DM Regression non-parametrique - Version finale

September 16, 2019

Alexandre Marette

1 Etude de la densité g des X

1.1 Construire un estimateur non-paramétrique g(n,h)(x) de g(x) pour une fenetre de lissage h >0 donnée et représenter graphiquement x-> g(n,h)(x) pour différentes valeurs de h que vous choisirez. On discutera la raison pour laquelle ce choix est important et cequi se produit si h est mal choisi.

[2]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from distutils.version import LooseVersion
from scipy.stats import norm
from sklearn.neighbors import KernelDensity

# `normed` is being deprecated in favor of `density` in histograms
if LooseVersion(matplotlib.__version__) >= '2.1':
    density_param = {'density': True}
else:
    density_param = {'normed': True}
```

On importe le fichier Data1.csv

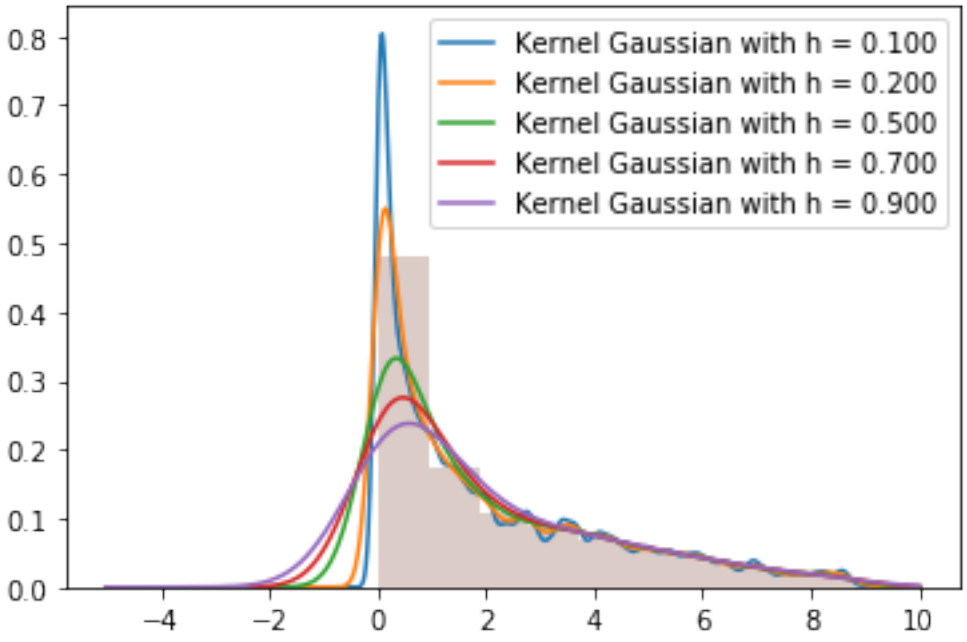
[3]:
```
#Import du fichier Data1.csv

filename='Data1.csv'
filepath = 'C:/Users/maret/OneDrive/Documents/Formation Science des Données/
 ↪Regression non parametrique/'

data1 = pd.read_csv(filepath + filename)
data1.head()

print(data1.count())
```

```
Unnamed: 0    2000
X             2000
```

```
Y1              2000
dtype: int64
```

[ ]:
```
On estimme la densité g(n,h)(x) pour différentes valeurs de la fenêtre de␣
 ↪lissage h. Des valeurs comprises entre 0,1 et 0,9.
Puis, on affiche les densités ainsi estimées en fonction de h.
```

[3]:
```
#Estimation de la densité g pour différentes valeurs de h
data1.X.head()
X=data1.X[:,np.newaxis]
Y1=data1.Y1[:,np.newaxis]
#print(Y)
fig, ax = plt.subplots()
X_plot = np.linspace(-5, 10, 2000)[:, np.newaxis]
#print(X_plot)

for h in [0.1,0.2,0.5,0.7,0.9]:
    kde = KernelDensity(kernel='gaussian', bandwidth=h).fit(X)
    log_dens = kde.score_samples(X_plot)
    ax.plot(X_plot[:,0], np.exp(log_dens), '-',label='Kernel Gaussian with h =␣
 ↪%.3f' % h)

