Recitation 01. Brief Review of Vectors, Planes, and Optimization

Updated 20200206 9:50 UTC: Added vector projections and geometric series

Points and Vectors

Norm of a vector

Norm: Answer the question how big is a vector

- $\|X\|_l \equiv l - NORM := (\sum_{i=1}^{sise(X)} x_i^l)^(1/l)$
- Julia: `norm(x)`
- NumPy: `numpy.linalg.norm(x)`

If it is not specified, it is assumed to be the 2-norm, i.e. the Euclidian distance. It is also known as “length”.

Dot product of vector

Aka “scalar product” or “inner product”.

It has a relationship on how vectors are arranged relative to each other

- Algebraic definition: $x \cdot y \equiv x'y := \sum_{i=1}^{n} x_i * y_i$
- Geometric definition: $x \cdot y := \|x\| * \|y\| * cos(\theta)$ (where θ is the angle between the two vectors)
- Julia: `dot(x,y)`

- Numpy: `np.dot(x,y)`

Note that using the two definitions and the `arccos`, the inverse funcion for the cosene, you can retrieve the angle between two functions as `angle_x_y = acos(dot(x,y)/(norm(x)*norm(y)))`.

Planes

An (hyper)plane in n dimensions is a n−1 dimensional subspace defined by a linear relation. For example, if n=3 hyperplanes span 2 dimensions (and they are just called “planes”).

As hyperplanes separate the space into two sides, we can use (hyper)planes to set boundaries in classification problems, i.e. to discriminate all points on one side of the plane vs all the point on the other side.

- *Normal* of a plane: any vector perpendicular to the plane.
- *Offset of the plane with the origin*: the distance of the plan with the origin, that is the specific normal between the origin and the plane

For any point in the original space, it is part of the plane only if its vector is perpendicular to the normal, that is x is part of the plane if $(x - offset) \cdot normal = 0$.

For example, in two dimensions, let’s consider the “plane” (“line” in 1 d) between the points (4,0) and (0,4).

Here the offset is (2,2) and one possible normal is (4,4).

Let’s consider the point a = (1,3). As $(a - offset) \cdot normal = \left(\begin{bmatrix} 1 \\ 3 \end{bmatrix} - \begin{bmatrix} 2 \\ 2 \end{bmatrix} \right) \cdot \begin{bmatrix} 4 \\ 4 \end{bmatrix} = 0$, a is part of the plane.

Let's consider the point b = (1,4). As $(b - offset) \cdot normal = \left(\begin{bmatrix}1\\4\end{bmatrix} - \begin{bmatrix}2\\2\end{bmatrix}\right) \cdot \begin{bmatrix}4\\4\end{bmatrix} = 4$, b is not part of the plane.

Removing the offset creates a vector relative to a point on the plane, and then the dot product is used to check if that vector is orthogonal to the normal departing the plane from the same point.

Note that the equation $(x - offset) \cdot normal$ can be rewritten equivalently as $(x \cdot normal) - (offset \cdot normal)$, where the second term will be a constant of the plane.

Loss Function, Gradient Descent, and Chain Rule

Loss Function

The loss function, aka the *cost function* or the *race function*, is some way for us to value how far is our model from the data that we have.

We first define an "error" or "Loss". For example in Linear Regression the "error" is the Euclidean distance between the predicted and the observed value:

$L(x, y; \Theta) = \sum_{i=1}^{n} |\hat{y} - y| = \sum_{i=1}^{n} |\theta_1 x + \theta_2 - y|$

The objective is to minimise the loss function by changing the parameter theta. How?

Gradient Descent

The most common iterative algorithm to find the minimum of a function is the gradient descent.

We compute the loss function with a set of initial parameter(s), we compute the gradient of the function (the derivative concerning the various parameters), and we move our parameter(s) of a small delta against the direction of the gradient at each step:

$$\hat{\theta}_{s+1} = \hat{\theta}_{s+1} - \gamma \nabla L(x, y; \theta)$$

The γ parameter is known as the *learning rate*.

