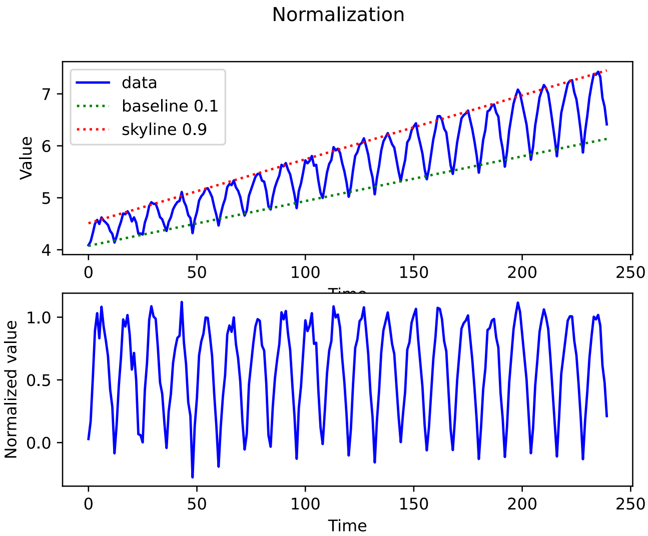
Régression Quantile

Romain Hérault, Benoit Gaüzère, Stéphane Canu
romain.herault@unicaen.fr

Université de Caen - INSA de Rouen - ITI

April 17, 2024

Conditionnement du signal



Régression linéaire : Formulation

Formulation

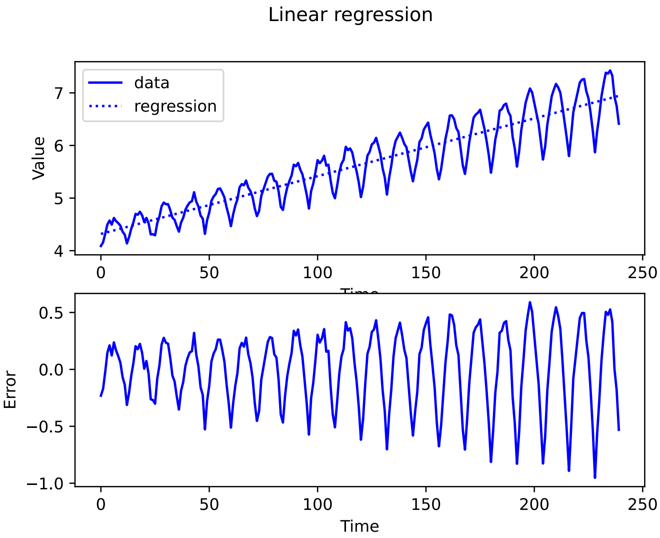
$$\min_{\boldsymbol{\alpha}} \quad ||X * \boldsymbol{\alpha} - \mathbf{y}||^2 \ .$$

avec $X \in \mathbb{R}^{n \times 2}$ la variable explicative concaténée de $\mathbf{1}$, $\mathbf{y} \in \mathbb{R}^n$ la variable à expliquer et $\boldsymbol{\alpha} \in \mathbb{R}^2$ les paramètres du modèle.

Régression linéaire : Code avec cvxpy

```python
import numpy as np
import cvxpy as cp

def linReg1(x,y):

    n=y.size
    p=2

    # Données de la regression
    X=np.zeros((n,p))
    Y=np.zeros((n,1))

    X[:,0]=x
    X[:,1]=np.ones((n,))

    Y[:,0]=y

    # Variables
    alpha = cp.Variable((p,1))

    # Constitution du problème
    objective = cp.Minimize(cp.sum(cp.square(X@alpha-Y)))
    prob = cp.Problem(objective)

    # Résolution
    prob.solve()
    alphares=np.array(alpha.value)

    return alphares
```