ax.hist(X, density=True, histtype='stepfilled', alpha=0.3)
#ax.scatter(X,Y)
ax.legend(loc='upper right')
plt.show()
```



On observe que plus le h est petit et plus l'estimation de la densité semble se rapprocher de la densité g des X (semble précise), mais moins la courbe de l'estimateur de g semble régulière. Et inversement, lorsque l'on augmente h.

Cela conforte l'idée que nous devons prendre en compte 2 types d'erreur lors de l'estimation de la densité d'une variable : - le Biais : qui est l'erreur déterministe liée au fait que nous faisons une estimation. - la Variance : qui est l'erreur aléatoire Le Biais sera faible si h est petit et g régulière. La Variance sera d'autant plus faible que le nombre n de d'échantillon de X est grand.

Le choix de h est donc crucial pour un nombre d'observations données : - Si h est trop grand, l'estimateur est régulier, mais biaisé si h est trop petit, l'estimateur semble osciller (variance plus importante), mais le biais est plus faible.

1.2 Représenter graphiquement un estimateur g de x où hn est la fenêtre donnée par validation croisée ou par une autre méthode que l'on précisera

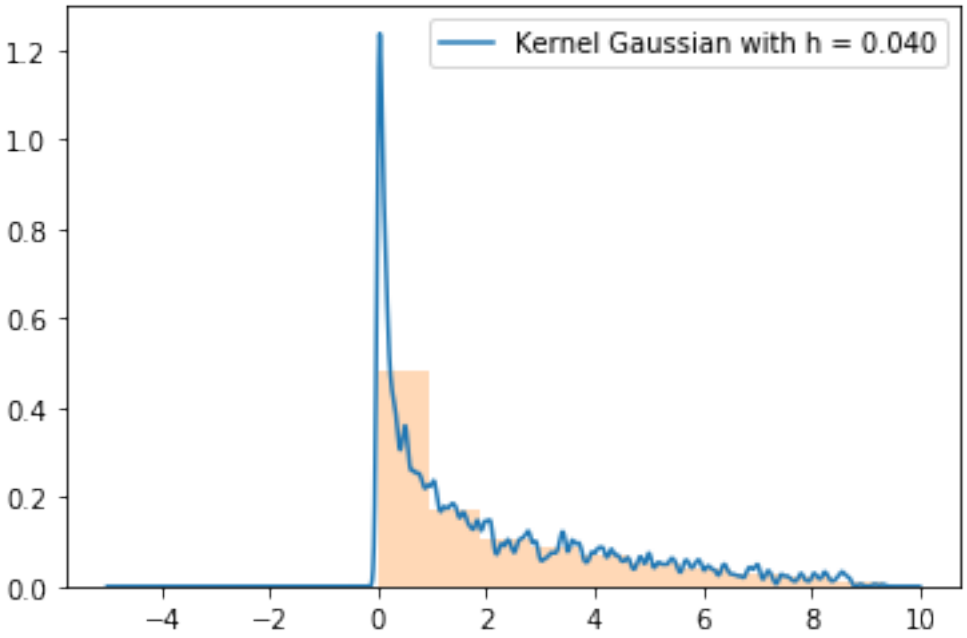
Avec le package scikit learn, nous utilisons la méthode GreadSearchCV pour trouver la fenêtre de lissage optimal. Il s'agit d'une méthode de validation croisée dans laquelle nous indiquons le paramètre à optimiser. Dans notre cas, il s'agit de h (= Bandwidth). Nous générons aléatoirement une série de 100 h différents que nous injectons dans la méthode GridSearchCV.

[4]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, make_scorer
from sklearn.model_selection import LeaveOneOut
#import pyqt_fit
#from pyqt_fit import kde

h_train=10**np.linspace(-2, 1, 100)

grid = GridSearchCV(KernelDensity(kernel="gaussian"),{'bandwidth': h_train},cv␣
 ↪= 10)
grid.fit(X)
h_cv=grid.best_params_["bandwidth"]
print('h_cv = %.3f' % h_cv)
```

```
h_cv = 0.040
```

Nous obtenons une fenêtre de lissage h = 0,04. Affichons la densité de l'estimateur avec cette fenetre en paramètre.

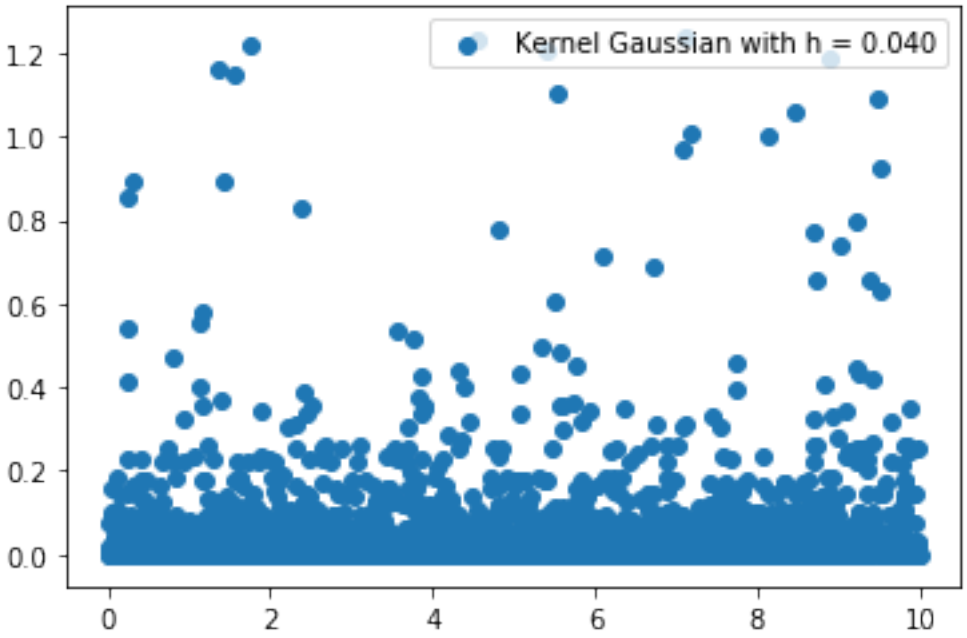
[6]:
```
fig, ax = plt.subplots()
kde_cv = KernelDensity(kernel='gaussian', bandwidth=h_cv).fit(X)
log_dens_cv = kde_cv.score_samples(X_plot)
ax.plot(X_plot[:,0], np.exp(log_dens_cv), '-',label='Kernel Gaussian with h = %.
 ↪3f' % h_cv)
ax.hist(X, density=True, histtype='stepfilled', alpha=0.3)
ax.legend(loc='upper right')
plt.show()
```



