# Lab 1 - Régression Linéaire et Polynomiale sur des données non-linéaires

Ce laboratoire est une introduction au **Machine Learning** sur exemple de la régression linéaire et polynomiale.

Nous allons utiliser un environnement Jupyter et le langage Python.

Pour la facilité du travail nous pouvons utiliser Google Colab , mais l'environnement Jupyter est également installé sur nos cartes Jetson Nano.

## 1.1 Régression linéaire

La première phase de travail dans un projet de Machine Learning est la préparation des données.

Dans notre cas nous allons générer un ensemble de données non-linéaires par injection d'une composante aléatoire.

Sur cet ensemble nous allons entraîner notre modèle et finalement nous allons visualiser le résultat.

1. Obtenir X et y à partir de la fonction **dataset**()
2. Entraînez un modèle de régression linéaire pour cet ensemble de données.
3. Visualisez la prédiction du modèle

Nous commençons par l'importation de **`numpy`** et **`matplotlib`**

```
import numpy as np
import matplotlib.pyplot as plt
```

### 1.1.1 Création et affichage du dataset

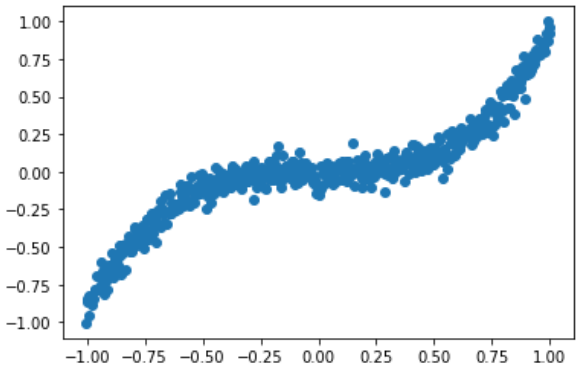
Nous allons créer un ensemble de **`500`** valeurs X et y. Les valeurs de y sont générées par la fonction **`y = X**3 + 20 + np.random.randn(500)*1000`** qui ajoute un **`élément`** aléatoire **`randn(500)`** – distribution normale pour 500 valeurs.

```
def dataset(show=True):
X = np.arange(-25, 25, 0.1)
# Try changing y to a different function
y = X**3 + 20 + np.random.randn(500)*1000
if show:
      plt.scatter(X, y)
      plt.show()
return X, y


X, y = dataset()
```

### 1.1.2 Mise à l'échelle et affichage du jeu de données

La mise à l'échelle est nécessaire pour réduire et simplifier le calcul de coefficients.

```
print(max(X), max(y), min(X), min(y))
X = X/max(X)
y = y/max(y)
print(max(X), max(y), min(X), min(y))
plt.scatter(X, y)
plt.show()
```




### 1.1.3 Création du modèle séquentiel pour la régression linéaire

Pour créer un modèle de la régression linéaire nous avons besoin de la librairie **keras**. Notre modèle est construit sur une couche **Dense**.

Les données d'entrée sont portées par un vecteur d'une dimension.

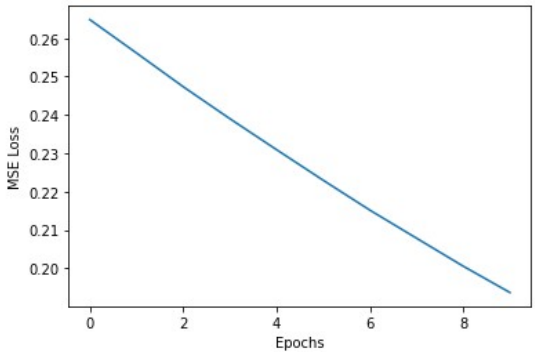
```
import tensorflow as tf
from tensorflow import keras
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
```

