldsymbol{x}_k - \frac{\boldsymbol{v}_1 \bullet \boldsymbol{x}_k}{\|\boldsymbol{v}_1\|^2}\boldsymbol{v}_1 - \cdots - \frac{\boldsymbol{v}_{k-1} \bullet \boldsymbol{x}_k}{\|\boldsymbol{v}_{k-1}\|^2}\boldsymbol{v}_{k-1}
\end{aligned}$$

and

$$W_1 = \operatorname{span}(\boldsymbol{x}_1), W_2 = \operatorname{span}(\boldsymbol{x}_1, \boldsymbol{x}_2), \ldots, W_k = \operatorname{span}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$$

Then $\forall i = 1, \ldots, k$, $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_i\}$ orthogonal basis for W_i



QR factorisation

Theorem 15

Let $Q \in \mathcal{M}_{mn}$. The columns of Q form an orthonormal set if and only if

$$Q^T Q = \mathbb{I}_n$$

Definition 16 (Orthogonal matrix)

$Q \in \mathcal{M}_n$ is an **orthogonal matrix** if its columns form an orthonormal set

So $Q \in \mathcal{M}_n$ orthogonal if $Q^T Q = \mathbb{I}$, i.e., $Q^T = Q^{-1}$

Theorem 17 (NSC for orthogonality)

$Q \in \mathcal{M}_n$ *orthogonal* $\iff$ $Q^{-1} = Q^T$

Theorem 18 (Orthogonal matrices "encode" isometries)

Let $Q \in \mathcal{M}_n$. TFAE

1. *Q orthogonal*
2. $\forall \boldsymbol{x} \in \mathbb{R}^n$, $\|Q\boldsymbol{x}\| = \|\boldsymbol{x}\|$
3. $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, $Q\boldsymbol{x} \bullet Q\boldsymbol{y} = \boldsymbol{x} \bullet \boldsymbol{y}$

Theorem 19

Let $Q \in \mathcal{M}_n$ be orthogonal. Then

1. *The rows of Q form an orthonormal set*
2. *Q^{-1} orthogonal*
3. $\det Q = \pm 1$
4. $\forall \lambda \in \sigma(Q)$, $|\lambda| = 1$
5. *If $Q_2 \in \mathcal{M}_n$ also orthogonal, then QQ_2 orthogonal*

Proof of 4 in Theorem 19

All statements in Theorem 19 are easy, but let's focus on 4

Let λ be an eigenvalue of $Q \in \mathcal{M}_n$ orthogonal, i.e., $\exists \mathbb{R}^n \ni \boldsymbol{x} \neq \mathbf{0}$ s.t.

$$Q\boldsymbol{x} = \lambda \boldsymbol{x}$$

Take the norm on both sides

$$\|Q\boldsymbol{x}\| = \|\lambda \boldsymbol{x}\|$$

From 2 in Theorem 18, $\|Q\boldsymbol{x}\| = \|\boldsymbol{x}\|$ and from the properties of norms, $\|\lambda \boldsymbol{x}\| = |\lambda| \, \|\boldsymbol{x}\|$, so we have

$$\|Q\boldsymbol{x}\| = \|\lambda \boldsymbol{x}\| \iff \|\boldsymbol{x}\| = |\lambda| \, \|\boldsymbol{x}\| \iff 1 = |\lambda|$$

(we can divide by $\|\boldsymbol{x}\|$ since $\boldsymbol{x} \neq \mathbf{0}$ as an eigenvector) □

The QR factorisation

Theorem 20

Let $A \in \mathcal{M}_{mn}$ with LI columns. Then A can be factored as

$$A = QR$$

where $Q \in \mathcal{M}_{mn}$ has orthonormal columns and $R \in \mathcal{M}_n$ is nonsingular upper triangular

Back to least squares

So what was the point of all that..?

Theorem 21 (Least squares with QR factorisation)

$A \in \mathcal{M}_{mn}$ with LI columns, $\boldsymbol{b} \in \mathbb{R}^m$. If $A = QR$ is a QR factorisation of A, then the unique least squares solution $\tilde{\boldsymbol{x}}$ of $A\boldsymbol{x} = \boldsymbol{b}$ is

$$\tilde{\boldsymbol{x}} = R^{-1}Q^T\boldsymbol{b}$$

Proof of Theorem 21

A has LI columns so

- least squares $A\boldsymbol{x} = \boldsymbol{b}$ has unique solution $\tilde{\boldsymbol{x}} = (A^T A)^{-1} A^T \boldsymbol{b}$
- by Theorem 20, A can be written as $A = QR$ with $Q \in \mathcal{M}_{mn}$ with orthonormal columns and $R \in \mathcal{M}_n$ nonsingular and upper triangular

So

$$\begin{aligned} A^T A\tilde{\boldsymbol{x}} = A^T \boldsymbol{b} &\implies (QR)^T QR\tilde{\boldsymbol{x}} = (QR)^T \boldsymbol{b} \\ &\implies R^T Q^T QR\tilde{\boldsymbol{x}} = R^T Q^T \boldsymbol{b} \\ &\implies R^T \mathbb{I}_n R\tilde{\boldsymbol{x}} = R^T Q^T \boldsymbol{b} \\ &\implies R^T R\tilde{\boldsymbol{x}} = R^T Q^T \boldsymbol{b} \\ &\implies (R^T)^{-1} R\tilde{\boldsymbol{x}} = (R^T)^{-1} R^T Q^T \boldsymbol{b} \\ &\implies R\tilde{\boldsymbol{x}} = Q^T \boldsymbol{b} \\ &\implies \tilde{\boldsymbol{x}} = R^{-1} Q^T \boldsymbol{b} \quad \square \end{aligned}$$

Least squares problems

QR factorisation

Singular values decomposition (SVD)

Principal component analysis (PCA)

Support vector machines

Matrix factorisations (continued)

The singular value decomposition (known mostly by its acronym, SVD) is yet another type of factorisation/decomposition..

Singular values decomposition (SVD)

Singular values

Definition 22 (Singular value)

Let $A \in \mathcal{M}_{mn}(\mathbb{R})$. The **singular values** of A are the real numbers

$$\sigma_1 \geq \sigma_2 \geq \cdots \sigma_n \geq 0$$

that are the square roots of the eigenvalues of $A^T A$

Singular values are real and nonnegative?

Recall that $\forall A \in \mathcal{M}_{mn}$, $A^T A$ is symmetric

Claim 1. Real symmetric matrices have real eigenvalues

Proof. $A \in \mathcal{M}_n(\mathbb{R})$ symmetric and $(\lambda, \boldsymbol{v})$ eigenpair of A, i.e, $A\boldsymbol{v} = \lambda \boldsymbol{v}$. Taking the complex conjugate, $\overline{A\boldsymbol{v}} = \overline{\lambda \boldsymbol{v}}$

Since $A \in \mathcal{M}_n(\mathbb{R})$, $\overline{A} = A \qquad (z = \bar{z} \iff z \in \mathbb{R})$

So

$$A\bar{\boldsymbol{v}} = \overline{A}\bar{\boldsymbol{v}} = \overline{A\boldsymbol{v}} = \overline{\lambda \boldsymbol{v}} = \bar{\lambda}\bar{\boldsymbol{v}}$$

i.e., if $(\lambda, \boldsymbol{v})$ eigenpair, $(\bar{\lambda}, \bar{\boldsymbol{v}})$ also eigenpair

Still assuming $A \in \mathcal{M}_n(\mathbb{R})$ symmetric and $(\lambda, \boldsymbol{v})$ eigenpair of A and using what we just proved (that $(\bar{\lambda}, \bar{\boldsymbol{v}})$ also eigenpair), take transposes

$$\begin{aligned} A\bar{\boldsymbol{v}} = \bar{\lambda}\bar{\boldsymbol{v}} &\iff (A\bar{\boldsymbol{v}})^T = (\bar{\lambda}\bar{\boldsymbol{v}})^T \\ &\iff \bar{\boldsymbol{v}}^T A^T = \bar{\lambda}\bar{\boldsymbol{v}}^T \\ &\iff \bar{\boldsymbol{v}}^T A = \bar{\lambda}\bar{\boldsymbol{v}}^T \qquad [A \text{ symmetric}] \end{aligned}$$

Let us now compute $\lambda(\bar{\boldsymbol{v}} \bullet \boldsymbol{v})$. We have

$$\begin{aligned} \lambda(\bar{\boldsymbol{v}} \bullet \boldsymbol{v}) &= \lambda\bar{\boldsymbol{v}}^T\boldsymbol{v} = \bar{\boldsymbol{v}}^T(\lambda\boldsymbol{v}) \\ &= \bar{\boldsymbol{v}}^T(A\boldsymbol{v}) = (\bar{\boldsymbol{v}}^T A)\boldsymbol{v} \\ &= (\bar{\lambda}\bar{\boldsymbol{v}}^T)\boldsymbol{v} = \bar{\lambda}(\bar{\boldsymbol{v}} \bullet \boldsymbol{v}) \\ &\iff (\lambda - \bar{\lambda})(\bar{\boldsymbol{v}} \bullet \boldsymbol{v}) = 0 \end{aligned}$$

We have shown

$$(\lambda - \bar{\lambda})(\bar{\boldsymbol{v}} \bullet \boldsymbol{v}) = 0$$

Let

$$\boldsymbol{v} = \begin{pmatrix} a_1 + ib_1 \\ \vdots \\ a_n + ib_n \end{pmatrix}$$

Then

$$\bar{\boldsymbol{v}} = \begin{pmatrix} a_1 - ib_1 \\ \vdots \\ a_n - ib_n \end{pmatrix}$$

So

$$\bar{\boldsymbol{v}} \bullet \boldsymbol{v} = (a_1^2 + b_1^2) + \cdots + (a_n^2 + b_n^2)$$

But $\boldsymbol{v}$ eigenvector is $\neq \mathbf{0}$, so $\bar{\boldsymbol{v}} \bullet \boldsymbol{v} \neq 0$, so

$$(\lambda - \bar{\lambda})(\bar{\boldsymbol{v}} \bullet \boldsymbol{v}) = 0 \iff \lambda - \bar{\lambda} = 0 \iff \lambda = \bar{\lambda} \iff \lambda \in \mathbb{R} \quad \square$$

Claim 2. For $A \in \mathcal{M}_{mn}(\mathbb{R})$, the eigenvalues of A^TA are real and nonnegative

Proof. We know that for $A \in \mathcal{M}_{mn}$, A^TA symmetric and from previous claim, if $A \in \mathcal{M}_{mn}(\mathbb{R})$, then A^TA is symmetric and real and with real eigenvalues

Let $(\lambda, \boldsymbol{v})$ be an eigenpair of A^TA, with $\boldsymbol{v}$ chosen so that $\|\boldsymbol{v}\| = 1$

Norms are functions $V \to \mathbb{R}_+$, so $\|A\boldsymbol{v}\|$ and $\|A\boldsymbol{v}\|^2$ are ≥ 0 and thus

$$\begin{aligned} 0 \leq \|A\boldsymbol{v}\|^2 &= (A\boldsymbol{v}) \bullet (A\boldsymbol{v}) = (A\boldsymbol{v})^T(A\boldsymbol{v}) \\ &= \boldsymbol{v}^TA^TA\boldsymbol{v} = \boldsymbol{v}^T(A^TA\boldsymbol{v}) = \boldsymbol{v}^T(\lambda\boldsymbol{v}) \\ &= \lambda(\boldsymbol{v}^T\boldsymbol{v}) = \lambda(\boldsymbol{v} \bullet \boldsymbol{v}) = \lambda\|\boldsymbol{v}\|^2 \\ &= \lambda \quad \square \end{aligned}$$

Claim 3. For $A \in \mathcal{M}_{mn}(\mathbb{R})$, the nonzero eigenvalues of A^TA and AA^T are the same

Proof. Let $(\lambda, \boldsymbol{v})$ be an eigenpair of A^TA with $\lambda \neq 0$. Then $\boldsymbol{v} \neq \mathbf{0}$ and

$$A^TA\boldsymbol{v} = \lambda\boldsymbol{v} \neq \mathbf{0}$$

Left multiply by A

$$AA^TA\boldsymbol{v} = \lambda A\boldsymbol{v}$$

Let $\boldsymbol{w} = A\boldsymbol{v}$, we thus have $AA^T\boldsymbol{w} = \lambda\boldsymbol{w}$; in other words, $A\boldsymbol{v}$ is an eigenvector of AA^T corresponding to the (nonzero) eigenvalue λ

The reverse works the same way.. □

Singular values decomposition (SVD)

The singular value decomposition (SVD)

Theorem 23 (SVD)

$A \in \mathcal{M}_{mn}$ with singular values $\sigma_1 \geq \cdots \geq \sigma_r > 0$ and $\sigma_{r+1} = \cdots = \sigma_n = 0$

Then there exists $U \in \mathcal{M}_m$ orthogonal, $V \in \mathcal{M}_n$ orthogonal and a block matrix $\Sigma \in \mathcal{M}_{mn}$ taking the form

$$\Sigma = \begin{pmatrix} D & 0_{r,n-r} \\ 0_{m-r,r} & 0_{m-r,n-r} \end{pmatrix}$$

where

$$D = \mathrm{diag}(\sigma_1, \ldots, \sigma_r) \in \mathcal{M}_r$$

such that

$$A = U\Sigma V^T$$

Definition 24

We call a factorisation as in Theorem 23 the **singular value decomposition** of A. The columns of U and V are, respectively, the **left** and **right singular vectors** of A

U and V^T are *rotation* or *reflection* matrices, Σ is a *scaling* matrix

$U \in \mathcal{M}_m$ orthogonal matrix with columns the eigenvectors of AA^T

$V \in \mathcal{M}_n$ orthogonal matrix with columns the eigenvectors of A^TA

Outer product form of the SVD

Theorem 25 (Outer product form of the SVD)

$A \in \mathcal{M}_{mn}$ with singular values $\sigma_1 \geq \cdots \geq \sigma_r > 0$ and $\sigma_{r+1} = \cdots = \sigma_n = 0$, $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r$, respectively, left and right singular vectors of A corresponding to these singular values

Then

$$A = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \cdots + \sigma_r \boldsymbol{u}_r \boldsymbol{v}_r^T \tag{2}$$

Computing the SVD (case of $\neq$ eigenvalues)

