I

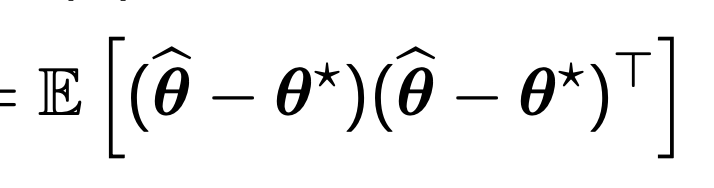
1) Cov(X+mu) = Cov(X) mu étant déterministe

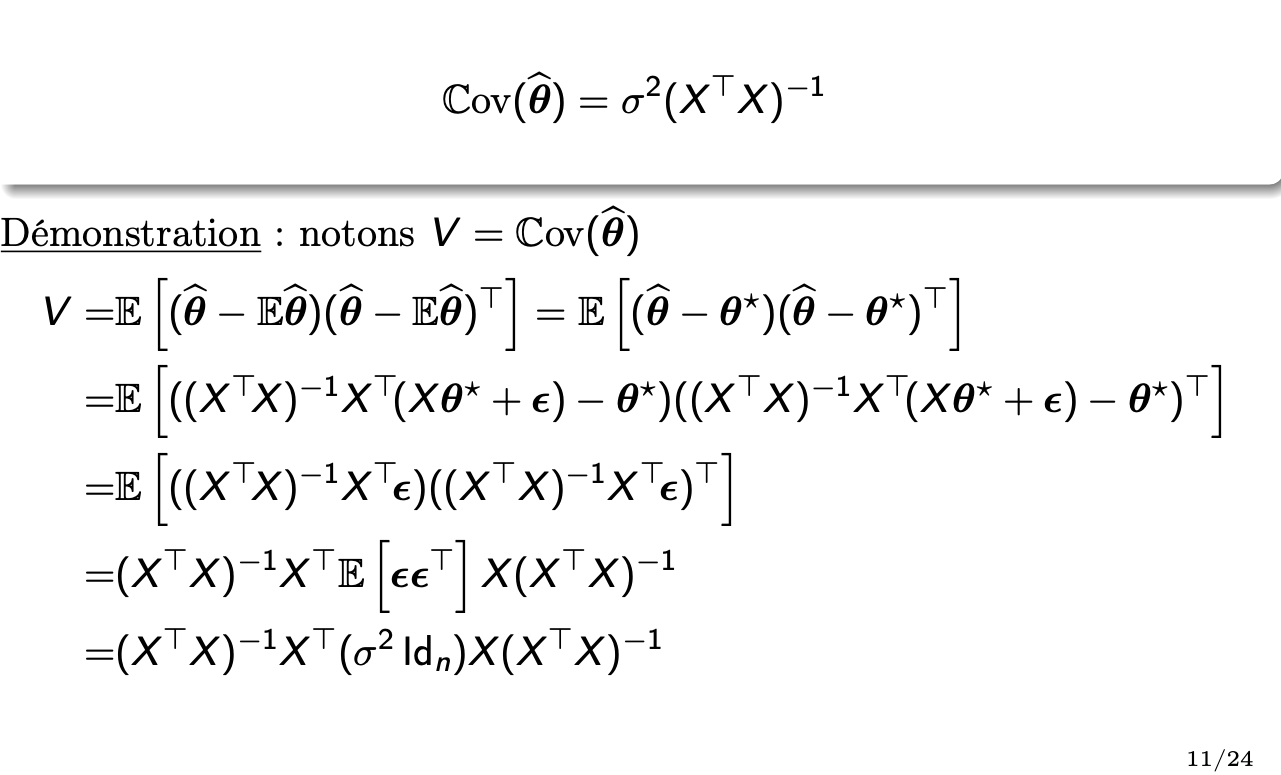
Explication 1) Comme X vector Cov(X+mu) = Var(X+mu) et mu déterministe

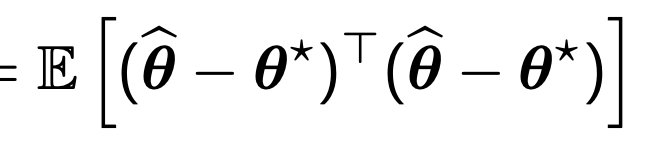
Explication 2 =

Cov (X+ MU) = E[ ((X-mu) - E( x - MU) ) . ((X-mu)-E(X-mu))T ]

Par linéarité de l’espérance on sort les mu des espérance







2) COV(AX) = A . cov(X) . AT

= E[ (AX - E(AX) ).(AX - E(AX))T ]

= E[ A (X - E(X)). (X-E(X))T. AT ]

= A E[(X - E(X))(X-E(X))T] AT

3) Cf. cours 1

Omega = {1,2,3,4,5,6}

P(X=k) = 1/6 pour tout k = {1,2,3,4,5}

P(X = 6) = 1 - sum P(X = k) avec k allant de 1 à 5

4) mu = moyenne empirique

Biais = 0

Var = var(mu)/n

5) Biais(estimator) = -sigma^2 / n ?