- too small learning rate: we may converge very slowly or end up trapped in small local minima;
- too high learning rate: we may diverge instead of converge to the minimum

Chain rule

How to compute the gradient for complex functions.

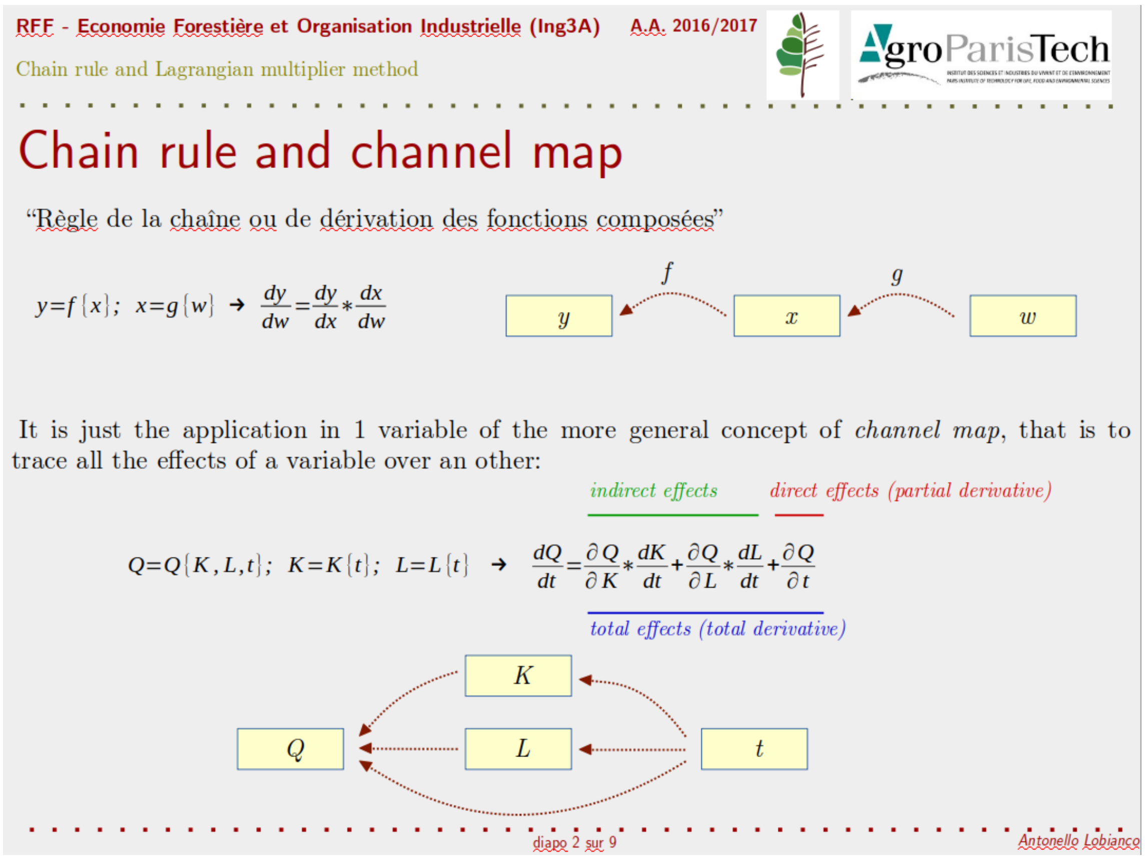
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z} * \frac{\partial z}{\partial x}$$

e.g. $\hat{y} = \frac{1}{1+e^{-(\theta_1 x + \theta_2)}} = (1 + e^{-(\theta_1 x + \theta_2)})^{-1}$,
$\frac{\partial \hat{y}}{\partial \theta_1} = - \frac{1}{(1+e^{-(\theta_1 x + \theta_2)})^2} * e^{-(\theta_1 x + \theta_2)} * -x$

For computing derivatives one can use SymPy, a library for symbolic computation. In this case the derivative can be computed with the following script:

```
from sympy import *
x, p1, p2 = symbols('x p1 p2')
y = 1/(1+exp( - (p1*x + p2)))
dy_dp1 = diff(y,p1)
print(dy_dp1)
```

