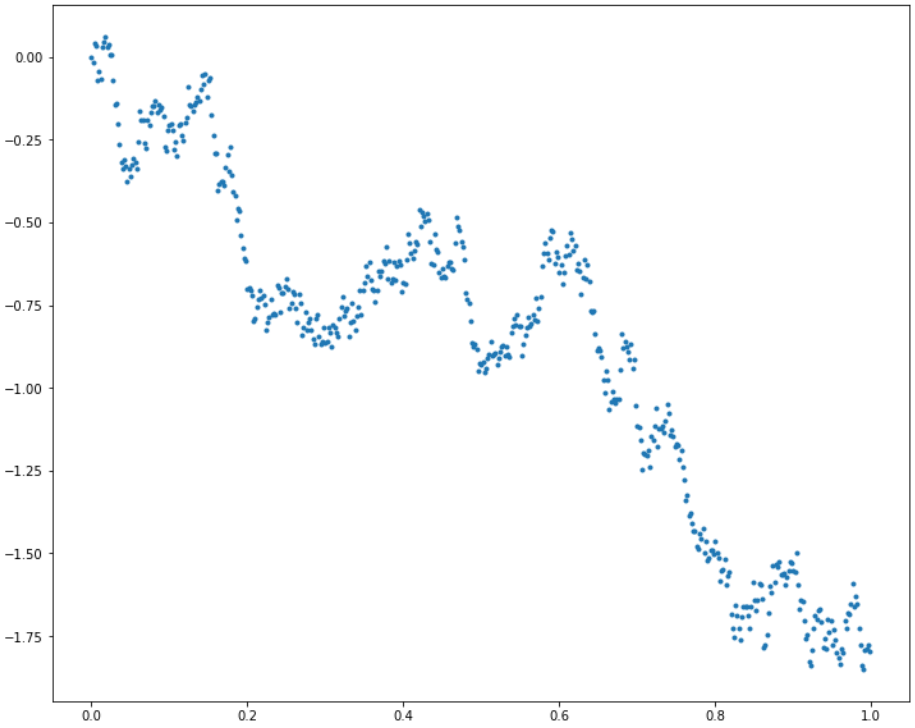
Partie 3 : Mise en œuvre numérique

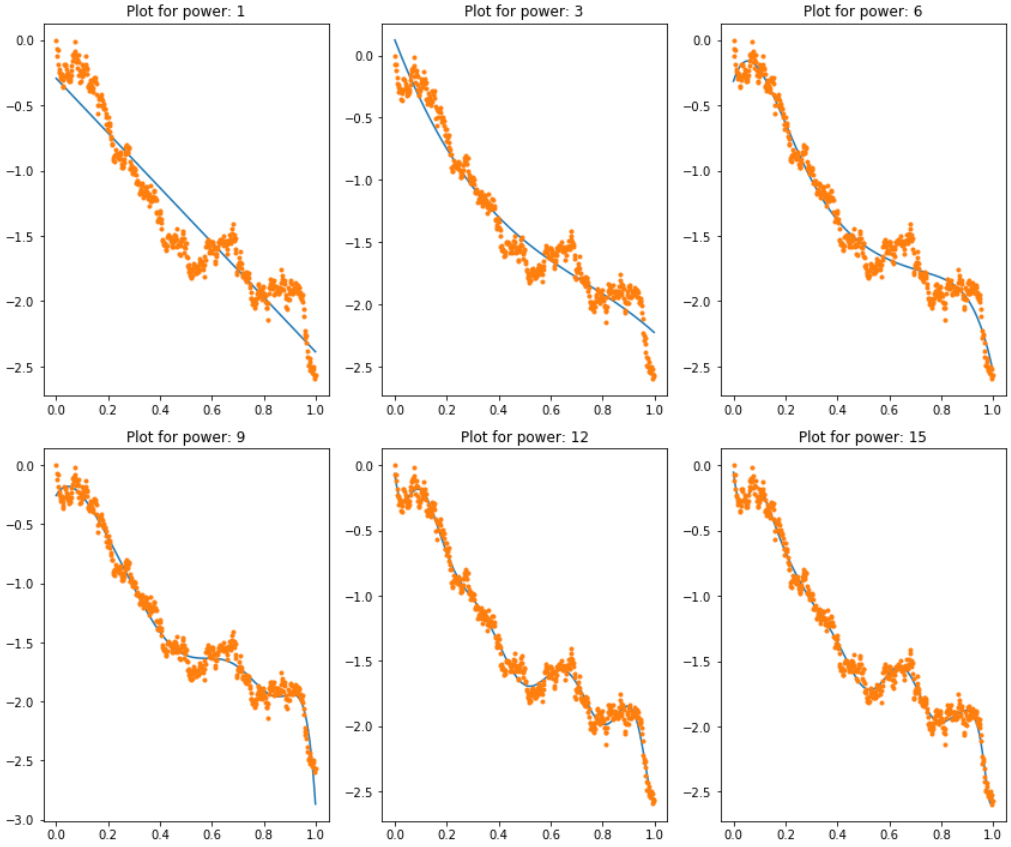
Dans cette partie nous nous concentrerons sur la régression polynomiale et l'estimateur Ridge dont nous avons étudié les propriétés théoriques dans la partie 4. Nous mettons ici en évidence les problèmes d'*underfitting* et *overfitting* des modèles.
Nous exposons nos résultats et disposons le code Python en intégralité en annexe à la fin de ce document.