**Biais(estimator) = E(estimator) - valeur\_theorique = sigma^2**

=E(1/n ∑[yi - yn]^2)

=E(1/n ∑[yi - μ + μ -yn]^2)

=E(1/n ∑(yi - μ)^2 +( μ -yn)^2+2(yi - μ)( μ -yn))

=1/n ∑E(yi - μ)^2 +1/n∑E( μ -yn)^2+2(yi - μ)( μ -yn))

=1/n ∑σ^2 +1/n∑var(yn)

=σ^2 -σ^2/n

Biais = E(1/n ∑[yi - yn]^2) -σ^2 = σ^2 -σ^2/n - σ^2 = -σ^2/n

B = 1/n E[∑ (yi – yn)² ] – **σ²**

B = 1/n E[∑ yi² + yn² - 2yi.yn)] – **σ².**

**On distribue l’espérance : E(yi)=0.**

**Or, Var(yi) = E(yi²) – E(yi)² = E(yi²)**

B = 1/n (∑ Var(yi) + Var(yn)) – **σ²**

**B = 1/n (n σ² + σ²) – σ²**

B = 1/n . **σ²**

6) Risque(estimator) = Biais(estimator) ^2 + Variance(estimator)

Biais(estimator) ^ 2 = sigma^4 / n^2

Variance(estimator) = Variance(1/n\*sum((yi-yn)^2)) = Variance(sigma^2) ??? par definition?

Risque quadratique = Var + B^2

= Var (sigma^n) + B(sigma^n)^2

= (1/n).sigma^2 + (E[sigma^n] - sigma)^2

= (1/n).sigma^2 + (((n-1)/n).sigma - sigma)^2

= (1/n).sigma^2 + (sigma/n)^2

=sigma^2.(1/n + 1/n^2)

Yi suit une loi normale : N(0, **σ²)**

**Donc Xi = Yi/σ suit une loi centrée**

-Var (1/n S (Xi - Xn)²) = 1/n² 2n

**On a :** Var (1/n S (yi/**σ** - yn/**σ**)²) = **σ^4 .** (1/n² \* 2n)

R = **σ^4 (1/n² (2n+1))**

7) Yhat = Xtheta (1)

Yhat = HY (H amtrice de projection)

1. + remplacement de theta par XTX-1 . XT

Identification de H = X (XTX)-1 XT

Remplacement par le vecteur 1n

H1N = 1\_n . (1\_nT . 1\_n)-1 . 1\_nT = 1/n \* (matrice remplie de 1 de taille n\*n)

8) Tous les yi constants et égaux

e.g.

(2, 2, 2)

(1, 1, 1)

(y, y, y)

=> K \* 1n

EXO II

1) Sum de fonctions convexes est convexe

2)

theta0 = yn - theta1 \* xn

theta1 = cov(x, y) \* std(y) / std(x)

EXO 3

1) prendre au hasard a0 et calcul de la pente en ce point, en fonction du signe de la pente soit on avance sur la courbe de alpha (=pas à prendre petit) => prise de a1 et recalcul de la pente

Soit on recule sur la courbe de alpha (= pas à prendre petit) => prise de a1 et recalcul de la pente

=> Jusqu’à converger vers le minimum (convexe = on arrive au minimum quand pente change de signe)

class **GradientDescentLinearRegression**:

def \_\_init\_\_(self, learning\_rate=0.01, iterations=1000):

self.learning\_rate, self.iterations = learning\_rate, iterations

def fit(self, X, y):

b = 0

m = 5

n = X.shape[0]

for \_ **in** range(self.iterations):

b\_gradient = -2 \* np.sum(y - m\*X + b) / n

m\_gradient = -2 \* np.sum(X\*(y - (m\*X + b))) / n

b = b + (self.learning\_rate \* b\_gradient)

m = m - (self.learning\_rate \* m\_gradient)

self.m, self.b = m, b

def predict(self, X):