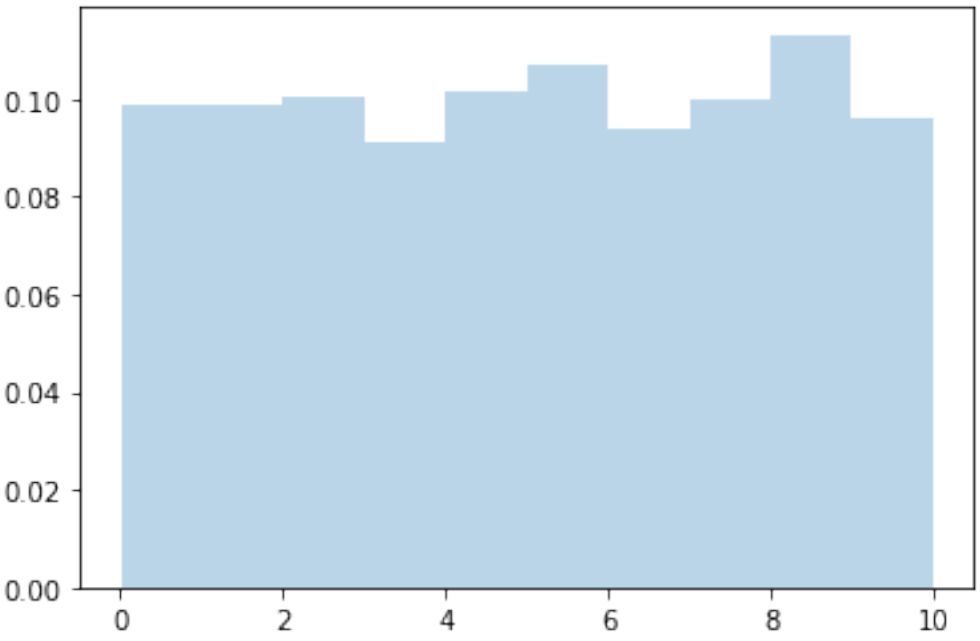
1.3 Implémenter un qq-plot pour vérifier empiriquement l'hypothèse g(x) = 1/10 pour tout x appartenant à [0,10]. L'hypothèse selon laquelle g est uniforme vous semble-t-elle raisonnable ?

Nous avons d'abord représenter empiriquement la densité d'une loi uniforme en générant 2000 points aléatoirement en utilisant la loi uniforme, puis en la représentant sous la forme d'un histogramme.

[22]:
```
fig, ax = plt.subplots()
X_Uni = np.random.uniform(0, 10, 2000)
ax.scatter(X_Uni, np.exp(log_dens_cv),label='Kernel Gaussian with h = %.3f' %␣
 ↪h_cv)
ax.legend(loc='upper right')
plt.show()
fig, ax = plt.subplots()
ax.hist(X_Uni, density=True, histtype='stepfilled', alpha=0.3)
plt.show
```



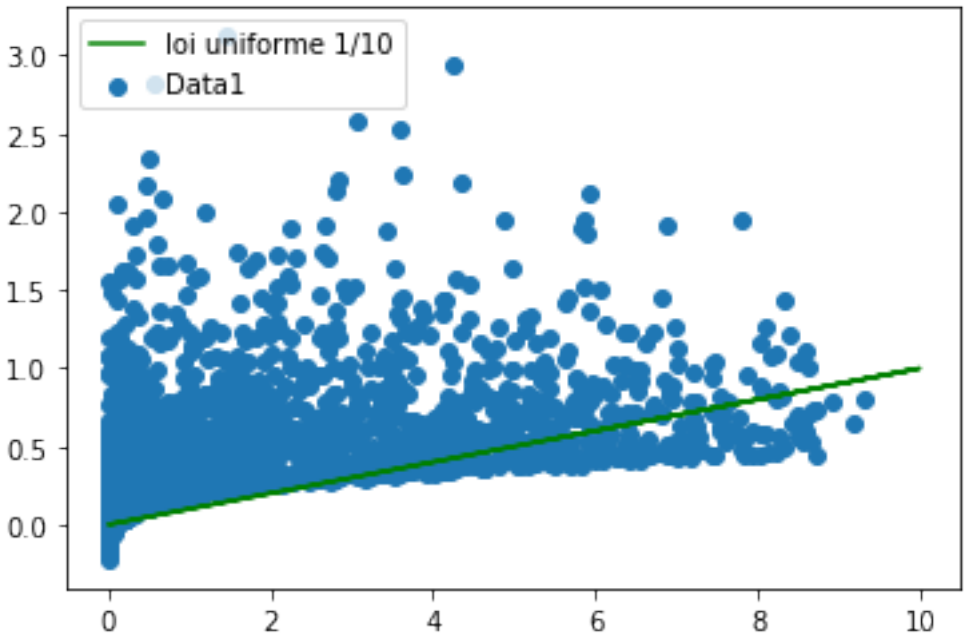
[22]: `<function matplotlib.pyplot.show(*args, **kw)>`