Régression linéaire: Illustration



Régression linéaire avec ressort : Formulation

Formulation

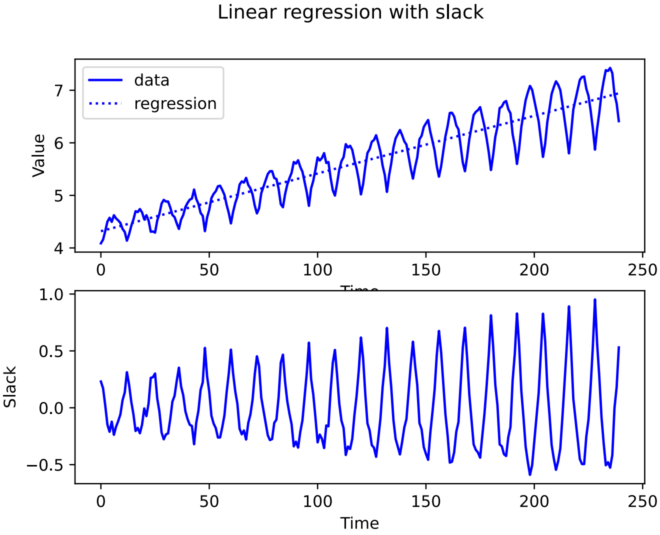
$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\xi}} & \quad ||\boldsymbol{\xi}||^2 \ , \\ s.t. & \quad X * \boldsymbol{\alpha} - \mathbf{y} - \boldsymbol{\xi} = 0 \ . \end{aligned}$$

avec $\boldsymbol{\xi} \in \mathbb{R}^n$ les variables ressorts.

Régression linéaire avec ressort : Code

```
# Variables
alpha = cp.Variable((p,1))
slack = cp.Variable((n,1))

# Constitution du problème
objective = cp.Minimize(cp.sum(cp.square(slack)))
constraints = [
    X@alpha-Y-slack==0]
prob = cp.Problem(objective, constraints)

# Résolution
prob.solve()
alphares=np.array(alpha.value)
slackres=np.array(slack.value)
```

Régression linéaire avec ressort : Illustration



Ligne de crête: Formulation

Formulation

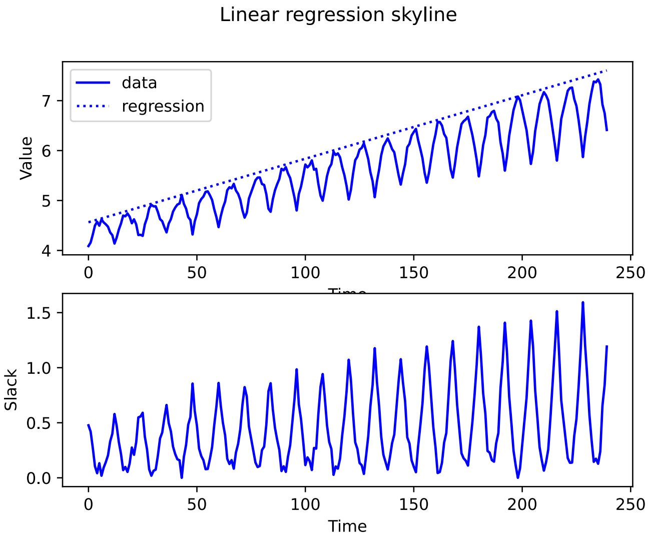
$$
\begin{aligned}
\min_{\boldsymbol{\alpha},\boldsymbol{\xi}} \quad & ||\boldsymbol{\xi}||^2 \ , \\
s.t. \quad & X * \boldsymbol{\alpha} - \mathbf{y} - \boldsymbol{\xi} = 0 \ , \\
& \boldsymbol{\xi}_i \geq 0 \ \ \forall i \in [1 \ldots n] \ .
\end{aligned}
$$

avec $\boldsymbol{\xi} \in \mathbb{R}^n$ les variables ressorts.