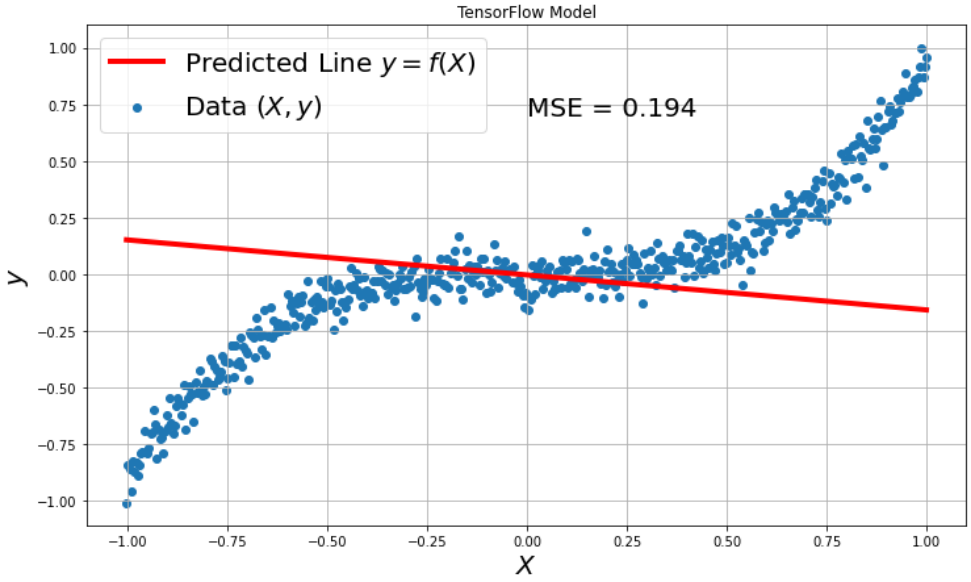
### 1.1.4 Compilation et entraînement du modèle (régression lineaire)

Dans la phase suivante nous allons compiler et entraîner notre modèle. L'optimiseur choisi est **Adam** et le taux d'apprentissage est **0,001**. Le nombre d'époques proposé est **10**, il peut être modifié pour rendre le modèle plus précis.

```
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
model.compile(optimizer=optimizer, loss='mean_squared_error')
model.summary()
tf_history = model.fit(X, y, epochs=500, verbose=False)
#tf_history = model.fit(X, y, epochs=10, verbose=True)
plt.plot(tf_history.history['loss'])
plt.xlabel('Epochs')
plt.ylabel('MSE Loss')
plt.show()
mse = tf_history.history['loss'][-1]
y_hat = model.predict(X)
plt.figure(figsize=(12,7))
plt.title('TensorFlow Model')
plt.scatter(X, y, label='Data $(X, y)$')
plt.plot(X, y_hat, color='red', label='Predicted Line $y = f(X)
$',linewidth=4.0)
plt.xlabel('$X$', fontsize=20)
plt.ylabel('$y$', fontsize=20)
plt.text(0,0.70,'MSE = {:.3f}'.format(mse), fontsize=20)
plt.grid(True)
plt.legend(fontsize=20)
plt.show()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
```

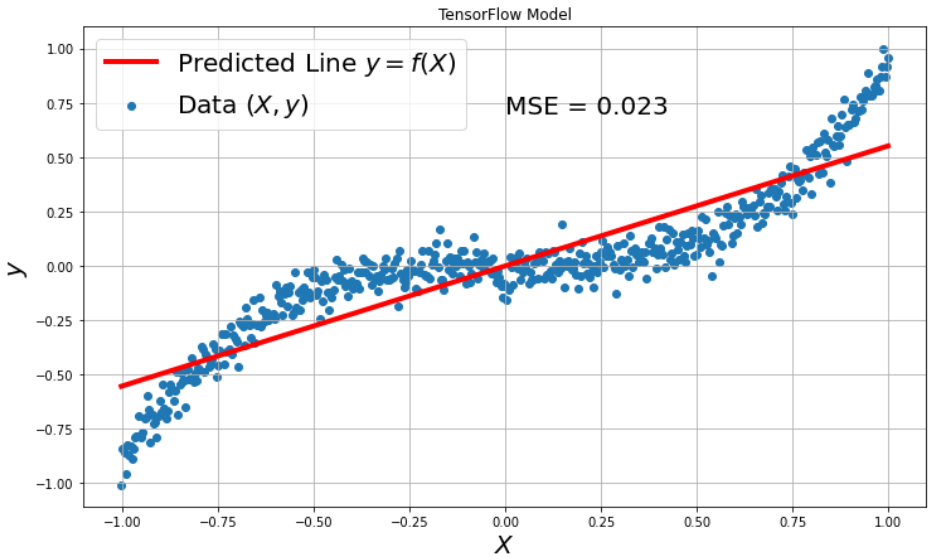
Notons que le historique de l'entraînement est mémorisé dans  vecteur **`tf_history`**. Le contenu de ce vecteur est ensuite utilisé pour l'affiche de la perte calculée avec MSE.