To compute the SVD, we use the following result

Theorem 26

Let $A \in \mathcal{M}_n$ symmetric, $(\lambda_1, \boldsymbol{u}_1)$ and $(\lambda_2, \boldsymbol{u}_2)$ be eigenpairs, $\lambda_1 \neq \lambda_2$. Then $\boldsymbol{u}_1 \bullet \boldsymbol{u}_2 = 0$

Proof of Theorem 26

$A \in \mathcal{M}_n$ symmetric, $(\lambda_1, \boldsymbol{u}_1)$ and $(\lambda_2, \boldsymbol{u}_2)$ eigenpairs with $\lambda_1 \neq \lambda_2$

$$\begin{aligned}
\lambda_1(\boldsymbol{v}_1 \bullet \boldsymbol{v}_2) &= (\lambda_1 \boldsymbol{v}_1) \bullet \boldsymbol{v}_2 \\
&= A\boldsymbol{v}_1 \bullet \boldsymbol{v}_2 \\
&= (A\boldsymbol{v}_1)^T \boldsymbol{v}_2 \\
&= \boldsymbol{v}_1^T A^T \boldsymbol{v}_2 \\
&= \boldsymbol{v}_1^T (A\boldsymbol{v}_2) \qquad [A \text{ symmetric so } A^T = A] \\
&= \boldsymbol{v}_1^T (\lambda_2 \boldsymbol{v}_2) \\
&= \lambda_2 (\boldsymbol{v}_1^T \boldsymbol{v}_2) \\
&= \lambda_2 (\boldsymbol{v}_1 \bullet \boldsymbol{v}_2)
\end{aligned}$$

So $(\lambda_1 - \lambda_2)(\boldsymbol{v}_1 \bullet \boldsymbol{v}_2) = 0$. But $\lambda_1 \neq \lambda_2$, so $\boldsymbol{v}_1 \bullet \boldsymbol{v}_2 = 0$ □

Computing the SVD (case of $\neq$ eigenvalues)

If all eigenvalues of A^TA (or AA^T) are distinct, we can use Theorem 26

1. Compute $A^TA \in \mathcal{M}_n$
2. Compute eigenvalues $\lambda_1, \ldots, \lambda_n$ of A^TA; order them as $\lambda_1 > \cdots > \lambda_n \geq 0$ ($>$ not $\geq$ since $\neq$)
3. Compute singular values $\sigma_1 = \sqrt{\lambda_1}, \ldots, \sigma_n = \sqrt{\lambda_n}$
4. Diagonal matrix D in Σ is either in $\mathcal{M}_n$ (if $\sigma_n > 0$) or in $\mathcal{M}_{n-1}$ (if $\sigma_n = 0$)

5. Since eigenvalues are distinct, Theorem 26 $\implies$ eigenvectors are orthogonal set. Compute these eigenvectors in the same order as the eigenvalues
6. Normalise them and use them to make the matrix V, i.e., $V = [\boldsymbol{v}_1 \cdots \boldsymbol{v}_n]$
7. To find the $\boldsymbol{u}_i$, compute, for $i = 1, \ldots, r$,

$$\boldsymbol{u}_i = \frac{1}{\sigma_i} A \boldsymbol{v}_i$$

and ensure that $\|\boldsymbol{u}_i\| = 1$

Computing the SVD (case where some eigenvalues are $=$)

1. Compute $A^T A \in \mathcal{M}_n$
2. Compute eigenvalues $\lambda_1, \ldots, \lambda_n$ of $A^T A$; order them as $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$
3. Compute singular values $\sigma_1 = \sqrt{\lambda_1}, \ldots, \sigma_n = \sqrt{\lambda_n}$, with $r \leq n$ the index of the last positive singular value
4. For eigenvalues that are distinct, proceed as before
5. For eigenvalues with multiplicity > 1, we need to ensure that the resulting eigenvectors are LI *and* orthogonal

Dealing with eigenvalues with multiplicity > 1

When an eigenvalue has (algebraic) multiplicity > 1, e.g., characteristic polynomial contains a factor like $(\lambda - 2)^2$, things can become a little bit more complicated

The proper way to deal with this involves the so-called Jordan Normal Form (another matrix decomposition)

In short: not all square matrices are diagonalisable, but all square matrices admit a JNF

Sometimes, we can find several LI eigenvectors associated to the same eigenvalue. Check this. If not, need to use the following

Definition 27 (Generalised eigenvectors)

$\boldsymbol{x} \neq \mathbf{0}$ **generalized eigenvector** of rank m of $A \in \mathcal{M}_n$ corresponding to eigenvalue λ if

$$(A - \lambda \mathbb{I})^m \boldsymbol{x} = \mathbf{0}$$

but

$$(A - \lambda \mathbb{I})^{m-1} \boldsymbol{x} \neq \mathbf{0}$$

Procedure for generalised eigenvectors

$A \in \mathcal{M}_n$ and assume λ eigenvalue with algebraic multiplicity k

Find $\boldsymbol{v}_1$, "classic" eigenvector, i.e., $\boldsymbol{v}_1 \neq \mathbf{0}$ s.t. $(A - \lambda \mathbb{I})\boldsymbol{v}_1 = \mathbf{0}$

Find generalised eigenvector $\boldsymbol{v}_2$ of rank 2 by solving for $\boldsymbol{v}_2 \neq \mathbf{0}$,

$$(A - \lambda \mathbb{I})\boldsymbol{v}_2 = \boldsymbol{v}_1$$

...

Find generalised eigenvector $\boldsymbol{v}_k$ of rank k by solving for $\boldsymbol{v}_k \neq \mathbf{0}$,

$$(A - \lambda \mathbb{I})\boldsymbol{v}_k = \boldsymbol{v}_{k-1}$$

Then $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ LI

Back to the normal procedure

With the LI eigenvectors $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ corresponding to λ

Apply Gram-Schmidt to get orthogonal set

For all eigenvalues with multiplicity > 1, check that you either have LI eigenvectors or do what we just did

When you are done, be back on your merry way to step 6 in the case where eigenvalues are all $\neq$

I am caricaturing a little here: there can be cases that do not work exactly like this, but this is general enough..

Singular values decomposition (SVD)

Applications of the SVD

Many applications of the SVD, both theoretical and practical..

1. Obtaining a unique solutions to least squares when $A^T A$ singular
2. Image compression

Least squares revisited

Theorem 28

Let $A \in \mathcal{M}_{mn}$, $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^m$. The least squares problem $A\boldsymbol{x} = \boldsymbol{b}$ has a unique least squares solution $\tilde{\boldsymbol{x}}$ of minimal length (closest to the origin) given by

$$\tilde{\boldsymbol{x}} = A^+ \boldsymbol{b}$$

where A^+ is the pseudoinverse of A

Definition 29 (Pseudoinverse)

$A = U\Sigma V^T$ an SVD for $A \in \mathcal{M}_{mn}$, where

$$\Sigma = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}, \text{ with } D = \text{diag}(\sigma_1, \ldots, \sigma_r)$$

(D contains the nonzero singular values of A ordered as usual)

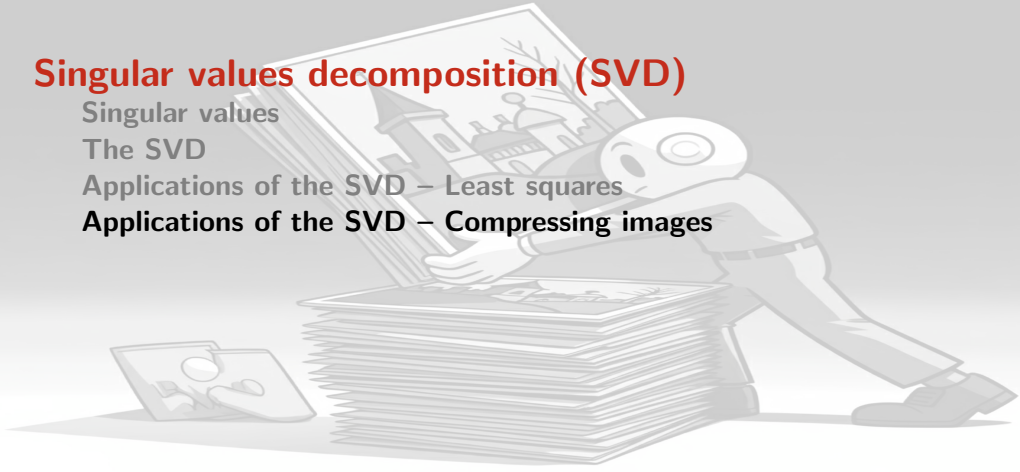
The **pseudoinverse** (or **Moore-Penrose inverse**) of A is $A^+ \in \mathcal{M}_{nm}$ given by

$$A^+ = V\Sigma^+ U^T$$

with

$$\Sigma^+ = \begin{pmatrix} D^{-1} & 0 \\ 0 & 0 \end{pmatrix} \in \mathcal{M}_{nm}$$

Singular values decomposition (SVD)



Compressing images

Consider an image (for simplicity, assume in shades of grey). This can be stored in a matrix $A \in \mathcal{M}_{mn}$

Take the SVD of A. Then the small singular values carry information about the regions with little variation and can perhaps be omitted, whereas the large singular values carry information about more "dynamic" regions of the image

Suppose A has r nonzero singular values. For $k \leq r$, let

$$A_k = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \cdots + \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k^T$$

For $k = r$ we get the usual outer product form (2)

Load the image using `bmp::read.bmp`

```
my_image = bmp::read.bmp("../CODE/Julien_and_friend_1000x800.bmp")
my_image_g = pixmap::pixmapGrey(my_image)
my_image_g
```

```
## Pixmap image
##   Type          : pixmapGrey
##   Size          : 800x1000
##   Resolution    : 1x1
##   Bounding box  : 0 0 1000 800
```

Doing the computations "by hand"

```
M = my_image_g@grey
MTM = t(M) %*% M
# Ensure matrix is symmetric
MTM = (MTM+t(MTM))/2
ev = eigen(MTM)
```

Given the size and nature of the entries, the matrix $M^T M$ is symmetric only to 1e−5 precision, so we use a little trick to make it symmetric no matter what: take the average of $M^T M$ and its transpose MM^T

Which version of the algorithm to use?

Make zero the eigenvalues that are close to zero (200 out of 1000)

```
ev$values = ev$values*(ev$values>1e-10)
```

Can we use the algorithm for all eigenvalues being distinct or do we have repeated ones?

```
any(duplicated(ev$values[ev$values>1e-10]))

## [1] FALSE
```

So we can use the standard algorithm

Computing the SVD

```
idx_positive_ev = which(ev$values>1e-10)
sv = sqrt(ev$values[idx_positive_ev])
```

Computing the SVD

Then $D = \text{diag}(\sigma_1, \ldots, \sigma_r)$, V is the matrix of normalised eigenvectors in the same order as the σ_i and for $i = 1, \ldots, r$

$$\mathbf{u}_i = \frac{1}{\sigma_i} A \mathbf{v}_i$$

ensuring that $\|\mathbf{u}_i\| = 1$

```
D = diag(sv)
V = ev$vectors[idx_positive_ev, idx_positive_ev]
c1 = colSums(V)
for (i in 1:dim(V)[2]) {
    V[,i] = V[,i]/c1[i]
}
```

Computing the SVD

Finally, we compute the $\boldsymbol{u}_i$'s

```
U = M %*% V %*% diag(1/sv)

## Error in M %*% V: non-conformable arguments

r = length(sv)
im = list(u=U, d=sv, v=V)

## Error in eval(expr, envir, enclos):  object 'U' not found
```

Using built-in functions

We can also use the built-in function `svd` to compute the SVD of M

```
M.svd = svd(M)
```

The results are stored in a list with components `u`, `d` and `v`

Make function to recreate an image from the SVD

Given the SVD `im` of an image and a number of singular values to keep `n`, we can recreate the image using the function `compress_image`

We output the new image, but also, the amount of information required to encode this new image, as a percentage of the original image size

```
compress_image = function(im, n) {
  if (n > length(im$d)) {
    # Check that we gave a value of n within range, otherwise
    # just set to the max
    n = length(im$d)
  }
  d_tmp = im$d[1:n]
  u_tmp = im$u[,1:n]
  v_tmp = im$v[,1:n]
  # We store the results in a list (so we can return other information)
    out = list()
    # First, compute the resulting image
    out$img = mat.or.vec(nr = dim(im$u)[1], nc = dim(im$v)[1])
    for (i in 1:n) {
        out$img = out$img + d_tmp[i] * u_tmp[,i] %*% t(v_tmp[,i])
    }
```

```
    # Values of the "colours" must be between 0 and 1, so we shift and res
    if (min(min(out$img)) < 0 ) {
        out$img = out$img - min(min(out$img))
    }
    out$img = out$img / max(max(out$img))
    # Store some information: number of points needed and percentage of th
    out$nb_pixels_original = dim(im$u)[1] * dim(im$v)[2]
    out$nb_pixels_compressed = length(d_tmp) + dim(u_tmp)[1]*dim(u_tmp)[2]
    out$pct_of_original = out$nb_pixels_compressed / out$nb_pixels_original
    # Return the result
    return(out)
}
```

Recreating the image

We can now recreate the image using the function `compress_image`

```
new_image = my_image_g
M.svd = svd(M)
M_tmp = compress_image(M.svd, 2)
new_image@grey = M_tmp$img
plot(new_image)
```

Using $n = 2$ singular values

Uses 0.56% of the original information

Using $n = 5$ singular values

Uses 1.41% of the original information

Using $n = 10$ singular values

Uses 2.81% of the original information

Using $n = 20$ singular values

Uses 5.63% of the original information

Using $n = 50$ singular values

Uses 14.07% of the original information

Least squares problems

QR factorisation

Singular values decomposition (SVD)

Principal component analysis (PCA)

Support vector machines

Dimensionality reduction

One of the reasons the SVD is used is for dimensionality reduction. However, SVD has many many other uses

Now we look at another dimensionality reduction technique, PCA

PCA is often used as a blackbox technique, here we take a look at the math behind it

What is PCA?