Ligne de crête: Code

```
alpha = cp.Variable((p,1))
slack = cp.Variable((n,1))

# Constitution du problème
objective = cp.Minimize(cp.sum(cp.square(slack)))
constraints = [
    X@alpha-Y-slack==0,
    #X@alpha-Y>0 si  X@alpha (droite) >   Y (observation)
    # donc -slack<=0 donc slack >=0
    slack>=0]
prob = cp.Problem(objective, constraints)

# Résolution
prob.solve()
alphares=np.array(alpha.value)
slackres=np.array(slack.value)
```

Ligne de crête: Illustration



Ligne de base: Formulation

Formulation

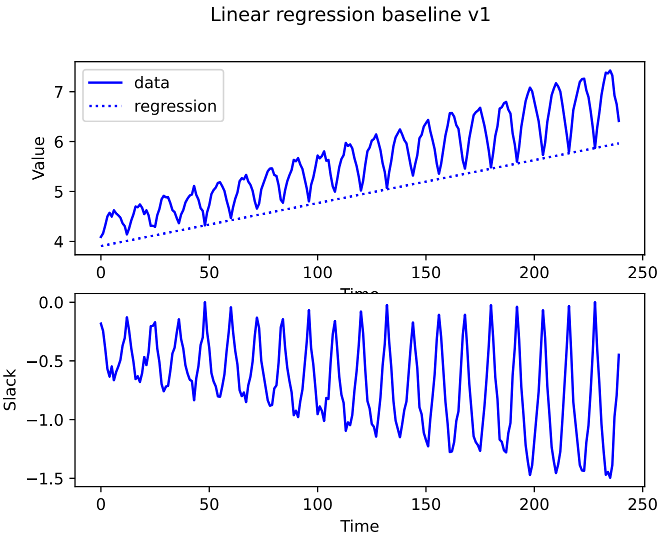
$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\xi}} \quad & ||\boldsymbol{\xi}||^2 \ , \\ s.t. \quad & X * \boldsymbol{\alpha} - \mathbf{y} - \boldsymbol{\xi} = 0 \ , \\ & -\boldsymbol{\xi}_i \geq 0 \ \ \forall i \in [1 \ldots n] \ . \end{aligned}$$

avec $\boldsymbol{\xi} \in \mathbb{R}^n$ les variables ressorts.

Ligne de base: Code

```
alpha = cp.Variable((p,1))
slack = cp.Variable((n,1))

# Constitution du problème
objective = cp.Minimize(cp.sum(cp.square(slack)))
constraints = [
    X@alpha-Y-slack==0,
    #X@alpha-Y<0 si Y (observation ) >  X@alpha (droite)
    # donc -slack>=0
    -slack>=0]
prob = cp.Problem(objective, constraints)

# Résolution
prob.solve()
alphares=np.array(alpha.value)
slackres=np.array(slack.value)
```

Ligne de base: Illustration



Ligne de base: ReFormulation

Formulation

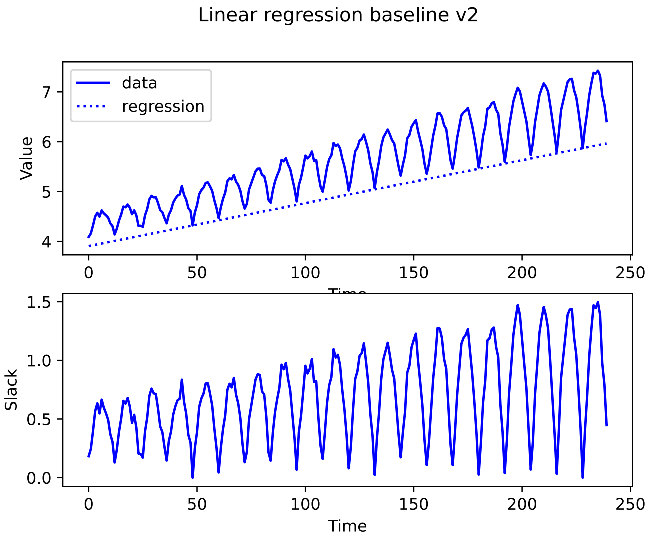
$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\xi}} \quad & ||\boldsymbol{\xi}||^2 \ , \\ s.t. \quad & X * \boldsymbol{\alpha} - \mathbf{y} + \boldsymbol{\xi} = 0 \ , \\ & \boldsymbol{\xi}_i \geq 0 \ \ \forall i \in [1 \dots n] \ . \end{aligned}$$

avec $\boldsymbol{\xi} \in \mathbb{R}^n$ les variables ressorts.