return self.m\*X + self.b

<https://sebastianraschka.com/faq/docs/sgd-methods.html>

<https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate>

2) Vu au TD

3) Vu au TD

4) theta = (Xt.X)^-1.Xt.Y

5) Xt.X.theta = Xt.Y

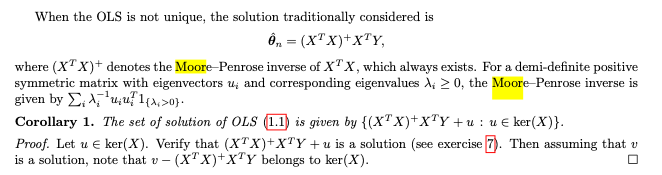
Ou (Y - X.theta)t . X.theta = 0

6) existence et unicité? Hilbert thm de projection => existence et l’unicité ssi (XtX) inversible

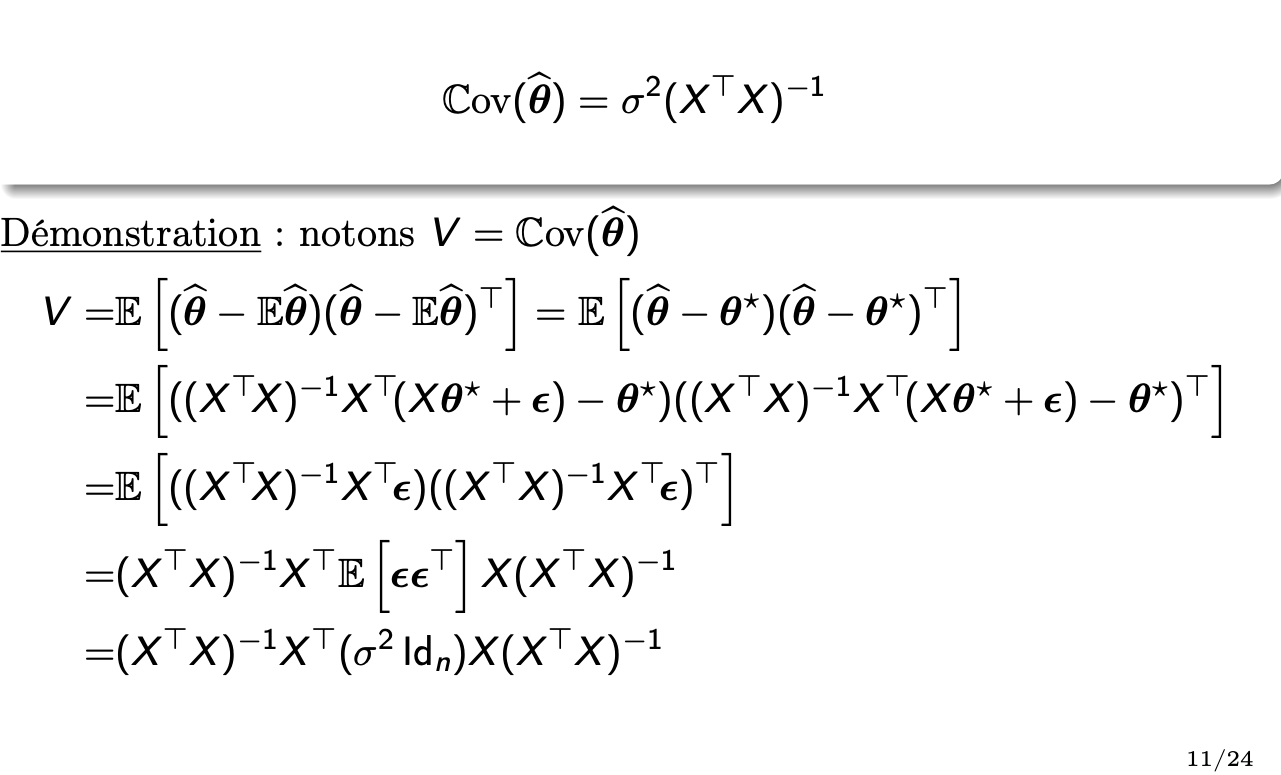
(Ker(XtX) = Ker(X) = 0 ou valeurs propres non nulles ou det(XtX) != 0)

7) XtX (theta chapeau + theta k) = XtY

THETA =(Xt.X)^+.Xt.Y + ker(X)



8)



9)