Linear algebraic technique

Helps reduce a complex dataset to a lower dimensional one

Non-parametric method: does not assume anything about data distribution (distribution from the statistical point of view)

Principal component analysis (PCA)

Brief "review" of some probability concepts

Proper definition of *probability* requires to use *measure theory*.. will not get into details here

A **random variable** X is a *measurable* function $X : \Omega \to E$, where Ω is a set of outcomes (*sample space*) and E is a measurable space

$$\mathbb{P}(X \in S \subseteq E) = \mathbb{P}(\omega \in \Omega | X(\omega) \in S)$$

Distribution function of a r.v., $F(x) = \mathbb{P}(X \leq x)$, describes the distribution of a r.v.

R.v. can be discrete or continuous or .. other things.

Definition 30 (Variance)

Let X be a random variable. The **variance** of X is given by

$$\text{Var } X = E\left[(X - E(X))^2\right]$$

where E is the expected value

Definition 31 (Covariance)

Let X, Y be jointly distributed random variables. The **covariance** of X and Y is given by

$$\text{cov}(X, Y) = E\left[(X - E(X))\,(Y - E(Y))\right]$$

Note that $\text{cov}(X, X) = E\left[(X - E(X))^2\right] = \text{Var } X$

In practice: "true law" versus "observation"

In statistics: we reason on the *true law* of distributions, but we usually have only access to a sample

We then use **estimators** to .. estimate the value of a parameter, e.g., the mean, variance and covariance

Definition 32 (Unbiased estimators of the mean and variance)

Let $x_1, \ldots, x_n$ be data points (the *sample*) and

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

be the **mean** of the data. An unbiased estimator of the variance of the sample is

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

Definition 33 (Unbiased estimator of the covariance)

Let $(x_1, y_1), \ldots, (x_n, y_n)$ be data points,

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \text{ and } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

be the means of the data. An estimator of the covariance of the sample is

$$\text{cov}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

What does covariance do?

Variance explains how data disperses around the mean, in a 1-D case

Covariance measures the relationship between two dimensions. E.g., height and weight

More than the exact value, the sign is important:

- $\text{cov}(X, Y) > 0$: both dimensions change in the same "direction"; e.g., larger height usually means higher weight
- $\text{cov}(X, Y) < 0$: both dimensions change in reverse directions; e.g., time spent on social media and performance in this class
- $\text{cov}(X, Y) = 0$: the dimensions are independent from one another; e.g., sex/gender and "intelligence"

The covariance matrix

Typically, we consider more than 2 variables..

Definition 34

Suppose p random variables $X_1, \ldots, X_p$. Then the covariance matrix is the symmetric matrix

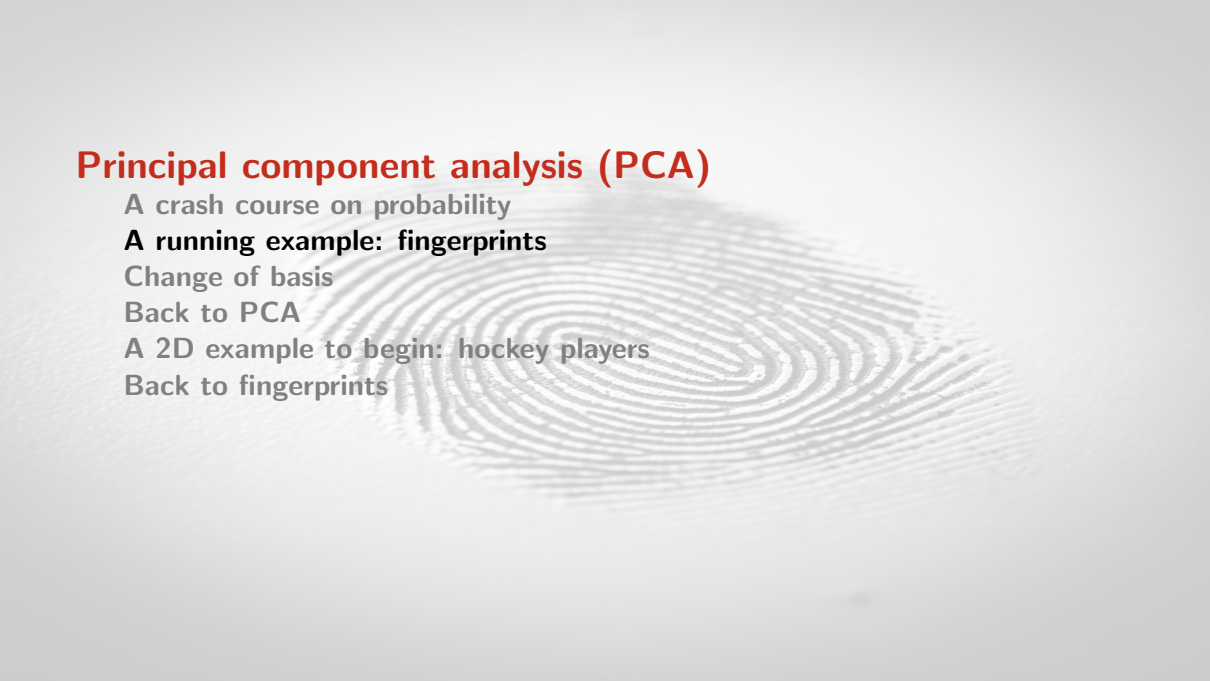
$$\begin{pmatrix} \operatorname{cov}(X_1, X_1) & \operatorname{cov}(X_1, X_2) & \cdots & \operatorname{cov}(X_1, X_p) \\ \operatorname{cov}(X_2, X_1) & \operatorname{cov}(X_2, X_2) & \cdots & \operatorname{cov}(X_2, X_p) \\ \vdots & \vdots & & \vdots \\ \operatorname{cov}(X_p, X_1) & \operatorname{cov}(X_p, X_2) & \cdots & \operatorname{cov}(X_p, X_p) \end{pmatrix}$$

i.e., using the properties of covariance,

$$\begin{pmatrix} \operatorname{Var} X_1 & \operatorname{cov}(X_1, X_2) & \cdots & \operatorname{cov}(X_1, X_p) \\ \operatorname{cov}(X_1, X_2) & \operatorname{Var} X_2 & \cdots & \operatorname{cov}(X_2, X_p) \\ \vdots & \vdots & & \vdots \\ \operatorname{cov}(X_1, X_p) & \operatorname{cov}(X_2, X_p) & \cdots & \operatorname{Var} X_p \end{pmatrix}$$

Principal component analysis (PCA)

- A crash course on probability
- **A running example: fingerprints**
- Change of basis
- Back to PCA
- A 2D example to begin: hockey players
- Back to fingerprints



Example of a PCA problem

We collect a bunch of information about a bunch of people.. for instance this data from Loughborough University

> *This dataset contains the height, weight and 4 fingerprint measurements (length, width, area and circumference), collected from 200 participants.*

What best describes a participant?

The variables

Each participant is associated to 11 variables

- "Participant Number"
- "Gender"
- "Age"
- "Dominant Hand"
- "Height (cm) (average of 3 measurements)"
- "Weight (kg) (average of 3 measurements)"
- "Fingertip Temperature (°C)"
- "Fingerprint Height (mm)"
- "Fingerprint Width (mm)"
- "Fingerprint Area (mm2)"
- "Fingerprint Circumference (mm)"

Nature of variables

Variables have different natures

- "Participant Number": $\in \mathbb{N}$ (not interesting)
- "Gender": categorical
- "Age": $\in \mathbb{N}$
- "Dominant Hand": categorical
- "Height (cm) (average of 3 measurements)": $\in \mathbb{R}$
- "Weight (kg) (average of 3 measurements)": $\in \mathbb{R}$
- "Fingertip Temperature (°C)": $\in \mathbb{R}$
- "Fingerprint Height (mm)": $\in \mathbb{R}$
- "Fingerprint Width (mm)": $\in \mathbb{R}$
- "Fingerprint Area (mm2)": $\in \mathbb{R}$
- "Fingerprint Circumference (mm)": $\in \mathbb{R}$

Setting things up

Each participant is a row in the matrix (an *observation*)

Each variable is a column

So we have an 200×10 matrix (we discard the "Participant number" column)

We want to find what carries the most information

For this, we are going to project the information in a new basis in which the first "dimension" will carry most variance, the second dimension will carry a little less, etc.

In order to do so, we need to learn how to change bases

Principal component analysis (PCA)

- A crash course on probability
- A running example: fingerprints
- **Change of basis**
- Back to PCA
- A 2D example to begin: hockey players
- Back to fingerprints

In the following slide,

$$[\boldsymbol{x}]_{\mathcal{B}}$$

denotes the coordinates of $\boldsymbol{x}$ in the basis $\mathcal{B}$

The aim of a change of basis is to express vectors in another coordinate system (another basis)

We do so by finding a matrix allowing to move from one basis to another

Change of basis

Definition 35 (Change of basis matrix)

$\mathcal{B} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$ and $\mathcal{C} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ bases of vector space V
The **change of basis matrix** $P_{\mathcal{C} \leftarrow \mathcal{B}} \in \mathcal{M}_n$,

$$P_{\mathcal{C} \leftarrow \mathcal{B}} = [[\boldsymbol{u}_1]_{\mathcal{C}} \cdots [\boldsymbol{u}_n]_{\mathcal{C}}]$$

has columns the coordinate vectors $[\boldsymbol{u}_1]_{\mathcal{C}}, \ldots, [\boldsymbol{u}_n]_{\mathcal{C}}$ of vectors in $\mathcal{B}$ with respect to $\mathcal{C}$

Theorem 36

$\mathcal{B} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$ and $\mathcal{C} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ bases of vector space V and $P_{\mathcal{C} \leftarrow \mathcal{B}}$ a change of basis matrix from $\mathcal{B}$ to $\mathcal{C}$

1. $\forall \boldsymbol{x} \in V$, $P_{\mathcal{C} \leftarrow \mathcal{B}}[\boldsymbol{x}]_{\mathcal{B}} = [\boldsymbol{x}]_{\mathcal{C}}$
2. $P_{\mathcal{C} \leftarrow \mathcal{B}}$ *s.t.* $\forall \boldsymbol{x} \in V$, $P_{\mathcal{C} \leftarrow \mathcal{B}}[\boldsymbol{x}]_{\mathcal{B}} = [\boldsymbol{x}]_{\mathcal{C}}$ *is* **unique**
3. $P_{\mathcal{C} \leftarrow \mathcal{B}}$ *invertible and* $P^{-1}_{\mathcal{C} \leftarrow \mathcal{B}} = P_{\mathcal{B} \leftarrow \mathcal{C}}$

Row-reduction method for changing bases

Theorem 37

$\mathcal{B} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$ and $\mathcal{C} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ bases of vector space V. Let $\mathcal{E}$ be any basis for V,

$$B = [[\boldsymbol{u}_1]_{\mathcal{E}}, \ldots, [\boldsymbol{u}_n]_{\mathcal{E}}] \text{ and } C = [[\boldsymbol{v}_1]_{\mathcal{E}}, \ldots, [\boldsymbol{v}_n]_{\mathcal{E}}]$$

and let $[C|B]$ be the augmented matrix constructed using C and B. Then

$$RREF([C|B]) = [\mathbb{I}|P_{\mathcal{C}\leftarrow\mathcal{B}}]$$

If working in $\mathbb{R}^n$, this is quite useful with $\mathcal{E}$ the standard basis of $\mathbb{R}^n$ (it does not matter if $\mathcal{B} = \mathcal{E}$)

So the question now becomes

How do we find what new basis to look at our data in?

(Changing the basis does not change the data, just the view you have of it)

(Think of what happens when you do a headstand.. your up becomes down, your right and left switch, but the world does not change, just your view of it)

(Changes of bases are *fundamental* operations in Science)

Principal component analysis (PCA)

- A crash course on probability
- A running example: fingerprints
- Change of basis
- **Back to PCA**
- A 2D example to begin: hockey players
- Back to fingerprints

Setting things up

I will use notation (mostly) as in Joliffe's *Principal Component Analysis* (PDF of older version available for free from UofM Libraries)

$\boldsymbol{x} = (x_1, \ldots, x_p)$ vector of p random variables

We seek a linear function $\boldsymbol{\alpha}_1^T \boldsymbol{x}$ with maximum variance, where $\boldsymbol{\alpha}_1 = (\alpha_{11}, \ldots, \alpha_{1p})$, i.e.,

$$\boldsymbol{\alpha}_1^T \boldsymbol{x} = \sum_{j=1}^{p} \alpha_{1j} x_j$$

Then we seek a linear function $\boldsymbol{\alpha}_2^T \boldsymbol{x}$ with maximum variance, uncorrelated to $\boldsymbol{\alpha}_1^T \boldsymbol{x}$

And we continue...

At kth stage, we find a linear function $\boldsymbol{\alpha}_k^T \boldsymbol{x}$ with maximum variance, uncorrelated to $\boldsymbol{\alpha}_1^T \boldsymbol{x}, \ldots, \boldsymbol{\alpha}_{k-1}^T \boldsymbol{x}$

$\boldsymbol{\alpha}_i^T \boldsymbol{x}$ is the ith **principal component** (PC)

Case of known covariance matrix

Suppose we know Σ, covariance matrix of $\boldsymbol{x}$ (i.e., typically: we know $\boldsymbol{x}$)

Then the kth PC is

$$z_k = \boldsymbol{\alpha}_k^T \boldsymbol{x}$$

where $\boldsymbol{\alpha}_k$ is an eigenvector of Σ corresponding to the kth largest eigenvalue λ_k

If, additionally, $\|\boldsymbol{\alpha}_k\| = \boldsymbol{\alpha}_k^T \boldsymbol{\alpha} = 1$, then $\lambda_k = \text{Var } z_k$

Why is that?