Nous avons ensuite tracer le le jeux de données Data1 (X en abscisse et Y1 en ordonnées), puis la fonction de répartition de la loi uniforme à partir des 2000 points générée précédemment.

[84]:
```
fig, ax = plt.subplots()
y=(1/10)*X_Uni
ax.scatter(X, Y,label='Data1')
ax.plot(X_Uni, y, '-',c='g',label='loi uniforme 1/10')
ax.legend(loc='upper left')
plt.show
```

[84]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



L'hypothèse selon laquelle g serait uniforme ne nous parait pas concluante au vue de l'observation du graphique ci-dessus. En effet, la loi uniforme semble trop linéaire et ne suit pas bien la concavité du nuage de points du jeux de données. L'erreur est importante proche de 0 et augmente lorsque X augmente.

1.4 Dans quelle zone de l'espace l'estimation de r sera plus précise ? Pourqoi ?

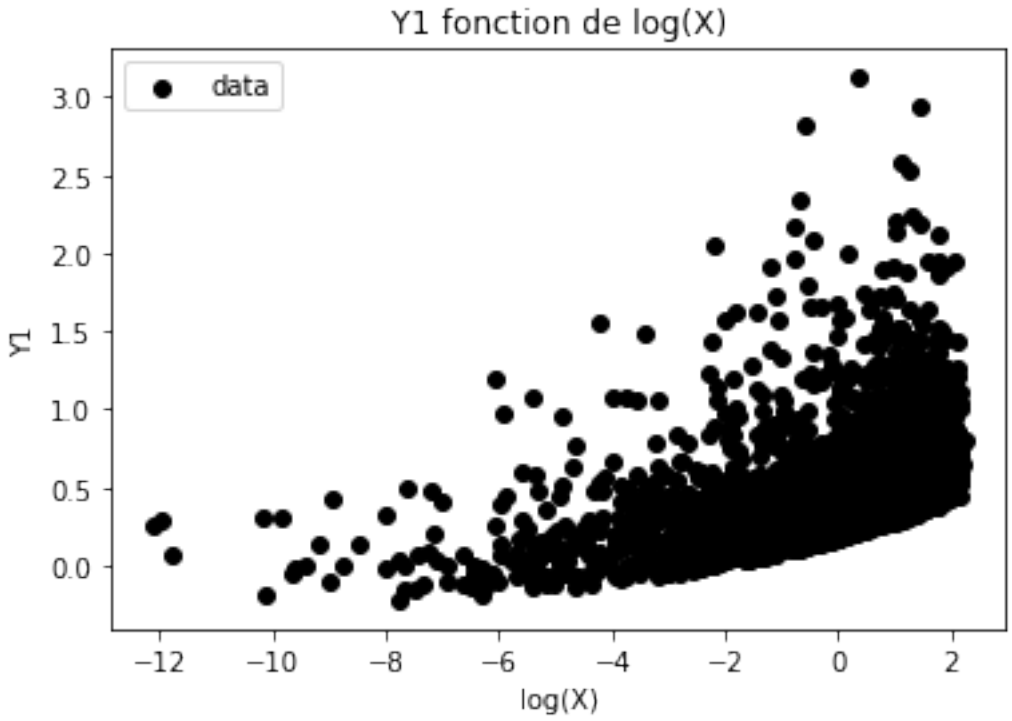
L'estimation de r sera plus précise dans les zones de plus forte concentration de données (de points). En effet, h dans ces zones de l'espace peut être petit, car la concentration de points est importante et il aura, donc, suffisamment de points pour que la régression soit efficace. Plus h est petit, plus l'estimation est précise comme nous en avons discuté plus haut.

2 Reconstruction de r(x)

2.1 Est-il plausible de penser que la fonction r est linéaire ? Tracer Y1 en fonction de log(X), que remarque-t-on ?

Au regard de la représentation de Y1 en fonction de X plus haut, il semble peut probable que la fonction r soit linéaire. En effet, elle semble concave et avoir une forte concentration de point entre les abscisses 0 et 2.