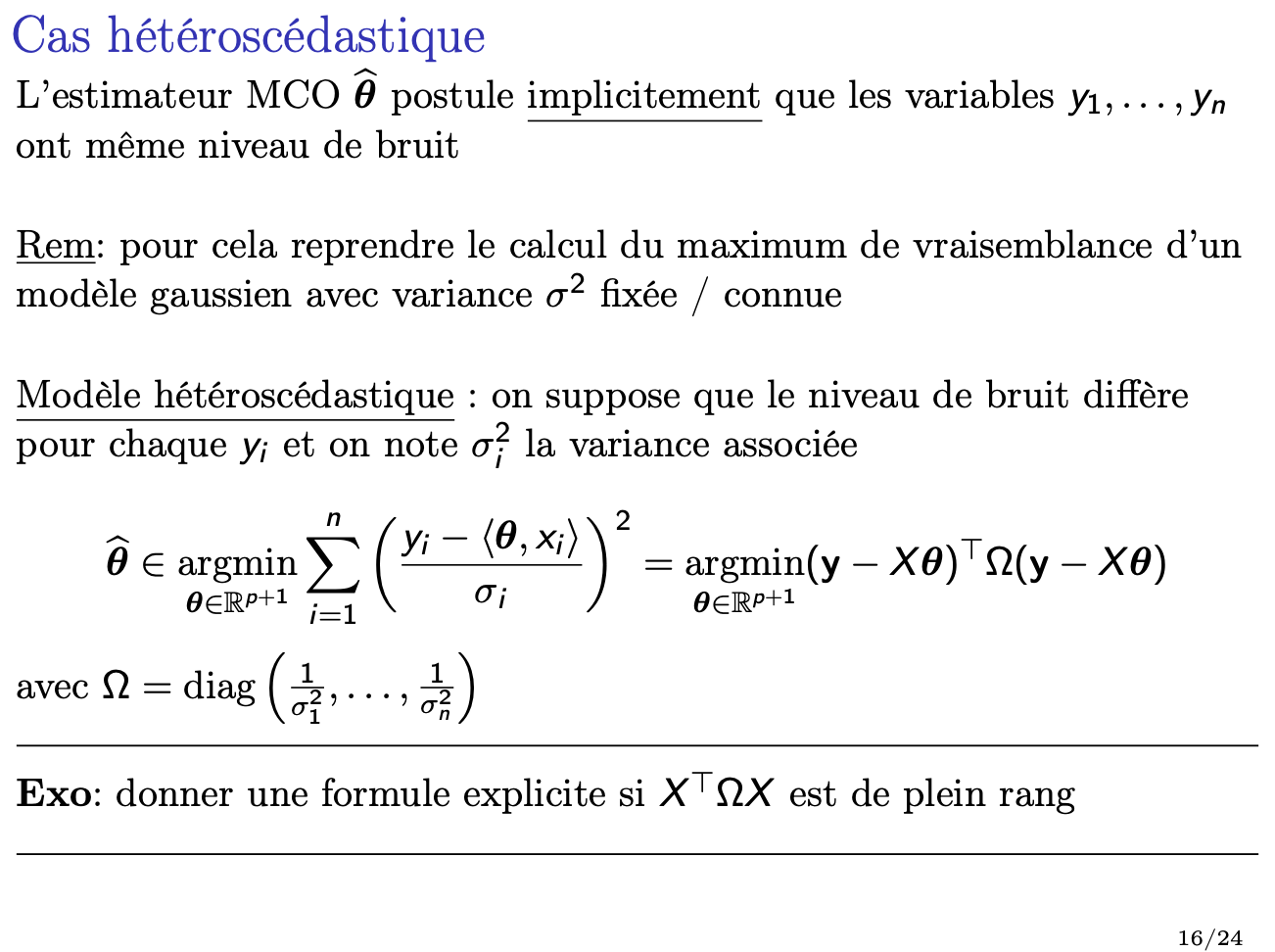
sigma \_ =1 / (n-rang(X)) . (norme\_2(Y-Y\_chapeau))^2

Obtenu en calculant l’espérance d’un possible estimateur (nome\_2 (Y - Y\_chapeau))^2

Cette espérance est égale sigma^2 \* (n - rang(X))

10) c) Var(theta\_b,n) sigma ^ 2 . D ^-1 (Xt.X)^1 . D ^-1

11)



X = U.D.Vt

Xt = V.D.Ut

Xt.omega.X = (V.D.Ut).omega.U.D.Vt => matrice de valeurs propres les si^2 \* omega\_i ??

Xt.X = V.D.D.Vt => matrice de valeurs propres les si^2

(Xt.omega.X)^-1.Xt.omega.Y

12) ???

13) Sqrt(n) (Beta - Beta\*) suit N(0, G^-1. σ²)

14) rang(X) / n \* sigma ^ 2 quand X est de plein rang + model homoscedastic