Ligne de base: ReCode

```
alpha = cp.Variable((p,1))
slackm = cp.Variable((n,1))

# Constitution du problème
objective = cp.Minimize(cp.sum(cp.square(slackm)))
constraints = [
    X@alpha-Y+slackm==0,
    #X@alpha-Y<0 si Y (observation ) >  X@alpha (droite)
    # donc slackm>=0
    slackm>=0]
prob = cp.Problem(objective, constraints)

# Résolution
prob.solve()
alphares=np.array(alpha.value)
slackm=np.array(slackm.value)
```

Ligne de base: ReIllustration



Régression médiane: Formulation

Formulation

$$
\begin{aligned}
\min_{\boldsymbol{\alpha}} \quad & ||\boldsymbol{\xi}^{(-)} + \boldsymbol{\xi}^{(+)}||^2 \ , \\
s.t. \quad & X * \boldsymbol{\alpha} - y + \boldsymbol{\xi}^{(-)} - \boldsymbol{\xi}^{(+)} = 0 \ , \\
& \boldsymbol{\xi}_i^{(-)} \geq 0 \ \ \forall i \in [1 \ldots n] \ , \\
& \boldsymbol{\xi}_i^{(+)} \geq 0 \ \ \forall i \in [1 \ldots n] \ .
\end{aligned}
$$

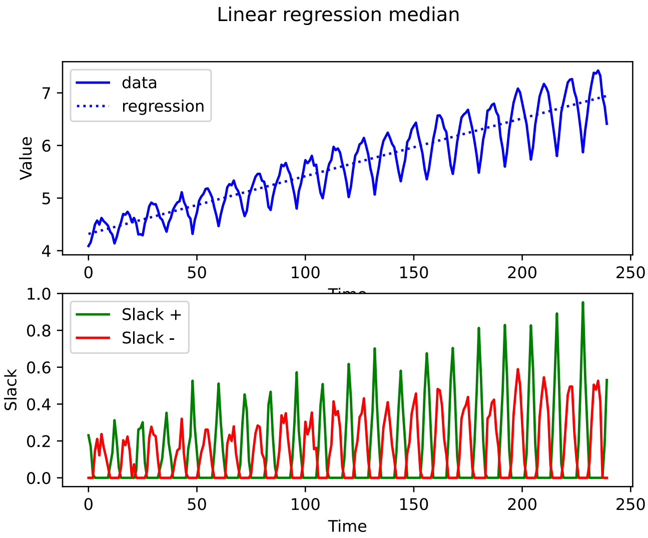
où $y \in \mathbb{R}^n$, et les paramètres $\boldsymbol{\alpha} \in \mathbb{R}^2$, $\boldsymbol{\xi}^{(-)} \in \mathbb{R}^n$, $\boldsymbol{\xi}^{(-)} \in \mathbb{R}^n$.

Si $\boldsymbol{\xi}_i^{(-)} \geq 0$ alors $\boldsymbol{\xi}_i^{(+)} = 0$, c'est le cas où y est au dessus de la droite de régression.
Si $\boldsymbol{\xi}_i^{(+)} \geq 0$ alors $\boldsymbol{\xi}_i^{(-)} = 0$, c'est le cas où y est en dessous de la droite de régression.

Régression médiane: Code

```
alpha = cp.Variable((p,1))
slackm = cp.Variable((n,1))
slackp = cp.Variable((n,1))

# Constitution du problème
objective = cp.Minimize(cp.sum(cp.square(slackm+slackp)))
constraints = [
    X@alpha-Y+slackm-slackp==0,
    slackp>=0,
    slackm>=0]
prob = cp.Problem(objective, constraints)

# Résolution
prob.solve()
alphares=np.array(alpha.value)
slackm=np.array(slackm.value)
slackp=np.array(slackp.value)
```

Régression médiane: Illustration



Régression quantile: Formulation

La régression quantile donne une estimation d'un quantile du signal aléatoire y pour chaque instant t.