Let us start with

$$\boldsymbol{\alpha}_1^T \boldsymbol{x}$$

We want maximum variance, where $\boldsymbol{\alpha}_1 = (\alpha_{11}, \ldots, \alpha_{1p})$, i.e.,

$$\boldsymbol{\alpha}_1^T \boldsymbol{x} = \sum_{j=1}^{p} \alpha_{1j} x_j$$

with the constraint that $\|\boldsymbol{\alpha}_1\| = 1$

We have

$$\text{Var } \boldsymbol{\alpha}_1^T \boldsymbol{x} = \boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1$$

Objective

We want to maximise Var $\boldsymbol{\alpha}_1^T \boldsymbol{x}$, i.e.,

$$\boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1$$

under the constraint that $\|\boldsymbol{\alpha}_1\| = 1$

$\Longrightarrow$ use **Lagrange multipliers**

Maximisation using Lagrange multipliers

(A.k.a. super-brief intro to multivariable calculus)

We want the max of $f(x_1, \ldots, x_n)$ under the constraint $g(x_1, \ldots, x_n) = k$

1. Solve

$$\nabla f(x_1, \ldots, x_n) = \lambda \nabla g(x_1, \ldots, x_n)$$
$$g(x_1, \ldots, x_n) = k$$

   where $\nabla = (\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})$ is the **gradient operator**
2. Plug all solutions into $f(x_1, \ldots, x_n)$ and find maximum values (provided values exist and $\nabla g \neq \mathbf{0}$ there)

λ is the **Lagrange multiplier**

The gradient

(Continuing our super-brief intro to multivariable calculus)

$f : \mathbb{R}^n \to \mathbb{R}$ function of several variables, $\nabla = \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\right)$ the gradient operator

Then

$$\nabla f = \left(\frac{\partial}{\partial x_1} f, \ldots, \frac{\partial}{\partial x_n} f\right)$$

So ∇f is a *vector-valued* function, $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$; also written as

$$\nabla f = f_{x_1}(x_1, \ldots, x_n)\boldsymbol{e}_1 + \cdots f_{x_n}(x_1, \ldots, x_n)\boldsymbol{e}_n$$

where f_{x_i} is the partial derivative of f with respect to x_i and $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$ is the standard basis of $\mathbb{R}^n$

Bear with me..

(You may experience a brief period of discomfort)

$\boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1$ and $\|\boldsymbol{\alpha}_1\|^2 = \boldsymbol{\alpha}_1^T \boldsymbol{\alpha_1}$ are functions of $\boldsymbol{\alpha}_1 = (\alpha_{11}, \ldots, \alpha_{1p})$

In the notation of the previous slide, we want the max of

$$f(\alpha_{11}, \ldots, \alpha_{1p}) := \boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1$$

under the constraint that

$$g(\alpha_{11}, \ldots, \alpha_{1p}) := \boldsymbol{\alpha}_1^T \boldsymbol{\alpha_1} = 1$$

and with gradient operator

$$\nabla = \left(\frac{\partial}{\partial \alpha_{11}}, \ldots, \frac{\partial}{\partial \alpha_{1p}} \right)$$

Effect of ∇ on g

g is easiest to see:

$$\begin{aligned}
\nabla g(\alpha_{11}, \ldots, \alpha_{1p}) &= \left(\frac{\partial}{\partial \alpha_{11}}, \ldots, \frac{\partial}{\partial \alpha_{1p}} \right) (\alpha_{11}, \ldots, \alpha_{1p}) \begin{pmatrix} \alpha_{11} \\ \vdots \\ \alpha_{1p} \end{pmatrix} \\
&= \left(\frac{\partial}{\partial \alpha_{11}}, \ldots, \frac{\partial}{\partial \alpha_{1p}} \right) (\alpha_{11}^2 + \cdots + \alpha_{1p}^2) \\
&= (2\alpha_{11}, \ldots, 2\alpha_{1p}) \\
&= 2\boldsymbol{\alpha}_1
\end{aligned}$$

(And that's a general result: $\nabla \|\boldsymbol{x}\|_2^2 = 2\boldsymbol{x}$ with $\|\cdot\|_2$ the Euclidean norm)

Effect of ∇ on f

Expand (write $\Sigma = [s_{ij}]$ and do not exploit symmetry)

$$\boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1 = (\alpha_{11}, \ldots, \alpha_{1p}) \begin{pmatrix} s_{11} & s_{12} & \cdots & s_{1p} \\ s_{21} & s_{22} & \cdots & s_{2p} \\ \vdots & \vdots & & \vdots \\ s_{p1} & s_{p2} & & s_{pp} \end{pmatrix} \begin{pmatrix} \alpha_{11} \\ \alpha_{12} \\ \vdots \\ \alpha_{1p} \end{pmatrix}$$

$$= (\alpha_{11}, \ldots, \alpha_{1p}) \begin{pmatrix} s_{11}\alpha_{11} + s_{12}\alpha_{12} + \cdots + s_{1p}\alpha_{1p} \\ s_{21}\alpha_{11} + s_{22}\alpha_{12} + \cdots + s_{2p}\alpha_{1p} \\ \vdots \\ s_{p1}\alpha_{11} + s_{p2}\alpha_{12} + \cdots + s_{pp}\alpha_{1p} \end{pmatrix}$$

$$= (s_{11}\alpha_{11} + s_{12}\alpha_{12} + \cdots + s_{1p}\alpha_{1p})\alpha_{11}$$
$$+ (s_{21}\alpha_{11} + s_{22}\alpha_{12} + \cdots + s_{2p}\alpha_{1p})\alpha_{12}$$
$$\vdots$$
$$+ (s_{p1}\alpha_{11} + s_{p2}\alpha_{12} + \cdots + s_{pp}\alpha_{1p})\alpha_{1p}$$

We have

$$\begin{aligned}\boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1 = &(s_{11}\alpha_{11} + s_{12}\alpha_{12} + \cdots + s_{1p}\alpha_{1p})\alpha_{11} \\ &+ (s_{21}\alpha_{11} + s_{22}\alpha_{12} + \cdots + s_{2p}\alpha_{1p})\alpha_{12} \\ &\vdots \\ &+ (s_{p1}\alpha_{11} + s_{p2}\alpha_{12} + \cdots + s_{pp}\alpha_{1p})\alpha_{1p}\end{aligned}$$

$$\begin{aligned}\implies \frac{\partial}{\partial \alpha_{11}} \boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1 &= (s_{11}\alpha_{11} + s_{12}\alpha_{12} + \cdots + s_{1p}\alpha_{1p}) + s_{11}\alpha_{11} \\ &\quad + s_{21}\alpha_{12} + \cdots + s_{p1}\alpha_{1p} \\ &= s_{11}\alpha_{11} + s_{12}\alpha_{12} + \cdots + s_{1p}\alpha_{1p} \\ &\quad + s_{11}\alpha_{11} + s_{21}\alpha_{12} + \cdots + s_{p1}\alpha_{1p} \\ &= 2(s_{11}\alpha_{11} + s_{12}\alpha_{12} + \cdots + s_{1p}\alpha_{1p})\end{aligned}$$

(last equality stems from symmetry of Σ)

In general, for $i = 1, \ldots, p$,

$$\begin{aligned}\frac{\partial}{\partial \alpha_{1i}} \boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1 &= s_{i1}\alpha_{11} + s_{i2}\alpha_{12} + \cdots + s_{ip}\alpha_{1p} \\ &\quad + s_{i1}\alpha_{11} + s_{2i}\alpha_{12} + \cdots + s_{pi}\alpha_{1p} \\ &= 2(s_{i1}\alpha_{11} + s_{i2}\alpha_{12} + \cdots + s_{ip}\alpha_{1p})\end{aligned}$$

(because of symmetry of Σ)

As a consequence,

$$\nabla \boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1 = 2\Sigma \boldsymbol{\alpha}_1$$

So solving

$$\nabla f(x_1, \ldots, x_n) = \lambda \nabla g(x_1, \ldots, x_n)$$

means solving

$$2\Sigma\boldsymbol{\alpha}_1 = \lambda 2\boldsymbol{\alpha}_1$$

i.e.,

$$\Sigma\boldsymbol{\alpha}_1 = \lambda\boldsymbol{\alpha}_1$$

$\implies$ $(\lambda, \boldsymbol{\alpha}_1)$ eigenpair of Σ, with $\boldsymbol{\alpha}_1$ having unit length

Picking the right eigenvalue

$(\lambda, \boldsymbol{\alpha}_1)$ eigenpair of Σ, with $\boldsymbol{\alpha}_1$ having unit length

But which λ to choose?

Recall that we want Var $\boldsymbol{\alpha}_1^T \boldsymbol{x} = \boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1$ maximal

We have

$$\text{Var } \boldsymbol{\alpha}_1^T \boldsymbol{x} = \boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_1 = \boldsymbol{\alpha}_1^T (\Sigma \boldsymbol{\alpha}_1) = \boldsymbol{\alpha}_1^T (\lambda \boldsymbol{\alpha}_1) = \lambda (\boldsymbol{\alpha}_1^T \boldsymbol{\alpha}_1) = \lambda$$

$\implies$ we pick $\lambda = \lambda_1$, the largest eigenvalue (covariance matrix symmetric so eigenvalues real)

What we have this far..

The first principal component is $\boldsymbol{\alpha}_1^T \boldsymbol{x}$ and has variance λ_1, where λ_1 the largest eigenvalue of Σ and $\boldsymbol{\alpha}_1$ an associated eigenvector with $\|\boldsymbol{\alpha}_1\| = 1$

We want the second principal component to be *uncorrelated* with $\boldsymbol{\alpha}_1^T \boldsymbol{x}$ and to have maximum variance Var $\boldsymbol{\alpha}_2^T \boldsymbol{x} = \boldsymbol{\alpha}_2^T \Sigma \boldsymbol{\alpha}_2$, under the constraint that $\|\boldsymbol{\alpha}_2\| = 1$

$\boldsymbol{\alpha}_2^T \boldsymbol{x}$ uncorrelated to $\boldsymbol{\alpha}_1^T \boldsymbol{x}$ if $\text{cov}(\boldsymbol{\alpha}_1^T \boldsymbol{x}, \boldsymbol{\alpha}_2^T \boldsymbol{x}) = 0$

We have

$$\begin{aligned}
\operatorname{cov}(\boldsymbol{\alpha}_1^T \boldsymbol{x}, \boldsymbol{\alpha}_2^T \boldsymbol{x}) &= \boldsymbol{\alpha}_1^T \Sigma \boldsymbol{\alpha}_2 \\
&= \boldsymbol{\alpha}_2^T \Sigma^T \boldsymbol{\alpha}_1 \\
&= \boldsymbol{\alpha}_2^T \Sigma \boldsymbol{\alpha}_1 \quad [\Sigma \text{ symmetric}] \\
&= \boldsymbol{\alpha}_2^T (\lambda_1 \boldsymbol{\alpha}_1) \\
&= \lambda \boldsymbol{\alpha}_2^T \boldsymbol{\alpha}_1
\end{aligned}$$

So $\boldsymbol{\alpha}_2^T \boldsymbol{x}$ uncorrelated to $\boldsymbol{\alpha}_1^T \boldsymbol{x}$ if $\boldsymbol{\alpha}_1 \perp \boldsymbol{\alpha}_2$

This is beginning to sound a lot like Gram-Schmidt, no?

In short

Take whatever covariance matrix is available to you (known Σ or sample S_X) – assume sample from now on for simplicity

For $i = 1, \ldots, p$, the ith principal component is

$$z_i = \boldsymbol{v}_i^T \boldsymbol{x}$$

where $\boldsymbol{v}_i$ eigenvector of S_X associated to the ith largest eigenvalue λ_i

If $\boldsymbol{v}_i$ is normalised, then $\lambda_i = \text{Var } z_k$

Covariance matrix

Σ the covariance matrix of the random variable, S_X the sample covariance matrix

$X \in \mathcal{M}_{mp}$ the data, then the (sample) covariance matrix S_X takes the form

$$S_X = \frac{1}{n-1} X^T X$$

where the data is centred!

Sometimes you will see $S_X = 1/(n-1) X X^T$. This is for matrices with observations in columns and variables in rows. Just remember that you want the covariance matrix to have size the number of variables, not observations, this will give you the order in which to take the product

Principal component analysis (PCA)

- A crash course on probability
- A running example: fingerprints
- Change of basis
- Back to PCA
- **A 2D example to begin: hockey players**
- Back to fingerprints

A 2D example

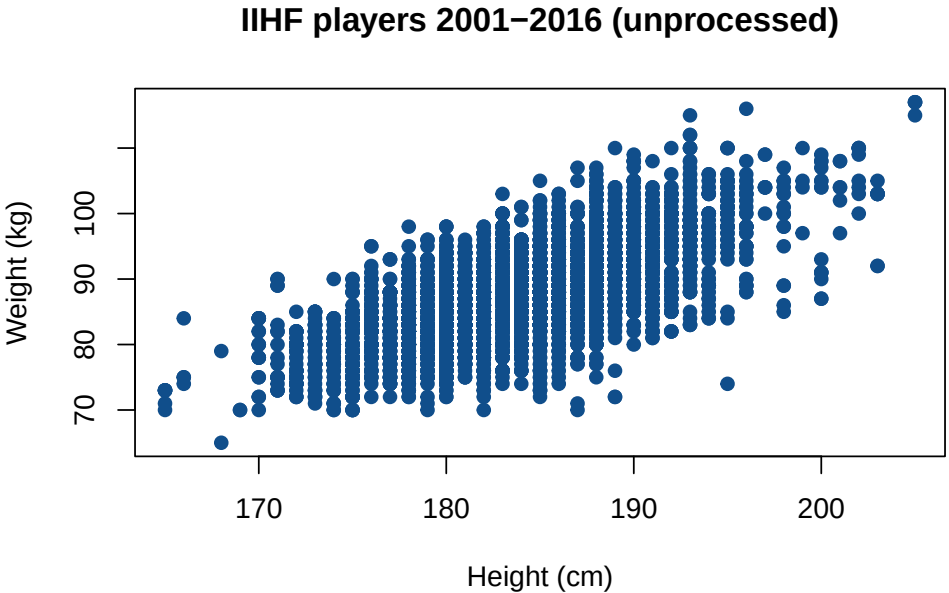
See a dataset on this page for a dataset of height and weight of some hockey players

```
data = read.csv("https://figshare.com/ndownloader/files/5303173")
head(data, n=3)

##   year country no                name position side height weight      b
## 1 2001     RUS 10    tverdovsky oleg         D    L    185     84 1976-0
## 2 2001     RUS  2  vichnevsky vitali         D    L    188     86 1980-0
## 3 2001     RUS 26 petrochinin evgeni         D    L    182     95 1976-0
##                      club      age cohort      bmi
## 1   anaheim mighty ducks  24.95277   1976 24.54346
## 2   anaheim mighty ducks  21.11978   1980 24.33228
## 3 severstal cherepovetal  25.22930   1976 28.68011

dim(data)

## [1] 6292   13
```

In case you are wondering, this is a database of ice hockey players at IIHF world championships, 2001-2016, assembled by the dataset's author