[8]:
```
from pylab import log

plt.scatter(log(X), Y1, c='k', label='data')

plt.xlabel('log(X)')
plt.ylabel('Y1')
plt.title('Y1 fonction de log(X)')
plt.legend()
plt.show()
```



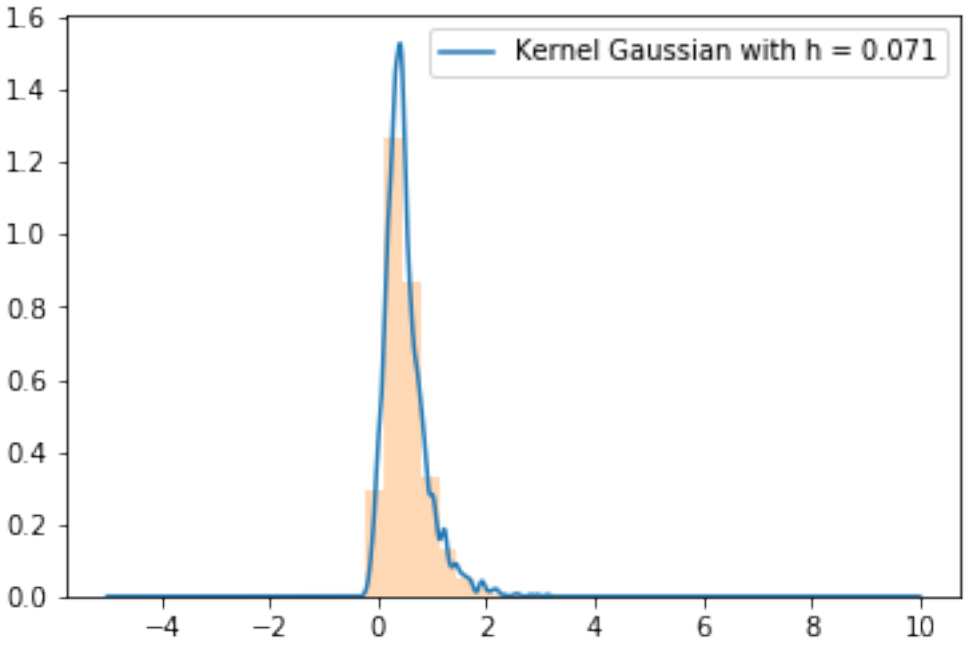
Le fait de passer par log(X) semble avoir linéarisé la fonction de lien entre Y1 et X.

2.2 Construire un estimateur non-paramétrique r(n,h)(x) de r(x) pour une femêtre de lissage h>0 bien choisie et le représenter graphiquement.

Nous avons d'abord estimé et affiché la densité de Y1.

[34]:
```
# Estimation de la densité de Y1

#Estimation de h
h_r_train=10**np.linspace(-2, 1, 100)
Y1_plot = np.linspace(-5, 10, 2000)[:, np.newaxis]

grid = GridSearchCV(KernelDensity(kernel="gaussian"),{'bandwidth':␣
 ↪h_r_train},cv = 10)
grid.fit(Y1,X)
h_r_cv=grid.best_params_["bandwidth"]
print('h_r_cv = %.3f' % h_r_cv)


# Affichage
fig, ax = plt.subplots()
kde_cv = KernelDensity(kernel='gaussian', bandwidth=h_cv).fit(Y1)
log_dens_r_cv = kde_cv.score_samples(Y1_plot)
ax.plot(Y1_plot[:,0], np.exp(log_dens_r_cv), '-',label='Kernel Gaussian with h␣
 ↪= %.3f' % h_r_cv)
ax.hist(Y1, density=True, histtype='stepfilled', alpha=0.3)
ax.legend(loc='upper right')
plt.show()
```

```
h_r_cv = 0.071
```