Formulation

$$
\begin{aligned}
\min_{\boldsymbol{\alpha}} \quad & ||\tau * \boldsymbol{\xi}^{(-)} + (1-\tau)\boldsymbol{\xi}^{(+)}||^2 \ , \\
s.t. \quad & X * \boldsymbol{\alpha} - y + \boldsymbol{\xi}^{(-)} - \boldsymbol{\xi}^{(+)} = 0 \ , \\
& \boldsymbol{\xi}_i^{(-)} \geq 0 \ \forall i \in [1..n] \ , \\
& \boldsymbol{\xi}_i^{(+)} \geq 0 \ \forall i \in [1..n] \ .
\end{aligned}
$$

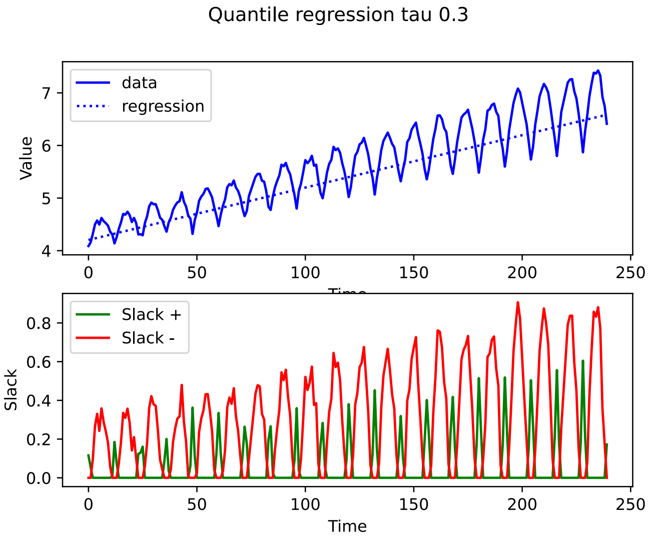
où $\tau \in [0 \ \ 1]$, $y \in \mathbb{R}^n$, et les paramètres $\boldsymbol{\alpha} \in \mathbb{R}^2$, $\boldsymbol{\xi}^{(-)} \in \mathbb{R}^n$, $\boldsymbol{\xi}^{(-)} \in \mathbb{R}^n$.

Pour faire le conditionnement, on peut utiliser deux **régressions**, une avec un τ bas (0.1) pour la ligne de base, une avec un τ haut (0.9) pour la ligne de crête.

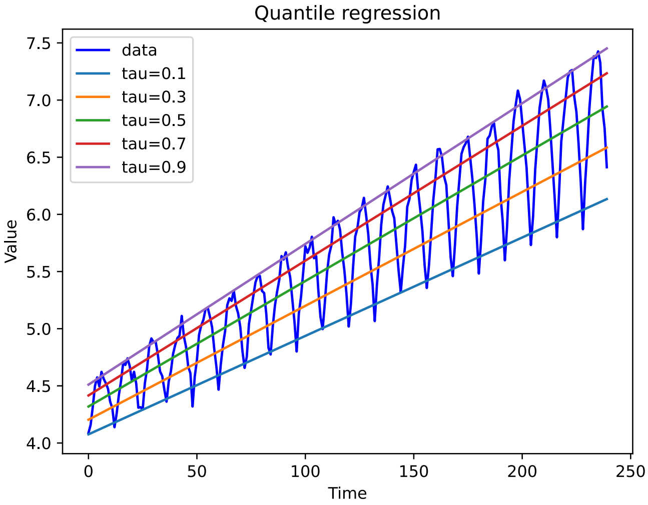
Régression quantile: Code

```
alpha = cp.Variable((p,1))
slackm = cp.Variable((n,1))
slackp = cp.Variable((n,1))

# Constitution du problème
objective =
↪  cp.Minimize(cp.sum(cp.square(tau*slackm+(1-tau)*slackp)))
constraints = [
    X@alpha-Y+slackm-slackp==0,
    slackp>=0,
    slackm>=0]
prob = cp.Problem(objective, constraints)

# Résolution
prob.solve()
alphares=np.array(alpha.value)
slackm=np.array(slackm.value)
slackp=np.array(slackp.value)
```

Régression quantile: Illustration pour $\tau = 0.3$



Régression quantile: Illustration pour divers τ



Régression quantile: Normalisation