See some comments here

As usual, it is a good idea to plot this to get a sense of the lay of the land


IIHF players 2001−2016 (unprocessed)
Weight (kg)
70
80
90
100
170
180
190
200
Height (cm)

The author of the study is interested in the evolution of weights, so it is likely that the same person will be in the dataset several times

Let us check this: first check will be `FALSE` if the number of unique names does not match the number of rows in the dataset

```
length(unique(data$name)) == dim(data)[1]

## [1] FALSE

length(unique(data$name))

## [1] 3278
```

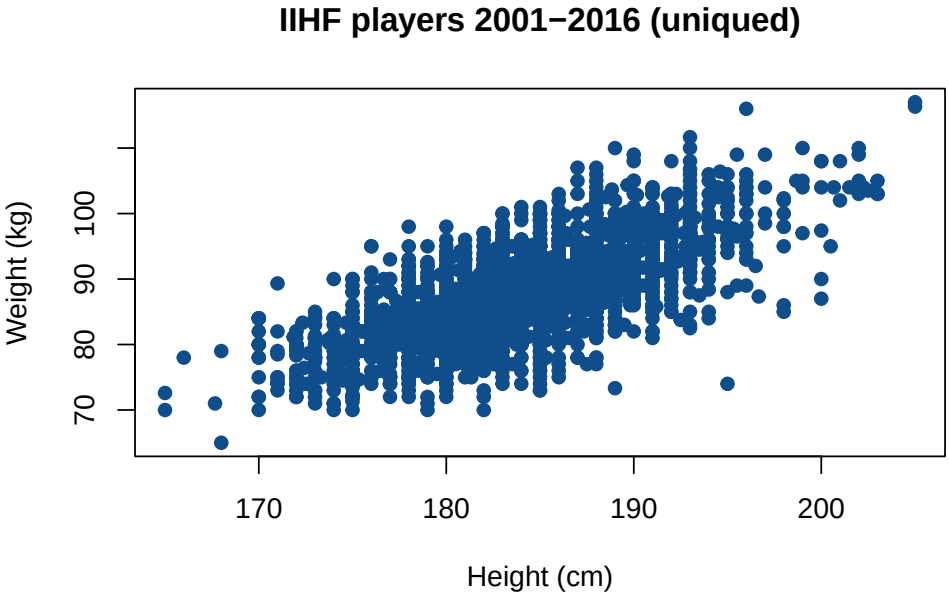
Not interested in the evolution of weights, so simplify: if more than one record for someone, take average of recorded weights and heights

To be extra careful, could check as well that there are no major variations on player height (homonymies?)

```
data_simplified = data.frame(name = unique(data$name))
w = c()
h = c()
for (n in data_simplified$name) {
    tmp = data[which(data$name == n),]
    h = c(h, mean(tmp$height))
    w = c(w, mean(tmp$weight))
}
data_simplified$weight = w
data_simplified$height = h
```

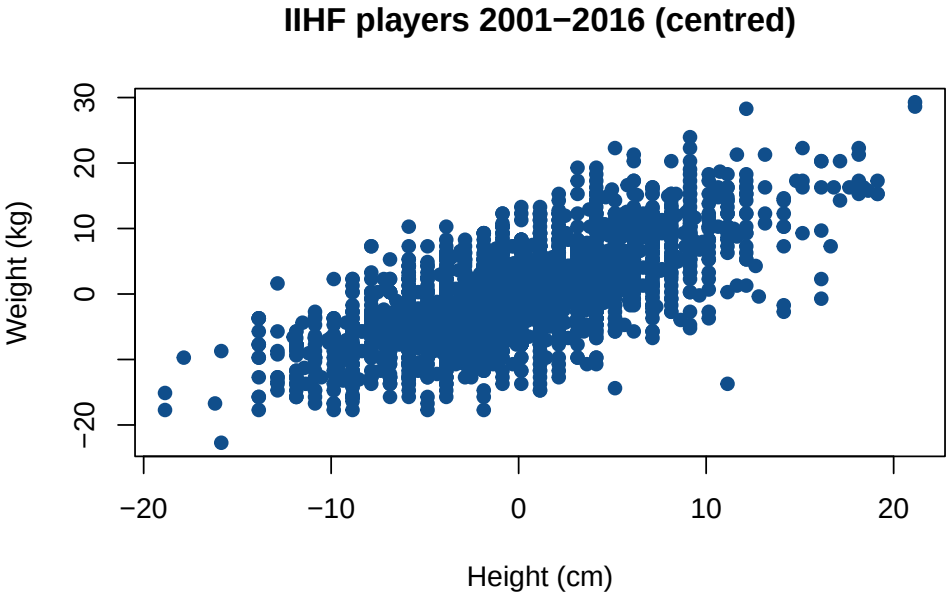
```r
data = data_simplified
head(data_simplified, n = 6)
```

```
##                   name weight height
## 1    tverdovsky oleg    84.0  185.0
## 2  vichnevsky vitali    86.0  188.0
## 3 petrochinin evgeni    95.0  182.0
## 4    zhdan alexander    85.5  178.5
## 5    orekhovsky oleg    88.0  175.0
## 6      zhukov sergei    92.5  193.0
```


IIHF players 2001−2016 (uniqued)
Weight (kg)
70
80
90
100
170
180
190
200
Height (cm)

Centre the data

```
mean(data$weight)

## [1] 87.71555

mean(data$height)

## [1] 183.8596

data$weight.c = data$weight-mean(data$weight)
data$height.c = data$height-mean(data$height)
```


IIHF players 2001−2016 (centred)
Weight (kg)
30
20
10
0
−20
−20
−10
0
10
20
Height (cm)

Covariance

The function cov returns the covariance of two samples

Note that the functions deals equally well with data that is not centred as with data that is centred

```
cov(data$height, data$weight)

## [1] 26.63506

cov(data$height.c, data$weight.c)

## [1] 26.63506
```

Covariance matrix

As we could see from plotting the data, there is a positive linear relationship between the two variables

Let us compute the sample covariance matrix

```
X = as.matrix(data[,c("height.c", "weight.c")])
S = 1/(dim(X)[1]-1)*t(X) %*% X
S

##           height.c weight.c
## height.c 29.66176 26.63506
## weight.c 26.63506 47.81112
```

Covariance matrix

The off-diagonal entries do match the computed covariance. Let us check that the variances are indeed a match too.

```
var(X[,1])

## [1] 29.66176

var(X[,2])

## [1] 47.81112
```

Hey, that works. Is math not cool? ;)

Principal components

Now compute the principal components. We need eigenvalues and eigenvectors

```
ev = eigen(S)
ev
```

```
## eigen() decomposition
## $values
## [1] 66.87496 10.59793
##
## $vectors
##           [,1]       [,2]
## [1,] 0.5820222 -0.8131729
## [2,] 0.8131729  0.5820222
```

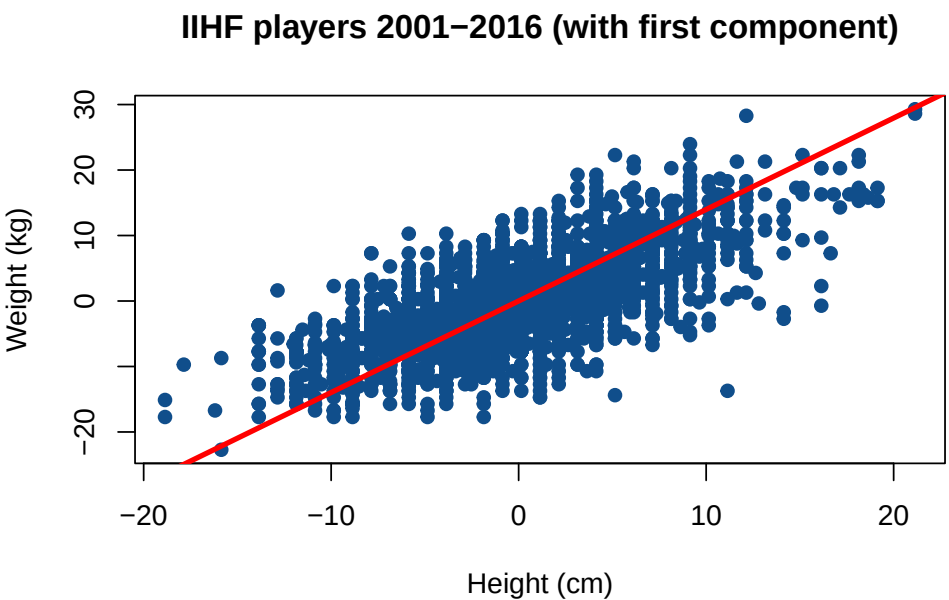
(`eigen` returns eigenvalues sorted in decreasing order and normalised eigenvectors)

First principal component

Let us plot this first eigenvector (well, the line carrying this first eigenvector)

To use the function `abline`, we need to give the coefficients of the line in the form of (intercept,slope). Intercept is easy, as the line goes through the origin (by construction and because we have centred the data). The slope is also quite simple..

```
plot(data$height.c, data$weight.c,
    pch = 19, col = "dodgerblue4",
    main = "IIHF players 2001-2016 (with first component)",
    xlab = "Height (cm)", ylab = "Weight (kg)")
abline(a = 0, b = ev$vectors[2,1]/ev$vectors[1,1],
       col = "red", lwd = 3)
```


IIHF players 2001−2016 (with first component)
Weight (kg)
−20
0
10
20
30
−20
−10
0
10
20
Height (cm)

Rotating the data

Let us rotate the data so that the red line becomes the x-axis

To do that, we use a rotation matrix

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

To find the angle θ, recall that $\tan\theta$ is equal to opposite length over adjacent length, i.e.,

$$\tan\theta = \frac{\texttt{ev\$vectors}[2,1]}{\texttt{ev\$vectors}[1,1]}$$

So we just use the arctan of this

Note that angles are in radians

Rotating the data

```
theta = atan(ev$vectors[2,1]/ev$vectors[1,1])
theta
```

```
## [1] 0.949583
```

```
R_theta = matrix(c(cos(theta), -sin(theta),
                   sin(theta), cos(theta)),
                 nr = 2, byrow = TRUE)
R_theta
```

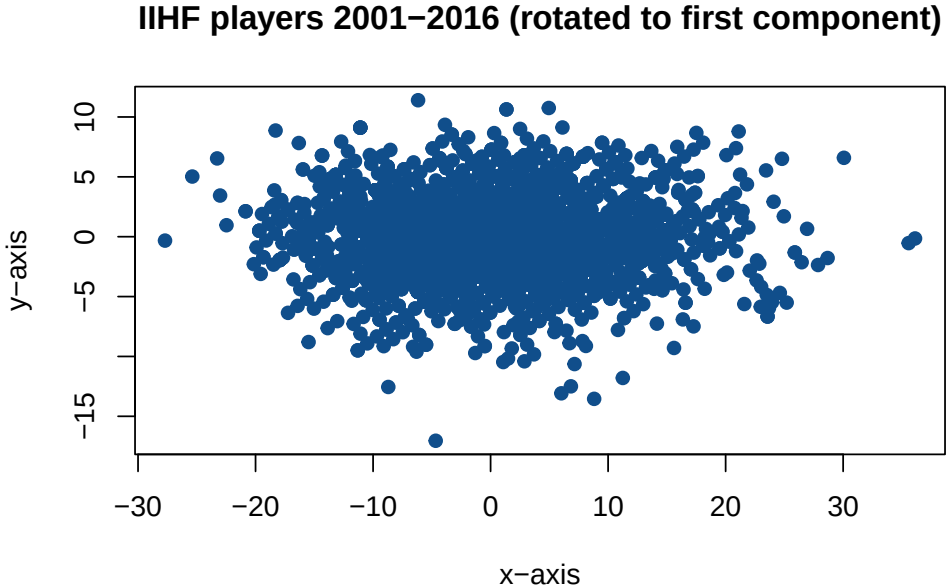
```
##           [,1]       [,2]
## [1,] 0.5820222 -0.8131729
## [2,] 0.8131729  0.5820222
```

Rotating the data

And now we rotate the points

(In this case, we think of the points as vectors, of course)

```
tmp_in = matrix(c(data$weight.c, data$height.c),
                nc = 2)
tmp_out = c()
for (i in 1:dim(tmp_in)[1]) {
    tmp_out = rbind(tmp_out,
                    t(R_theta %*% tmp_in[i,]))
}
data$weight.c_r = tmp_out[,1]
data$height.c_r = tmp_out[,2]
```

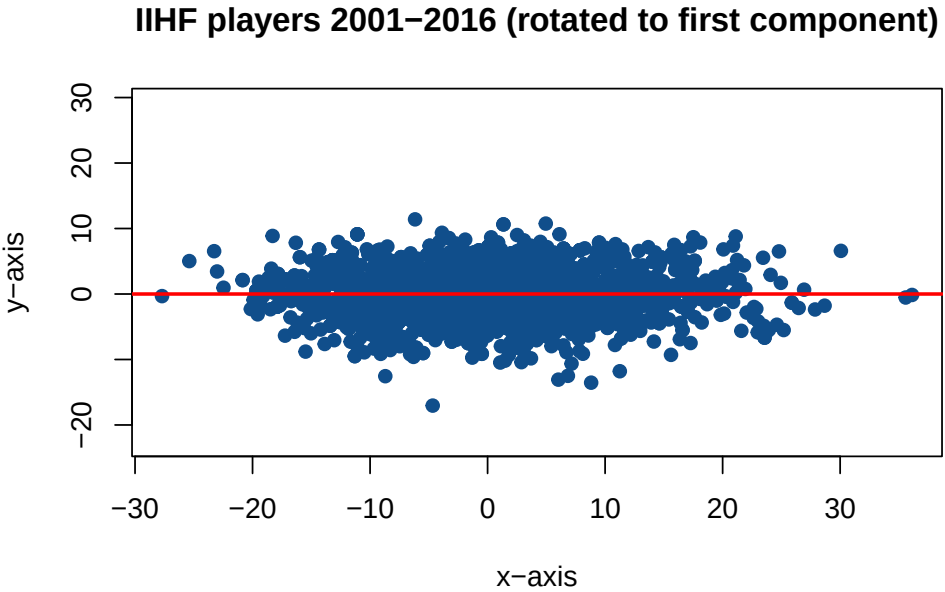

IIHF players 2001−2016 (rotated to first component)
y−axis
10
5
0
−5
−15
−30
−20
−10
0
10
20
30
x−axis