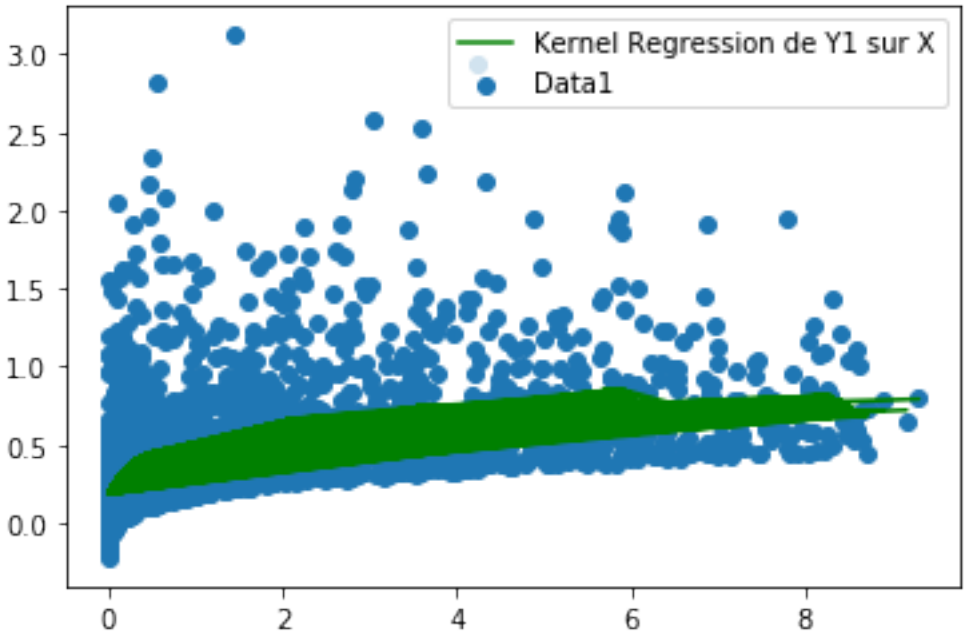


Puis nous avons construit un estimateur de r en régressant Y1 sur X, en sélectionnant la fenêtre de lissage par une méthode de validation croisée utilisant la méthode de Nadaraya-Watson et en minimisant l'erreur = [Y1 - estimateur(r(X))]^2 (least squared error).

[12]:
```
import statsmodels
from statsmodels.nonparametric.kernel_regression import KernelReg

kr_est1 = KernelReg(Y1, X,'c',bw='cv_ls')

pred_est1=kr_est1.fit(X)
h=kr_est1.bw
```

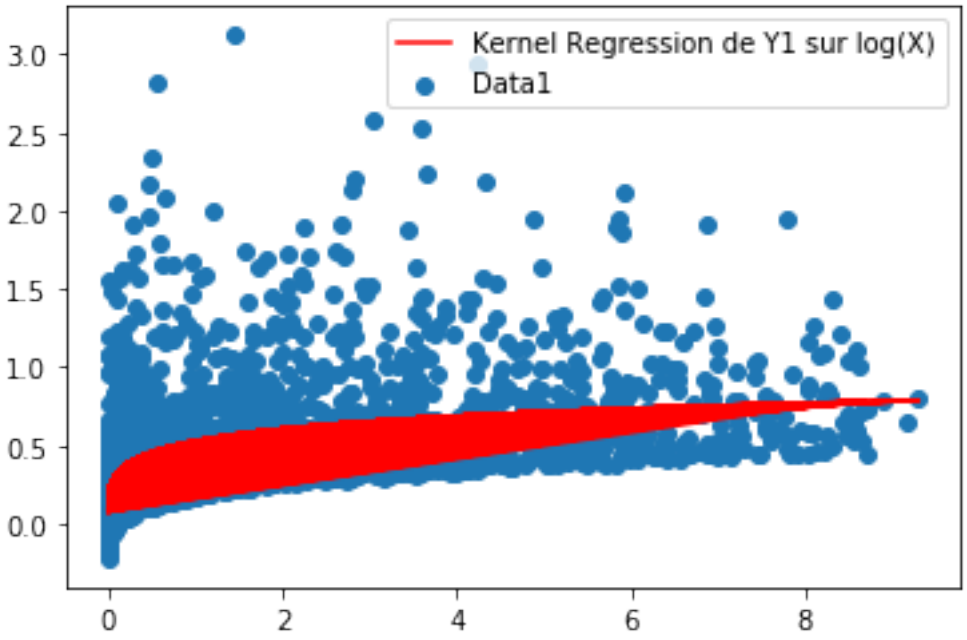
Affichage ci-dessous de l'estimateur de r(x) obtenu.

[14]:
```
# Affichage
fig, ax = plt.subplots()
ax.scatter(X, Y1,label='Data1')
ax.plot(X, pred_est1[0],'-', c='g', label='Kernel Regression de Y1 sur X' )
ax.legend(loc='upper right')
plt.show()
```