Nous simulons et discrétisons X un mouvement brownien sur [0,1]. Ce qui nous donne la figure suivante :



Régression polynomiale

Notre but est d'estimer ce mouvement brownien par des polynômes de degré allant de 1 à 15. On ajoute une colonne à notre dataframe pour chaque puissance.
Ceci étant fait, nous réalisons 15 différentes régressions linéaires, la première de degré 1, la seconde de degré 2… Pour cela, nous définissons une fonction générique prenant en argument le degré souhaité. Ensuite, nous stockons tous nos résultats dans un dataframe et traçons 6 modèles pour avoir une idée de la tendance polynomiale.
On s'attend à ce que les modèles augmentent en complexité avec le degré du polynôme et que notre estimateur fonctionnel (un polynôme) soit de plus en plus proche de la fonction voulue.
En effet c'est exactement ce que l'on constate sur les 6 graphiques suivants :



Affichons les coefficients de la régression polynomial pour les estimations de degré 1,2,7, 14,15. Ce qui nous donne les résultats suivants :

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5
model_pow_1	20	0.36	1.3	NaN	NaN	NaN	NaN
model_pow_2	19	0.25	2	-0.67	NaN	NaN	NaN
model_pow_7	4.8	-0.0082	8.5	-99	6.7e+02	-2.1e+03	3.2e+03
model_pow_14	3.9	-0.0023	26	-1.4e+03	3.3e+04	-4.2e+05	3.2e+06
model_pow_15	3.9	-0.018	30	-1.6e+03	4e+04	-5.2e+05	4.1e+06

	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	\
model_pow_1	NaN	NaN	NaN	NaN	NaN	NaN	
model_pow_2	NaN	NaN	NaN	NaN	NaN	NaN	
model_pow_7	-2.3e+03	6.7e+02	NaN	NaN	NaN	NaN	
model_pow_14	-1.6e+07	5.4e+07	-1.3e+08	2.1e+08	-2.5e+08	2e+08	
model_pow_15	-2.1e+07	7.5e+07	-1.9e+08	3.4e+08	-4.4e+08	4.1e+08	

	coef_x_12	coef_x_13	coef_x_14	coef_x_15
model_pow_1	NaN	NaN	NaN	NaN
model_pow_2	NaN	NaN	NaN	NaN
model_pow_7	NaN	NaN	NaN	NaN
model_pow_14	-1e+08	3.2e+07	-4.5e+06	NaN
model_pow_15	-2.6e+08	1.1e+08	-2.8e+07	3.1e+06

Nous remarquons que lorsque nous augmentons le degré de notre estimateur polynomial, ses coefficients augmentent de façon exponentielle ce qui cause un *overfitting*.
D'où la nécessité d'introduire un nouvel estimateur fonctionnel.