Principal components

Note that the axes have changed quite a lot, hence the very different aspect

Let us plot with the same range as for the non-rotated data for the y-axis

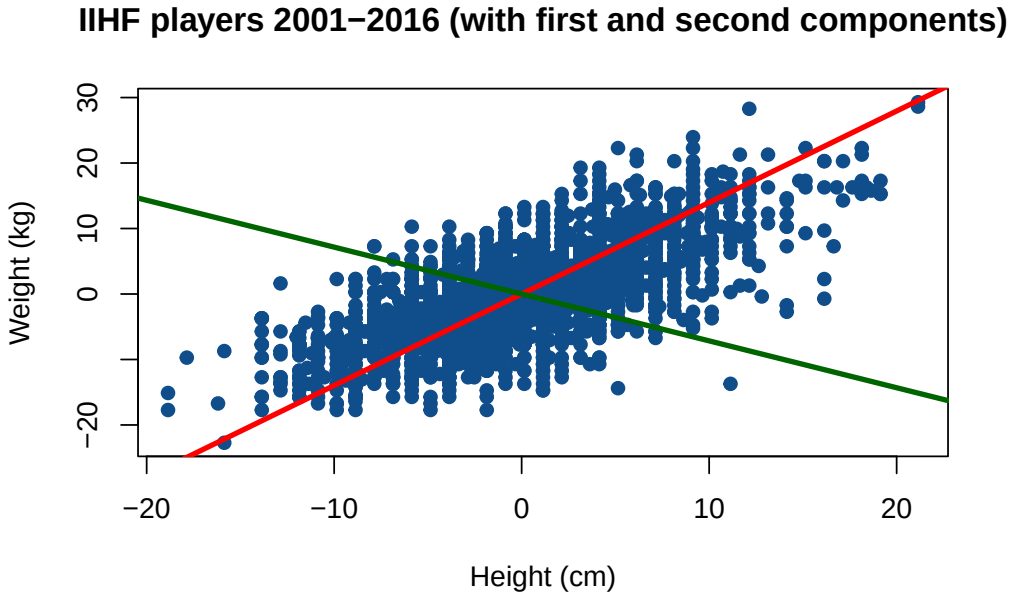
```
plot(data$height.c_r, data$weight.c_r,
    pch = 19, col = "dodgerblue4",
    xlab = "x-axis", ylab = "y-axis",
    main = "IIHF players 2001-2016 (rotated to first component)",
    ylim = range(data$weight.c))
abline(h = 0, col = "red", lwd = 2)
```


IIHF players 2001−2016 (rotated to first component)
y−axis
30
20
10
0
−20
−30
−20
−10
0
10
20
30
x−axis

First and second principal components

Plot the first and second eigenvectors

```
plot(data$height.c, data$weight.c,
    pch = 19, col = "dodgerblue4",
    main = "IIHF players 2001-2016 (with first and second components)",
    xlab = "Height (cm)", ylab = "Weight (kg)")
abline(a = 0, b = ev$vectors[2,1]/ev$vectors[1,1],
       col = "red", lwd = 3)
abline(a = 0, b = ev$vectors[2,2]/ev$vectors[1,2],
       col = "darkgreen", lwd = 3)
```


IIHF players 2001−2016 (with first and second components)
Weight (kg)
30
20
10
0
−20
−20
−10
0
10
20
Height (cm)

Proper change of basis

Let us change the basis so that, in the new basis, the first component is the x-axis and the second component is the y-axis

We want to use Theorem 37

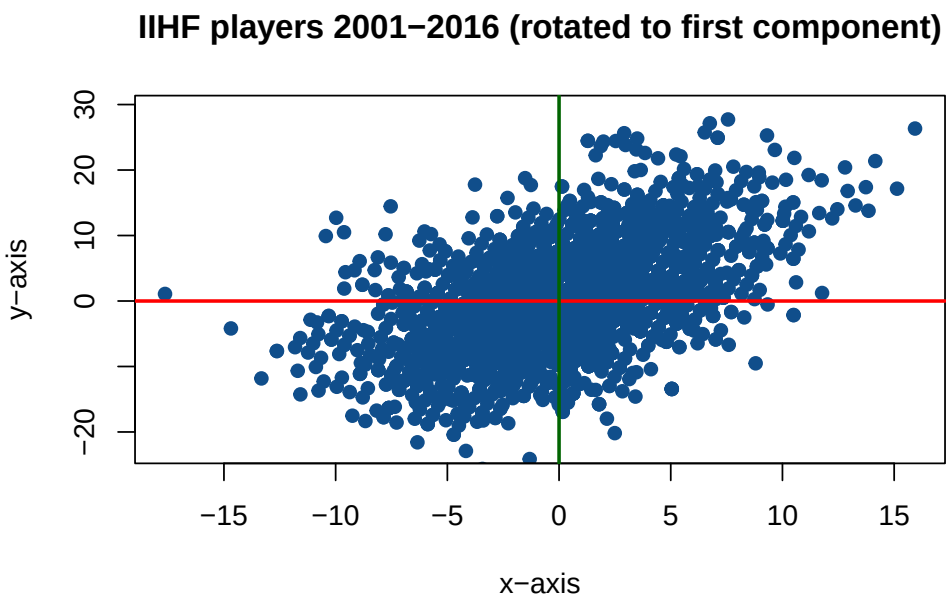
We need the coordinates of the new basis in the canonical basis of $\mathbb{R}^2$

Since both axes go through the origin, we can just use $y = ax$, with a the slope of the lines and, say, $x = 1$, i.e., $(x, y) = (1, a)$

We then normalise the resulting vectors

Proper change of basis

```
red_line = c(1, ev$vectors[2,1]/ev$vectors[1,1])
red_line = red_line/sqrt(sum(red_line^2))
green_line = c(1, ev$vectors[2,2]/ev$vectors[1,2])
green_line = green_line/sqrt(sum(green_line^2))
augmented_M = cbind(red_line,green_line, diag(2))
P = rref(augmented_M)[,3:4]

tmp_in = matrix(c(data$weight.c, data$height.c), nc = 2)
tmp_out = c()
for (i in 1:dim(tmp_in)[1]) {
    tmp_out = rbind(tmp_out, t(P %*% tmp_in[i,]))
}
data$weight.c_r2 = tmp_out[,1]
data$height.c_r2 = tmp_out[,2]
```


IIHF players 2001−2016 (rotated to first component)
y−axis
30
20
10
0
−20
−15
−10
−5
0
5
10
15
x−axis

PCA using built-in functions

Now do things "properly"

```
GS = pracma::gramSchmidt(A = ev$vectors, tol = 1e-10)
GS

## $Q
##           [,1]       [,2]
## [1,] 0.5820222 -0.8131729
## [2,] 0.8131729  0.5820222
##
## $R
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

PCA using built-in functions

Now recall we saw a theorem that told us how to construct a new basis..

```
A=matrix(c(GS$Q,1,0,0,1), nr = 2)
A
```

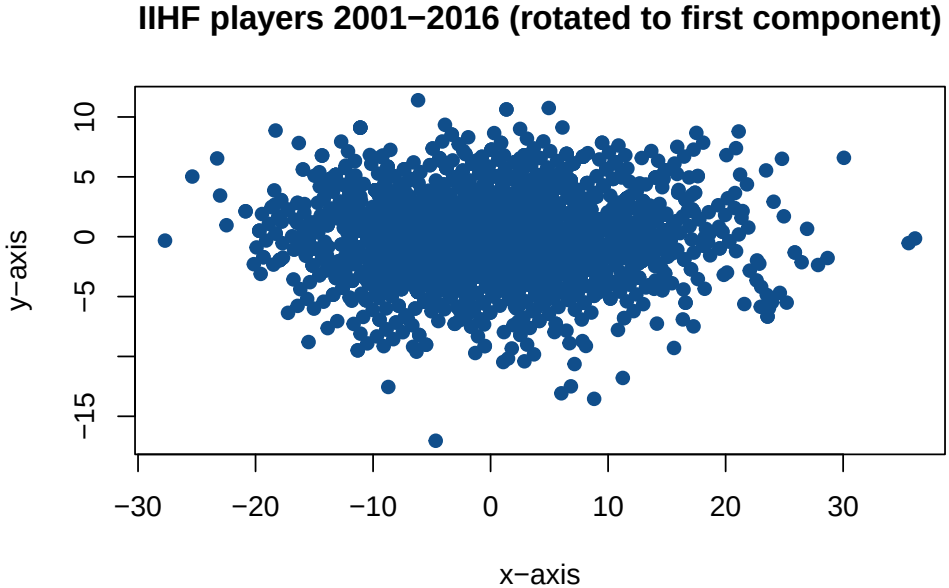
```
##           [,1]       [,2] [,3] [,4]
## [1,] 0.5820222 -0.8131729    1    0
## [2,] 0.8131729  0.5820222    0    1
```

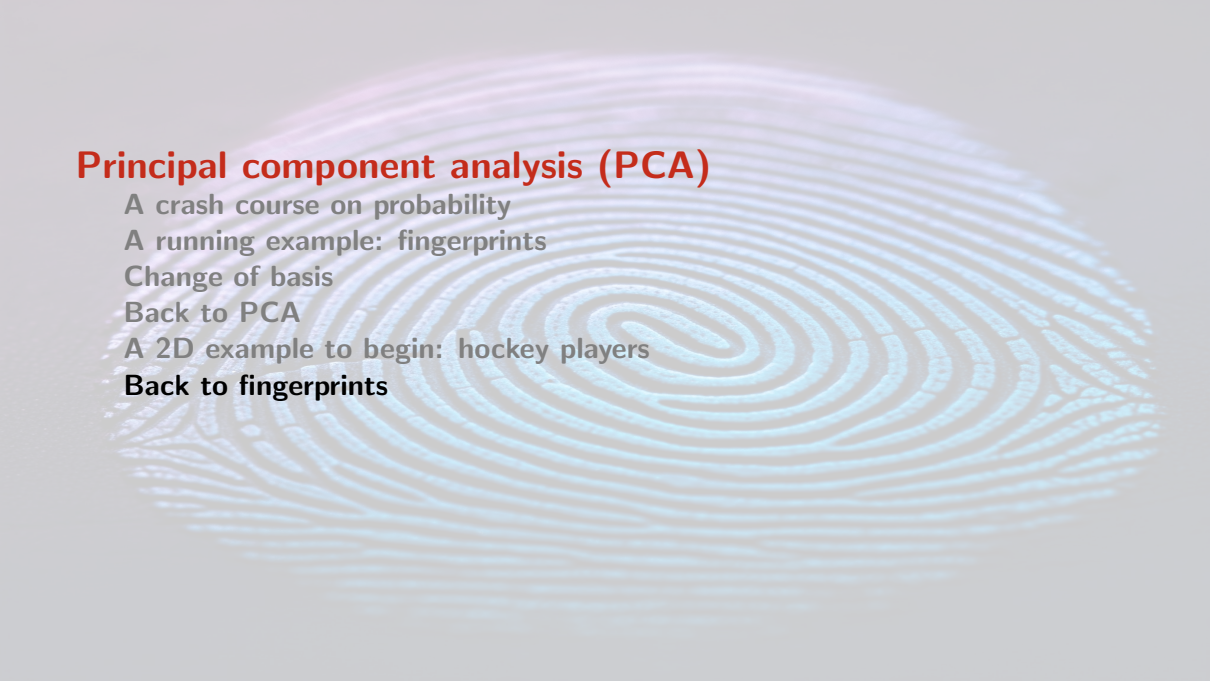
```
pracma::rref(A)
```

```
##      [,1] [,2]       [,3]      [,4]
## [1,]    1    0  0.5820222 0.8131729
## [2,]    0    1 -0.8131729 0.5820222
```

PCA using built-in functions

```
P = pracma::rref(A)[,c(3,4)]

##             [,1]      [,2]
## [1,]  0.5820222 0.8131729
## [2,] -0.8131729 0.5820222

X.new = X %*% t(P)
```


IIHF players 2001−2016 (rotated to first component)
y−axis
10
5
0
−5
−15
−30
−20
−10
0
10
20
30
x−axis



Principal component analysis (PCA)

- A crash course on probability
- A running example: fingerprints
- Change of basis
- Back to PCA
- A 2D example to begin: hockey players
- **Back to fingerprints**

We get the data from here

This time, we first download the data, then open the file

The file is an excel table, so we need to use a library for doing that

Loading the excel fingerprint data

```
download.file(url = "https://repository.lboro.ac.uk/ndownloader/files/14015
              destfile = "../CODE/fingerprint_data.xlsx")
data = openxlsx::read.xlsx("../CODE/fingerprint_data.xlsx")
head(data, n=3)

##   Participant.Number Gender Age Dominant.Hand
## 1                101   Male  NA         Right
## 2                102   Male  NA         Right
## 3                103   Male  NA         Right
##   Height.(cm).(average.of.3.measurments)
## 1                                174.0000
## 2                                202.0000
## 3                                182.3333
##   Weight.(kg).(average.of.3.measurements) Fingertip.Temperature.(°C)
## 1                                      70                         34
## 2                                      99                         30
## 3                                      82                         29
```

Some wrangling

Let us rework the names of columns a bit, for convenience. Let us also get rid of a few columns we are not using

```
data = data[,2:dim(data)[2]]
colnames(data) = c("gender", "age", "handedness", "height", "weight",
                   "fing_temp", "fing_height", "fing_width",
                   "fing_area", "fing_circ")
head(data, n=3)

##   gender age handedness   height weight fing_temp fing_height fing_widt
## 1   Male  NA      Right 174.0000     70        34        19.8       13.
## 2   Male  NA      Right 202.0000     99        30        24.0       14.
## 3   Male  NA      Right 182.3333     82        29        20.0       13.
##   fing_area fing_circ
## 1     240.6      57.7
## 2     278.8      62.7
## 3     223.8      55.5
```

Some wrangling – Centering

Plotting all these variables is complicated, so we forgo this for the time being

Let us centre the data. That there are some `NA` values, so we remove them using the function `complete.cases`, which identifies rows where at least one of the variables is `NA`

(We could also use `na.rm = TRUE` when taking the average to remove these values.)

We make new columns with the prefix `.c`, just to still have the initial data handy if need be.

Some wrangling – Centering