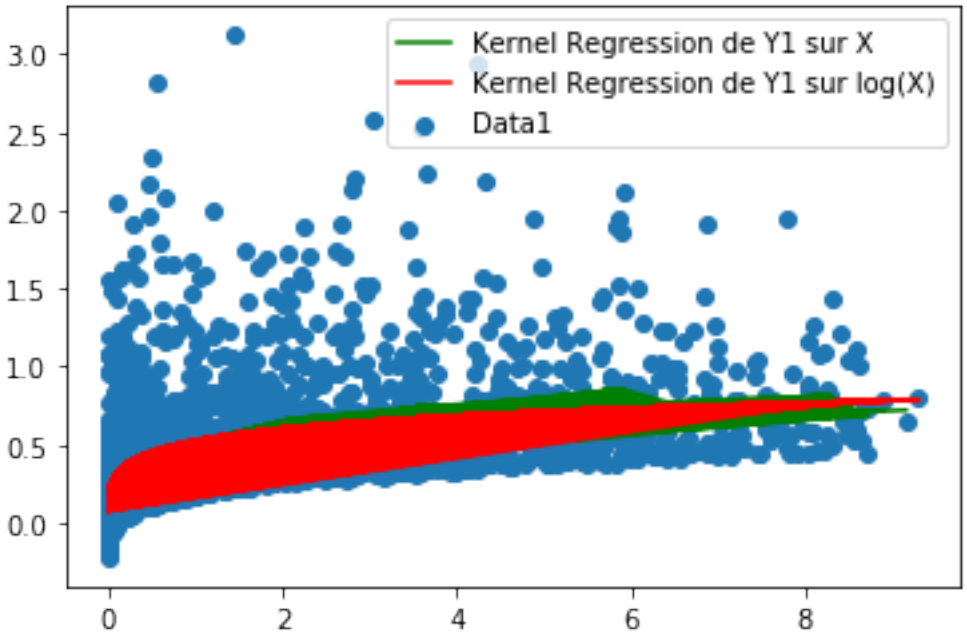


2.3 On se propose maintenant d'estimer r en régressant Y1 sur log(X). Construire un estimateur non_parametrique de r(x) dans le modèle de Y1 = ~r(log(X)) + , pour une fenêtre de lissage h>0 bien choisie. Superposer sur le graphe précédent le nouvel estimateur.

[15]:
```
# Construction de l'estimateur
kr_est2 = KernelReg(Y1, log(X),'c',bw='cv_ls')
pred_est2=kr_est2.fit(log(X))

# Affichage
fig, ax = plt.subplots()
ax.scatter(X, Y1,label='Data1')
ax.plot(X, pred_est2[0],'-', c='r', label='Kernel Regression de Y1 sur log(X)' )
ax.legend(loc='upper right')
plt.show()
```



[16]:
```
# Superposition des deux graphs :

fig, ax = plt.subplots()
ax.scatter(X, Y1,label='Data1')
ax.plot(X, pred_est1[0],'-', c='g', label='Kernel Regression de Y1 sur X' )
ax.plot(X, pred_est2[0],'-', c='r', label='Kernel Regression de Y1 sur log(X)' )
ax.legend(loc='upper right')
plt.show()
```



2.4 Que remarque-t-on ? Comment peut-on l'expliquer ?

L'estimateur basé sur la régression de Y1 sur log(X)semble plus linéaire, plus lisse. Utiliser le logarythme sur des séries de données les lisse, linéarise des relations multiplicatives entre des variables.

3 Etude à partir de la densité des i

3.1 A partir du jeu de données Data1

3.1.1 On cherche à estimer x -> (x).

Pour cela on coupe l'échantillon en 2 échantillon de 1000 chacun.

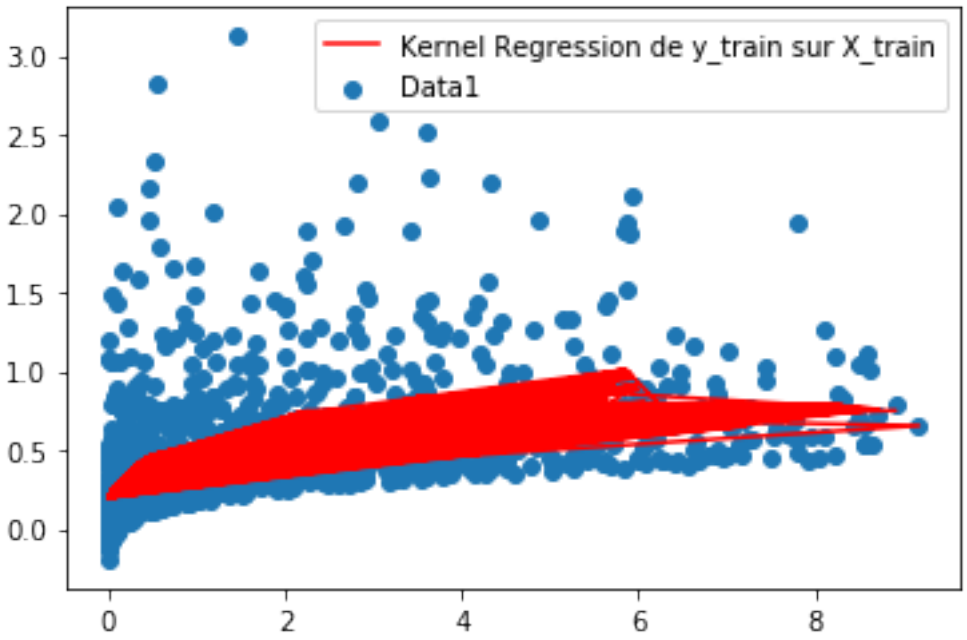
[42]:
```
# Split dees données en 2 échantillons
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y1, train_size= 1000)
```

L'espérance d'i est égal à 0 et sa variance égale à 1. Il semblerait que i suive une loi normale centrée réduite.

3.1.2 En déduire un estimateur de x -> (x) et l'implémenter graphiquement

a) On estime par une régression en reprenant le h établi à la question 2.2 et en utilisant les échantillons d'entrainement.