It may be useful to recognise the chain rule as an application of the *chain map*, that is tracking all the effect from one variable to the other:


RFE - Economie Forestière et Organisation Industrielle (Ing3A) A.A. 2016/2017

Chain rule and Lagrangian multiplier method

Chain rule and channel map

“Règle de la chaîne ou de dérivation des fonctions composées”

$$y=f(x);\; x=g(w) \rightarrow \frac{dy}{dw}=\frac{dy}{dx}*\frac{dx}{dw}$$

$$\begin{tikzcd}\boxed{y} & \boxed{x} \arrow[l, bend right, "f"'] & \boxed{w} \arrow[l, bend right, "g"']\end{tikzcd}$$

It is just the application in 1 variable of the more general concept of *channel map*, that is to trace all the effects of a variable over an other:

$$Q=Q(K,L,t);\; K=K(t);\; L=L(t) \rightarrow \frac{dQ}{dt}=\underbrace{\overbrace{\frac{\partial Q}{\partial K}*\frac{dK}{dt}+\frac{\partial Q}{\partial L}*\frac{dL}{dt}}^{\textit{indirect effects}}+\overbrace{\frac{\partial Q}{\partial t}}^{\textit{direct effects (partial derivative)}}}_{\textit{total effects (total derivative)}}$$

$$\begin{tikzcd} & \boxed{K} \arrow[dl] & \\ \boxed{Q} & \boxed{L} \arrow[l] & \boxed{t} \arrow[ul] \arrow[l] \arrow[ll, bend left]\end{tikzcd}$$

diapo 2 sur 9 Antonello Lobianco

Geometric progressions and series

Geometric progressions are sequence of numbers where each term after the first is found by multiplying the previous one by a fixed, non-zero number called the *common ratio*.

It results that geometric series can be wrote as $a, ar, ar^2, ar^3, ar^4, \ldots$ where r is the common ratio and the first value a is called the *scale factor*.

Geometric series are the sum of the values in a geometric progression.

Closed formula exist for both finite geometric series and, provided $|r| < 1$, for infinite ones:

- $\sum_{k=m}^{n} ar^k = \frac{a(r^m - r^{n+1})}{1-r}$ with $r \neq 1$ and $m < n$
- $\sum_{k=m}^{\infty} ar^k = \frac{ar^m}{1-r}$ with $|r| < 1$ and $m < n$.

Where m is the first element of the series that you want to consider for the summation. Typically $m = 0$, i.e. the summation considers the whole series from the scale factor onward.

For many more details on geometric progressions and series consult the relative excellent Wikipedia entry.

Vector projections

Let's be a and b two (not necessary unit) vectors. We want to compute the vector c being the projection of a on b and its l-2 norm (or length).

Let's start from the lenght. We know from a well-known trigonometric equation that

$\|c\| = \|a\| * cos(\alpha)$, where α is the angle between the two vectors a and b.

But we also know that the dot product $a \cdot b$ is equal to $\|a\| * \|b\| * cos(\alpha)$.

By substitution we find that $\|c\| = \frac{a \cdot b}{\|b\|}$. This quantity is also called the *component* of a in the direction of b.

To find the vector c we now simply multiply $\|c\|$ by the unit vector in the direction of b, $\frac{b}{\|b\|}$, obtaining $c = \frac{a \cdot b}{\|b\|^2} * b$.

If b is already a unit vector, the above equations reduce to:

$\|c\| = a \cdot b$ and $c = (a \cdot b) * b$

In Julia:

```
using LinearAlgebra
a = [4,1]
b = [2,3]
normC = dot(a,b)/norm(b)
c = (dot(a,b)/norm(b)^2) * b
```

In Python:

```
import numpy as np
a = np.array([4,1])
b = np.array([2,3])
normC = np.dot(a,b)/np.linalg.norm(b)
c = (np.dot(a,b)/np.linalg.norm(b)**2) * b
```