```
data = data[complete.cases(data),]
to_centre = c("age", "height",
              "weight", "fing_temp",
              "fing_height", "fing_width",
              "fing_area", "fing_circ")
for (c in to_centre) {
    new_c = sprintf("%s.c", c)
    data[[new_c]] = data[[c]] - mean(data[[c]], na.rm = TRUE)
}
head(data)
```

```
##    gender age handedness   height   weight fing_temp fing_height fing_wi
## 5    Male  18      Right 180.6667 80.33333        29        22.7       
## 6    Male  20      Right 180.0000 59.00000        32        24.3       
## 12   Male  18      Right 180.6667 68.00000        27        21.1       
## 23   Male  18      Right 188.6667 73.00000        29        23.0       
## 24 Female  19      Right 166.0000 65.00000        27        18.4
```

Covariance matrix

```
X = as.matrix(data[, to_centre])
S = 1/(dim(X)[1]-1)*t(X) %*% X
S
```

```
##                   age    height    weight fing_temp fing_height fing_wi
## age          478.9939  3692.562  1579.172  614.2393    435.6270   290.1
## height      3692.5624 30183.706 12766.541 4994.7669   3553.3620  2364.9
## weight      1579.1718 12766.541  5598.505 2118.0613   1505.3519  1006.2
## fing_temp    614.2393  4994.767  2118.061  840.9141    588.5436   392.2
## fing_height  435.6270  3553.362  1505.352  588.5436    420.9758   279.4
## fing_width   290.1264  2364.938  1006.262  392.2006    279.4357   186.8
## fing_area   5013.1387 41006.579 17504.745 6781.8000   4875.6983  3249.2
## fing_circ   1210.8221  9882.954  4190.940 1636.9264   1169.4310   778.0
##             fing_area  fing_circ
## age          5013.139  1210.8221
## height      41006.579  9882.9540
## weight      17504.745  4190.9405
```

Eigenvalues

```
ev = eigen(S)
ev$values
```

```
## [1] 9.730867e+04 4.982986e+02 1.669174e+02 2.565635e+01 1.349931e+01
## [6] 2.008242e+00 6.457803e-01 1.559819e-01
```

Let us add the singular values to ev

```
ev$sing_values = sqrt(ev$values)
```

Use built-in functions

```
GS = pracma::gramSchmidt(A = ev$vectors)
GS$Q
```

```
##              [,1]        [,2]         [,3]         [,4]        [,5]
## [1,] -0.06785926 -0.10703810 -0.074771554  0.959199870 -0.21608682  0.1
## [2,] -0.55435636 -0.72869902  0.290626700 -0.152379696 -0.15804372  0.1
## [3,] -0.23625691 -0.18581742 -0.946363825 -0.106759855 -0.00455192 -0.0
## [4,] -0.09173610 -0.13238200  0.016673068  0.178396675  0.95749120  0.1
## [5,] -0.06566056 -0.02693052  0.041586794  0.053196632  0.02950635 -0.4
## [6,] -0.04374434 -0.01430565  0.004070984  0.027623399  0.04800455  0.1
## [7,] -0.76437249  0.63185800  0.060301062  0.007078515 -0.00675132  0.1
## [8,] -0.18265026 -0.07289060  0.093169759  0.099246954  0.09109287 -0.8
##              [,7]         [,8]
## [1,] -0.030350587 -0.023178128
## [2,] -0.058350355 -0.012688619
## [3,]  0.002219217  0.001694049
## [4,] -0.077869385 -0.034295121
```

Some wrangling

Now recall we saw a theorem that told us how to construct a new basis..

```
# Make an identity matrix
Id = diag(dim(GS$Q)[1])
# Make the augmented matrix
A = cbind(GS$Q, Id)
# Compute the RREF and extract the relevant matrix
P = pracma::rref(A)[,(dim(GS$Q)[2]+1):dim(A)[2]]
X.new = X %*% t(P)
```

Use built-in functions

Use the built in function `prcomp` or `PCA` from the `FactoMineR` package

```
# data.pca = prcomp(X, center = TRUE, scale = TRUE)
data.pca = PCA(X, scale.unit = TRUE, graph = FALSE)
summary(data.pca)

##
## Call:
## PCA(X = X, scale.unit = TRUE, graph = FALSE)
##
##
## Eigenvalues
##                        Dim.1   Dim.2   Dim.3   Dim.4   Dim.5   Dim.6   I
## Variance               4.399   1.197   0.991   0.743   0.380   0.258   (
## % of var.             54.988  14.966  12.386   9.292   4.750   3.225   (
## Cumulative % of var.  54.988  69.955  82.341  91.632  96.383  99.608  9
##                        Dim.8
## Variance               0.006
```

Percentage of variance

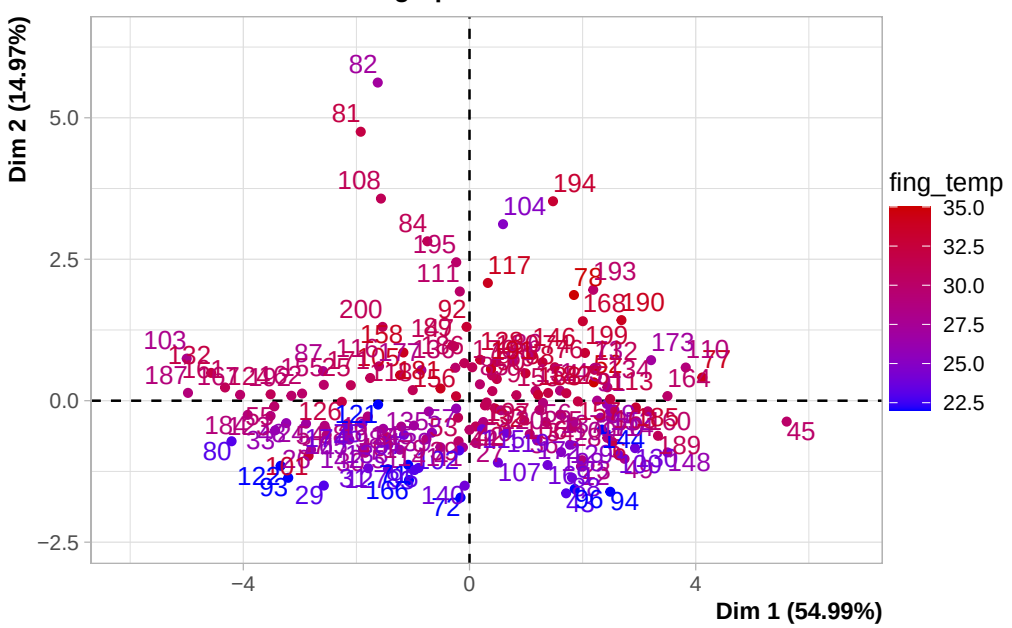
The "proportion of variance" (or "percentage of variance") information is actually the proportion (and then cumulative proportion) represented by the singular value associated to each principal component

We check this (approximately) by comparing with the singular values we computed

```
ev$sing_values/(sum(ev$sing_values))

## [1] 0.870036273 0.062259612 0.036033997 0.014127294 0.010247489 0.00395
## [7] 0.002241321 0.001101536

cumsum(ev$sing_values)/(sum(ev$sing_values))