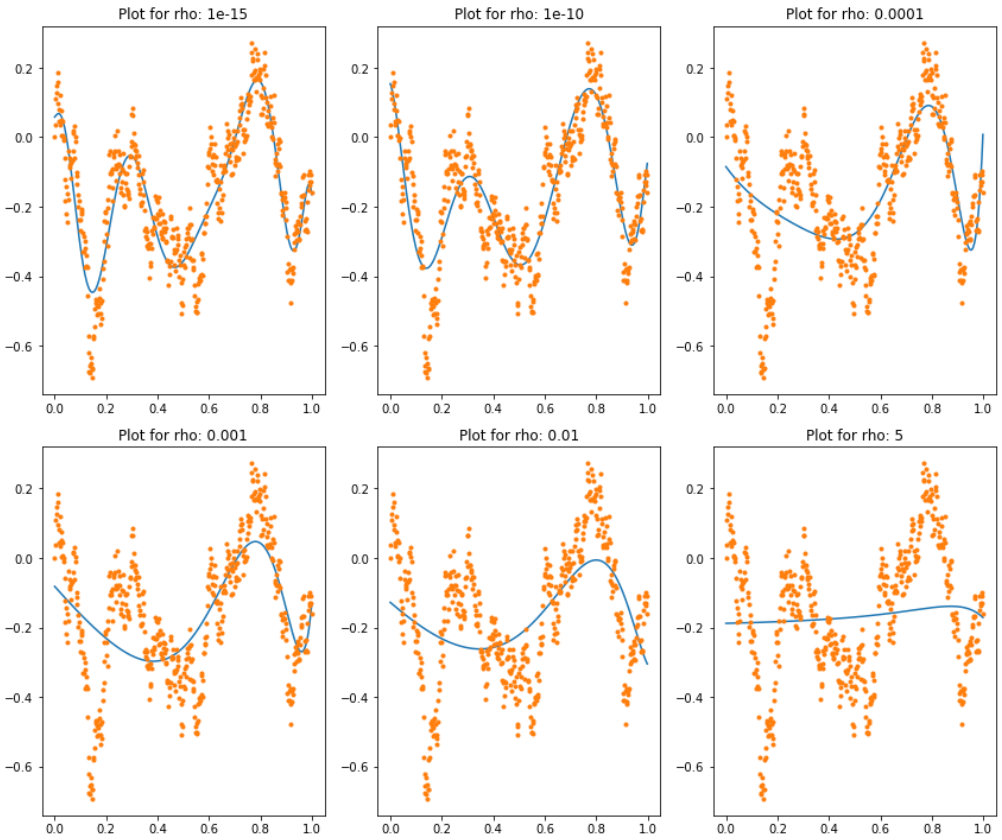
Estimateur Ridge

Notre objectif est de minimiser en *beta* l'expression suivante : RSS + *rho* * norme de *beta*.
(cf. partie 4 pour plus de détails).

Le paramètre *rho* mesure l'importance donnée à la norme de *beta* par rapport à celle du RSS dans la minimisation.
Rho peut prendre plusieurs valeurs :

- *Rho* = 0 : On retombe dans le cadre d'une simple régression linéaire (cf. début de la partie théorique pour plus de détails).
- *Rho* = ∞ : Les coefficients seront tous nuls.
- 0 < *rho* < ∞ : la valeur de *rho* détermine l'importance donnée aux différents éléments de notre expression. On se place dans ce cadre pour la suite.

Nous définissons comme précédemment une fonction générique pour l'estimateur Ridge. Puis nous analysons les résultats pour 10 valeurs différentes de rho (de 1e-15 à 20). Contrairement à la partie précédente, ces 10 modèles contiennent toutes les 15 variables, il n'y a que la valeur de rho qui diffère.
Ce qui nous donne les graphiques suivants :



Nous voyons clairement que lorsque la valeur de *rho* augmente la complexité du modèle diminue. Cependant de hautes valeurs de *rho* réduisent le problème d'*overfitting* mais de trop

grandes valeurs causent le problème inverse d'*underfitting*. Ainsi, *rho* doit être bien choisi par exemple en utilisant la technique de la cross-validation (validation croisée).

Conclusion

Après avoir étudié et implémenté sous Python différentes procédures d'estimation pour notre problème d'ACP fonctionnelle, nous parvenons aux conclusions suivantes :
L'estimateur Ridge donne une bonne approximation de *beta* et permet de résoudre de manière efficace le problème d'*overfitting*.
Cependant il n'est pas le meilleur estimateur lorsque les jeux de données sont de très grandes tailles et ne réponds pas de manière optimale au problème d'*underfitting* (cf pénalité sur le coefficient *rho*).
Ainsi une piste d'amélioration serait de s'intéresser aux propriétés de l'estimateur LASSO, lequel contourne le problème du *sparse*. Plutôt que d'essayer de réduire la complexité du problème comme pour l'estimateur Ridge, celui-ci introduit directement des coefficients nuls.