[43]:
```
ktrain = KernelReg(y_train, X_train,'c',bw=h)

pred_train=ktrain.fit(X_train)


# Affichage de la régression :

fig, ax = plt.subplots()
ax.scatter(X_train, y_train,label='Data1')
ax.plot(X_train, pred_train[0],'-', c='r', label='Kernel Regression de y_train␣
 ↪sur X_train' )
ax.legend(loc='upper right')
plt.show()
```

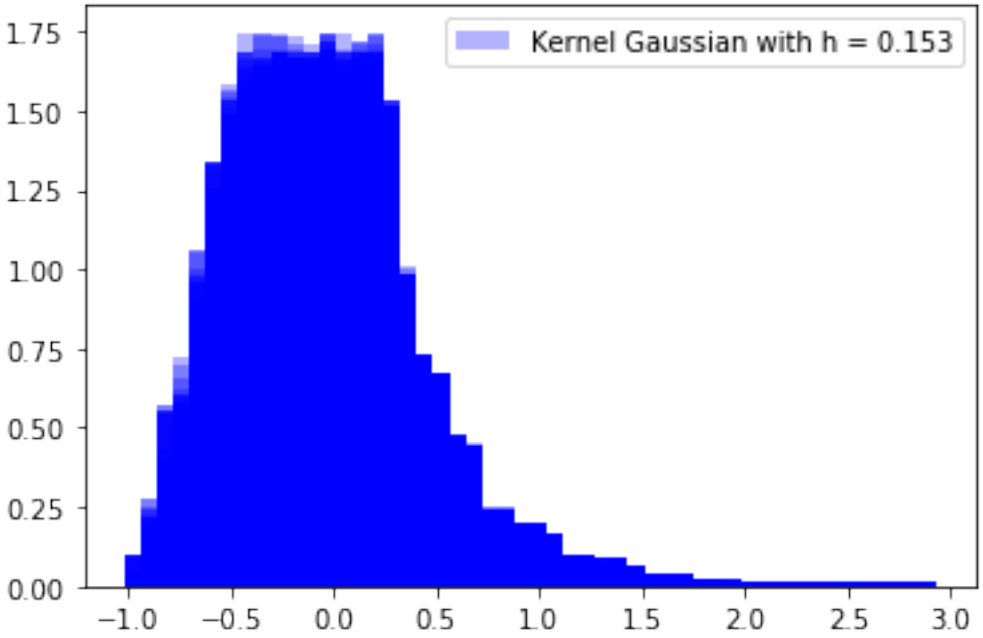


b) On estime empiriquement la densité des résidus i :

[40]:
```
res = (y_train - pred_train[0])#[:,np.newaxis]

X_plot = np.linspace(-5, 10, 1000)[:, np.newaxis]

print('h = %.3f' % h)


# Affichage
fig, ax = plt.subplots()
```

```
kde_cv = KernelDensity(kernel='gaussian', bandwidth=h).fit(res)
#log_dens_res_cv = kde_cv.score_samples(X_plot)
#ax.plot(X_plot[:,0], np.exp(log_dens_res_cv), '-',label='Kernel Gaussian with␣
 ↪h = %.3f' % h)
ax.hist(res,50, density=True, histtype='stepfilled',facecolor='b', alpha=0.
 ↪3,label='Kernel Gaussian with h = %.3f' % h)
ax.legend(loc='upper right')
plt.show()
```

```
h = 0.153
```



Construction, estimation et implémentation graphique de l'estimateur de x -> (x) :

3.1.3 Quel est l'intéret d'avoir découpé le jeu de données ?

C'est un moyen de prédire l'efficacité du modèle ou de l'estimateur sur un ensemble de validation lorsqu'un ensemble de validation indépendant et explicite n'est pas disponible.

3.1.4 La densit x->(x) peut-elle etre gaussienne ? Proposer un protocole pour le vérifier empiriquement et l'implémenter.

Effectivement, elle pourrait être gaussienne si le design est déterministe et que les i sont i.i.d et suivent une loi normale d'espérance = 0 et de variance = ^2.

Si nous sommes dans le cadre d'un modèle Gaussien, nous savons expliciter la loi des observations (Y1,...,Yn) : P = (1/sqrt($2^{2)}$)$^{exp((1/(2\hat{}2))(yr(,xi))}$2)

Il faudra maximiser la fonction de vraissemblance qui est une fonction exponentiel dans notre cas. Cela revient à minimiser la somme des carrés suivantes : Somme (de i=1 à n) de (Yi - r(,xi))^2

Cela revient à minimiser le carré des i.

3.1.5 (Facultatif.)Comment peut-on tester si le modèle est bien homoscédastique ?

On parle d'homoscédasticité lorsque la variance des erreurs stochastiques de la régression est la même pour chaque observation i. Il faut étudier la variance des résidus (i). Il existe différents tests dont le test de Test de Breusch-Pagan pour une régression linéaire.

3.2 A partir du jeux de données Data2