## [1] 0.8700363 0.9322959 0.9683299 0.9824572 0.9927047 0.9966571 0.99889
## [8] 1.0000000
```

Plot results

```
plot.PCA(data.pca, axes = c(1,2), choix = "ind", habillage = 4)
```


PCA graph of individuals
Dim 2 (14.97%)
Dim 1 (54.99%)
5.0
2.5
0.0
−2.5
−4
0
4
fing_temp
35.0
32.5
30.0
27.5
25.0
22.5

Least squares problems

QR factorisation

Singular values decomposition (SVD)

Principal component analysis (PCA)

Support vector machines

Support vector machines

Clustering and classification

Support vector machines (SVM)

Clustering vs classification

Clustering is partitioning an unlabelled dataset into groups of similar objects

Classification sorts data into specific categories using a labelled dataset

Clustering

From Wikipedia

> **Cluster analysis** *or* **clustering** *is the task of grouping a set of objects in such a way that objects in the same group (called a* **cluster***) are more similar (in some sense) to each other than to those in other groups (clusters).*

There are a myriad of ways to do clustering, this is an extremely active field of research and application. See the Wikipedia page for leads

Classification

From Wikipedia

> *In statistics,* **classification** *is the problem of identifying which of a set of categories (sub-populations) an observation (or observations) belongs to. Examples are assigning a given email to the "spam" or "non-spam" class, and assigning a diagnosis to a given patient based on observed characteristics of the patient (sex, blood pressure, presence or absence of certain symptoms, etc.).*

Support vector machines (SVM)

Support vector machines (SVM)

We are given a training dataset of n points of the form

$$(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$$

where $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i = \{-1, 1\}$. The value of y_i indicates the class to which the point $\boldsymbol{x}_i$ belongs

We want to find a **surface** $\mathcal{S}$ in $\mathbb{R}^p$ that divides the group of points into two subgroups

Once we have this surface $\mathcal{S}$, any additional point that is added to the set can then be *classified* as belonging to either one of the sets depending on where it is with respect to the surface $\mathcal{S}$

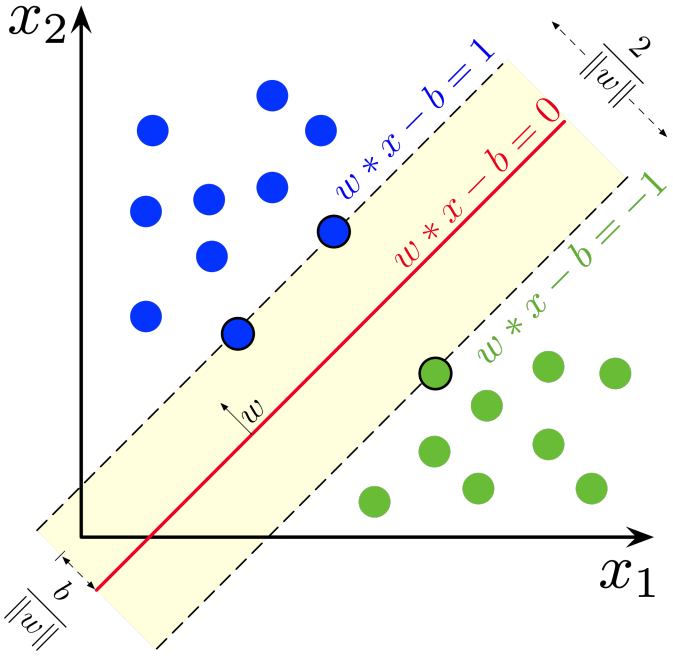
Linear SVM

We are given a training dataset of n points of the form

$$(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$$

where $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i = \{-1, 1\}$. The value of y_i indicates the class to which the point $\boldsymbol{x}_i$ belongs

Linear SVM – *Find the "maximum-margin hyperplane" that divides the group of points $\boldsymbol{x}_i$ for which $y_i = 1$ from the group of points for which $y_i = -1$, which is such that the distance between the hyperplane and the nearest point $\boldsymbol{x}_i$ from either group is maximized.*



Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are the **support vectors**

Any **hyperplane** can be written as the set of points $\mathbf{x}$ satisfying

$$\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} - b = 0$$

where $\boldsymbol{w}$ is the (not necessarily normalized) **normal vector** to the hyperplane (if the hyperplane has equation $a_1 z_1 + \cdots + a_p z_p = c$, then $(a_1, \ldots, a_n)$ is normal to the hyperplane)

The parameter $b/\|\boldsymbol{w}\|$ determines the offset of the hyperplane from the origin along the normal vector $\boldsymbol{w}$

Remark: a hyperplane defined thusly is not a subspace of $\mathbb{R}^p$ unless $b = 0$. We can of course transform the data so that it is...

Linearly separable points

Let X_1 and X_2 be two sets of points in $\mathbb{R}^p$

Then X_1 and X_2 are **linearly separable** if there exist $w_1, w_2, .., w_p, k \in \mathbb{R}$ such that

- every point $x \in X_1$ satisfies $\sum_{i=1}^{p} w_i x_i > k$
- every point $x \in X_2$ satisfies $\sum_{i=1}^{p} w_i x_i < k$

where x_i is the ith component of x

Hard-margin SVM

If the training data is **linearly separable**, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible

The region bounded by these two hyperplanes is called the "margin", and the maximum-margin hyperplane is the hyperplane that lies halfway between them

With a normalized or standardized dataset, these hyperplanes can be described by the equations

- $\mathbf{w}^\mathsf{T}\mathbf{x} - b = 1$ (anything on or above this boundary is of one class, with label 1)
- $\mathbf{w}^\mathsf{T}\mathbf{x} - b = -1$ (anything on or below this boundary is of the other class, with label -1)

Distance between these two hyperplanes is $2/\|\boldsymbol{w}\|$

$\Rightarrow$ to maximize the distance between the planes we want to minimize $\|\boldsymbol{w}\|$

The distance is computed using the distance from a point to a plane equation

We must also prevent data points from falling into the margin, so we add the following constraint: for each i either

$$\mathbf{w}^{\mathsf{T}}\mathbf{x}_i - b \geq 1\,, \text{ if } y_i = 1$$

or

$$\mathbf{w}^{\mathsf{T}}\mathbf{x}_i - b \leq -1\,, \text{ if } y_i = -1$$

(Each data point must lie on the correct side of the margin)

This can be rewritten as

$$y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b) \geq 1, \quad \text{for all } 1 \leq i \leq n$$

or

$$y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b) - 1 \geq 0, \quad \text{for all } 1 \leq i \leq n$$

We get the optimization problem:

Minimize $\|\mathbf{w}\|$ *subject to* $y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b) - 1 \geq 0$ *for* $i = 1, \ldots, n$

The $\mathbf{w}$ and b that solve this problem determine the classifier, $\mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w}^\mathsf{T}\mathbf{x} - b)$ where $\operatorname{sgn}(\cdot)$ is the **sign function**.

The maximum-margin hyperplane is completely determined by those $\boldsymbol{x}_i$ that lie nearest to it

These $\boldsymbol{x}_i$ are the **support vectors**

Writing the goal in terms of Lagrange multipliers

Recall that our goal is to

$$\textit{minimize } \|\mathbf{w}\| \textit{ subject to } y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b) - 1 \geq 0 \textit{ for } i = 1, \ldots, n$$

Using Lagrange multipliers $\lambda_1, \ldots, \lambda_n$, we have the function

$$L_P := F(\boldsymbol{w}, b\lambda_1, \ldots, \lambda_n) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \lambda_i y_i(\boldsymbol{x}_i \boldsymbol{w} + b) + \sum_{i=1}^{n} \lambda_i$$

Note that we have as many Lagrange multipliers as there are data points. Indeed, there are that many inequalities that must be satisfied

The aim is to minimise L_p with respect to $\boldsymbol{w}$ and b while the derivatives of L_p w.r.t. λ_i vanish and the $\lambda_i \geq 0$, $i = 1, \ldots, n$

Lagrange multipliers

We have already seen Lagrange multipliers, when we were studying PCA

Maximisation using Lagrange multipliers (V1.0)

We want the max of $f(x_1, \ldots, x_n)$ under the constraint $g(x_1, \ldots, x_n) = k$

1. Solve

$$\nabla f(x_1, \ldots, x_n) = \lambda \nabla g(x_1, \ldots, x_n)$$
$$g(x_1, \ldots, x_n) = k$$

   where $\nabla = (\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})$ is the **gradient operator**
2. Plug all solutions into $f(x_1, \ldots, x_n)$ and find maximum values (provided values exist and $\nabla g \neq \mathbf{0}$ there)

λ is the **Lagrange multiplier**

The gradient

$f : \mathbb{R}^n \to \mathbb{R}$ function of several variables, $\nabla = \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\right)$ the gradient operator

Then

$$\nabla f = \left(\frac{\partial}{\partial x_1} f, \ldots, \frac{\partial}{\partial x_n} f\right)$$

So ∇f is a *vector-valued* function, $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$; also written as

$$\nabla f = f_{x_1}(x_1, \ldots, x_n)\boldsymbol{e}_1 + \cdots f_{x_n}(x_1, \ldots, x_n)\boldsymbol{e}_n$$

where f_{x_i} is the partial derivative of f with respect to x_i and $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$ is the standard basis of $\mathbb{R}^n$

Lagrange multipliers (V2.0)

However, the problem we were considering then involved a single multiplier λ

Here we want $\lambda_1, \ldots, \lambda_n$

Lagrange multiplier theorem

Theorem 38

Let $f : \mathbb{R}^n \to \mathbb{R}$ be the objective function, $g : \mathbb{R}^n \to \mathbb{R}^c$ be the constraints function, both being C^1. Consider the optimisation problem

$$\begin{aligned} &\text{maximize } f(x) \\ &\text{subject to } g(x) = 0 \end{aligned}$$

Let x^* be an optimal solution to the optimization problem, such that $\text{rank}(Dg(x^*)) = c < n$, where $Dg(x^*)$ denotes the matrix of partial derivatives

$$[\partial g_j / \partial x_k]$$

Then there exists a unique Lagrange multiplier $\lambda^* \in \mathbb{R}^c$ such that

$$Df(x^*) = \lambda^{*T} Dg(x^*)$$

Lagrange multipliers (V3.0)

Here we want $\lambda_1, \ldots, \lambda_n$

But we also are looking for $\lambda_i \geq 0$

So we need to consider the so-called Karush-Kuhn-Tucker (KKT) conditions

Karush-Kuhn-Tucker (KKT) conditions

Consider the optimisation problem

$$\begin{aligned} \text{maximize } & f(x) \\ \text{subject to } & g_i(x) \leq 0 \\ & h_i(x) = 0 \end{aligned}$$

Form the Lagrangian

$$L(\boldsymbol{x}, \mu, \lambda) = f(\boldsymbol{x}) + \mu^T \boldsymbol{g}(\boldsymbol{x}) + \lambda^T \boldsymbol{h}(\boldsymbol{x})$$

Theorem 39

If $(\mathbf{x}^*, \mu^*)$ is a saddle point of $L(\mathbf{x}, \mu)$ in $\mathbf{x} \in \mathbf{X}$, $\mu \geq \mathbf{0}$, then $\mathbf{x}^*$ is an optimal vector for the above optimization problem. Suppose that $f(\mathbf{x})$ and $g_i(\mathbf{x})$, $i = 1, \ldots, m$, are convex in $\mathbf{x}$ and that there exists $\mathbf{x}_0 \in \mathbf{X}$ such that $\mathbf{g}(\mathbf{x}_0) < 0$. Then with an optimal vector $\mathbf{x}^*$ for the above optimization problem there is associated a non-negative vector μ^* such that $L(\mathbf{x}^*, \mu^*)$ is a saddle point of $L(\mathbf{x}, \mu)$

KKT conditions

$$\frac{\partial}{\partial w_\nu} L_P = w_\nu - \sum_i^n \lambda_i y_i x_{i\nu} = 0 \qquad \nu = 1, \ldots, p$$

$$\frac{\partial}{\partial b} L_P = -\sum_{i=1}^n \lambda_i y_i = 0$$

$$y_i(\boldsymbol{x}_i^T \boldsymbol{w} + b) - 1 \geq 0 \qquad i = 1, \ldots, n$$

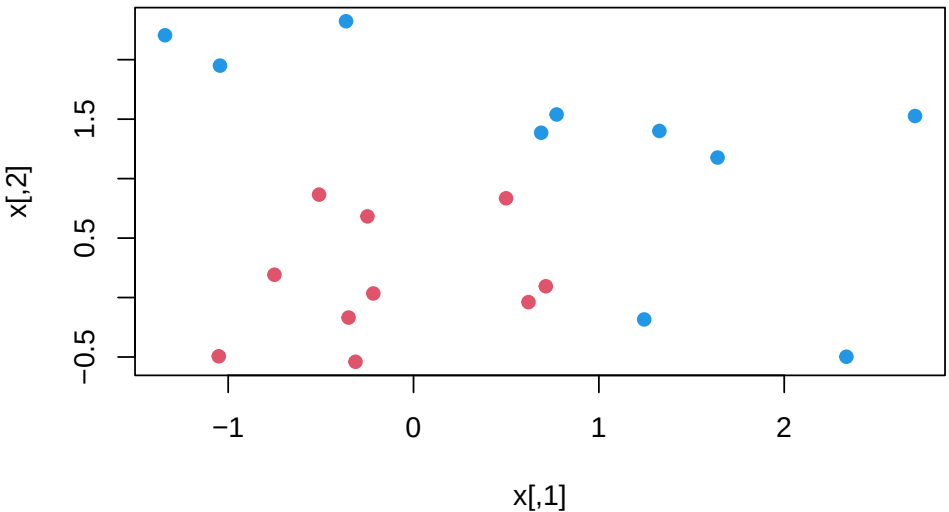
$$\lambda_i \geq 0 \qquad i = 1, \ldots, n$$

$$\lambda_i(y_i(\boldsymbol{x}_i^T \boldsymbol{w} + b) - 1) = 0 \qquad i = 1, \ldots, n$$

Numerical example

Example from here

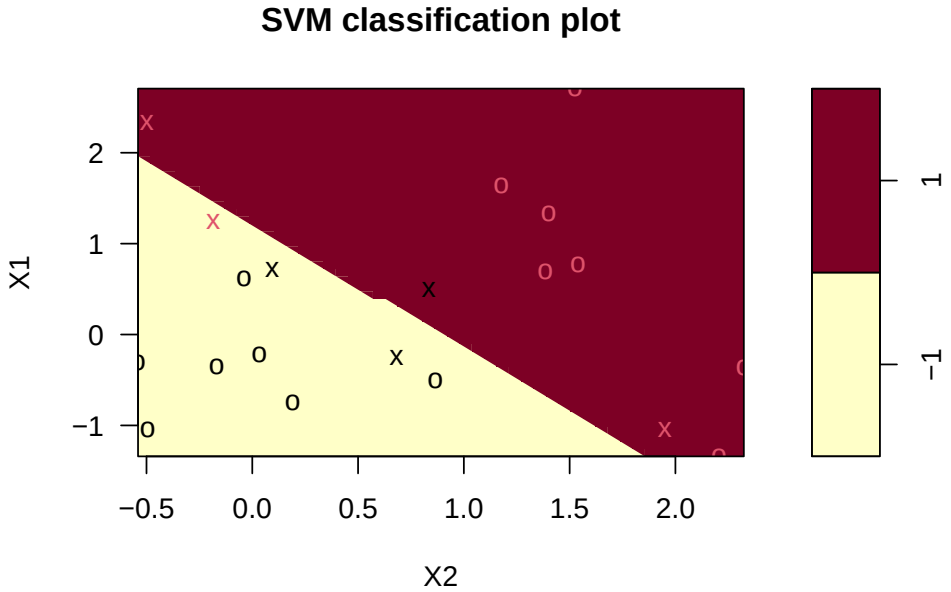
```
set.seed(10111)
x = matrix(rnorm(40), 20, 2)
y = rep(c(-1, 1), c(10, 10))
x[y == 1,] = x[y == 1,] + 1
plot(x, col = y + 3, pch = 19)
```


x[,2]
1.5
0.5
−0.5
−1
0
1
2
x[,1]

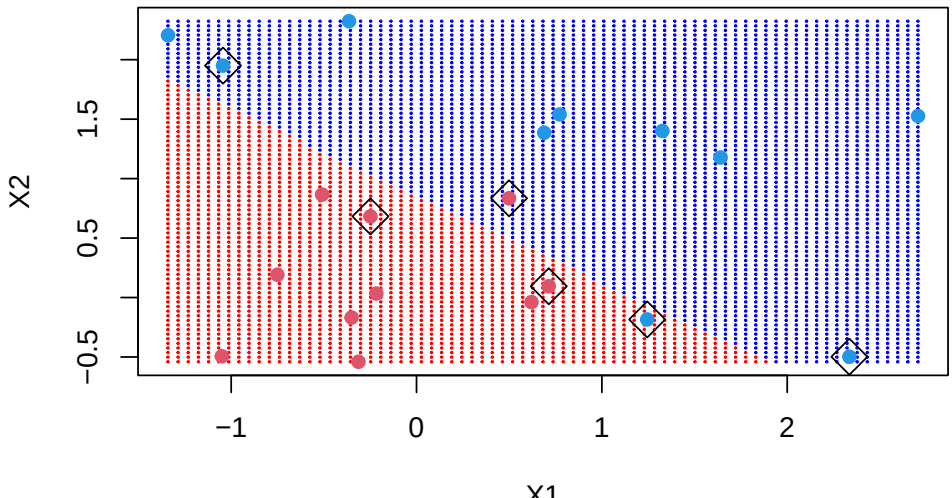
```r
dat = data.frame(x, y = as.factor(y))
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE
print(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = F
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  6
```

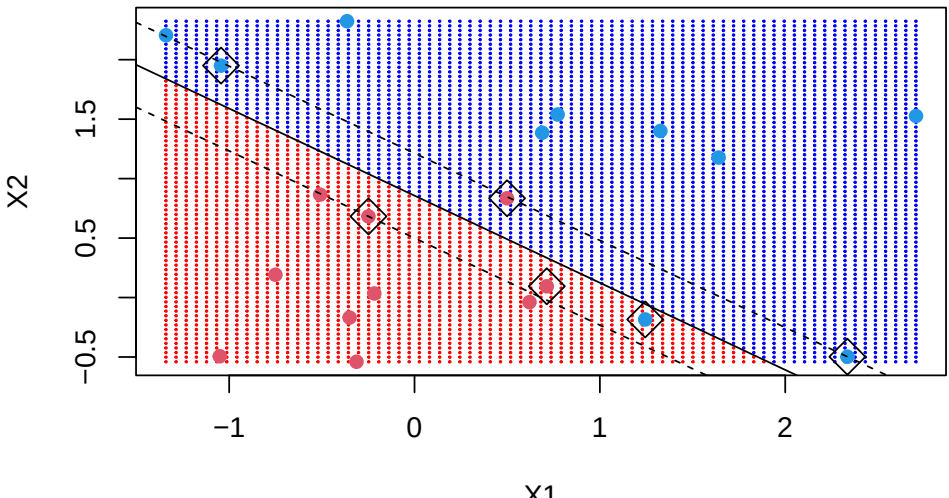
```r
plot(svmfit, dat)
```


SVM classification plot
X1
X2
2
1
0
−1
−0.5
0.0
0.5
1.0
1.5
2.0
1
−1

```
make.grid = function(x, n = 75) {
  grange = apply(x, 2, range)
  x1 = seq(from = grange[1,1], to = grange[2,1], length = n)
  x2 = seq(from = grange[1,2], to = grange[2,2], length = n)
  expand.grid(X1 = x1, X2 = x2)
}
xgrid = make.grid(x)
ygrid = predict(svmfit, xgrid)
plot(xgrid, col = c("red","blue")[as.numeric(ygrid)], pch = 20, cex = .2)
points(x, col = y + 3, pch = 19)
points(x[svmfit$index,], pch = 5, cex = 2)
```


X2
1.5
0.5
−0.5
−1
0
1
2
X1

```
beta = drop(t(svmfit$coefs)%*%x[svmfit$index,])
beta0 = svmfit$rho
plot(xgrid, col = c("red", "blue")[as.numeric(ygrid)], pch = 20, cex = .2)
points(x, col = y + 3, pch = 19)
points(x[svmfit$index,], pch = 5, cex = 2)
abline(beta0 / beta[2], -beta[1] / beta[2])
abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)
abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)
```


X2
1.5
0.5
−0.5
−1
0
1
2
X1

Soft-margin SVM

To extend SVM to cases in which the data are not linearly separable, the **hinge loss** function is helpful

$$\max\left(0, 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b)\right)$$

y_i is the ith target (i.e., in this case, 1 or -1), and $\mathbf{w}^\mathsf{T}\mathbf{x}_i - b$ is the i-th output

This function is zero if the constraint is satisfied, in other words, if $\mathbf{x}_i$ lies on the correct side of the margin

For data on the wrong side of the margin, the function's value is proportional to the distance from the margin

The goal of the optimization then is to minimize

$$\lambda \|\mathbf{w}\|^2 + \left[\frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b)\right) \right]$$

where the parameter $\lambda > 0$ determines the trade-off between increasing the margin size and ensuring that the $\mathbf{x}_i$ lie on the correct side of the margin

Thus, for sufficiently small values of λ, it will behave similar to the hard-margin SVM, if the input data are linearly classifiable, but will still learn if a classification rule is viable or not