Le résultat affiché n'est pas très probant. Essayons avec le nombre d'époques plus élevé, par exemple **500**.

Et voici le résultat :



## 1.2 Régression polynomiale (2ème degré)

Lorsque le jeu de données n'est pas linéaire, la régression linéaire ne peut pas apprendre le jeu de données et faire de bonnes prédictions. Nous avons besoin ici d'un modèle polynomial qui considère également les termes polynomiaux.

Nous pouvons calculer les caractéristiques polynomiales pour chaque caractéristique par la programmation ou nous pouvons essayer **`sklearn.preprocessing.PolynomialFeatures`** qui nous permet de créer des termes polynomiaux de nos données.

### 1.2.1 Préparation du dataset

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np

def dataset(show=True):
X = np.arange(-25, 25, 0.1)
# Try changing y to a different function
y = X**3 + 20 + np.random.randn(500)*1000
if show:
plt.scatter(X, y)
plt.show()
return X, y

X, y = dataset(show=False)
X_scaled = X/max(X)
y_scaled = y/max(y)
```

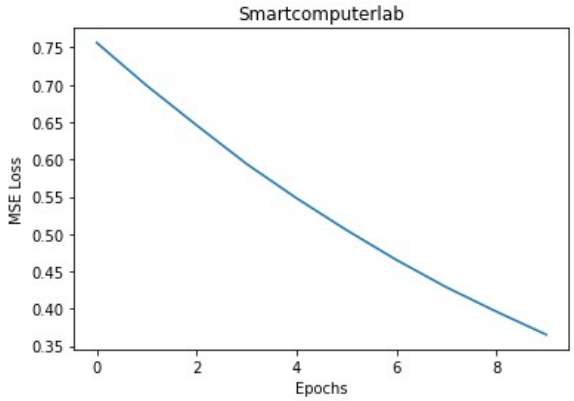
### 1.2.2 Création et l'entraînement du modèle

$$y=a*x^2+b*x^1+c*x^0$$

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)

X_2 = poly.fit_transform(X_scaled.reshape(-1,1))
print(X_2.shape)
print(X_2[0])

model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[3])])
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
model.compile(optimizer=optimizer, loss='mean_squared_error')
model.summary()

#tf_history = model.fit(X_2, y_scaled, epochs=500, verbose=False)
tf_history = model.fit(X_2, y_scaled, epochs=10, verbose=True)

plt.plot(tf_history.history['loss'])
plt.xlabel('Epochs')
plt.ylabel('MSE Loss')
plt.title('Smartcomputerlab')
plt.show()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_2 (Dense)              (None, 1)                 4
```

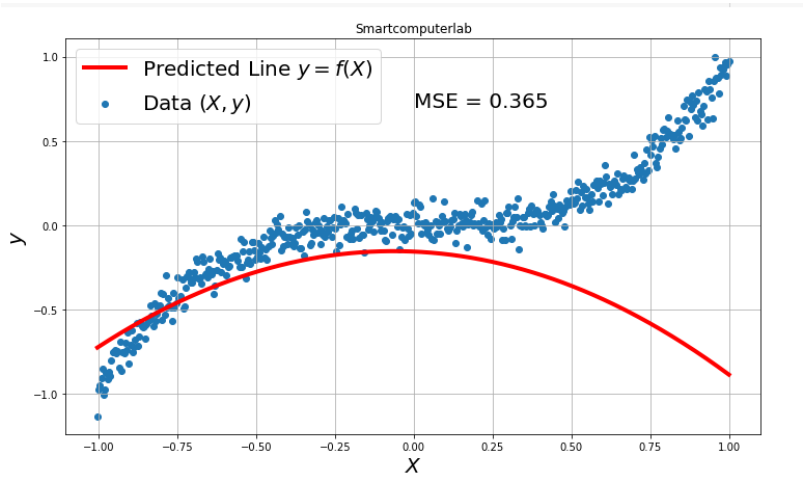
```
=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
```

Voici le résultat d'entraînement pour **10 époques** (perte MSE) ?

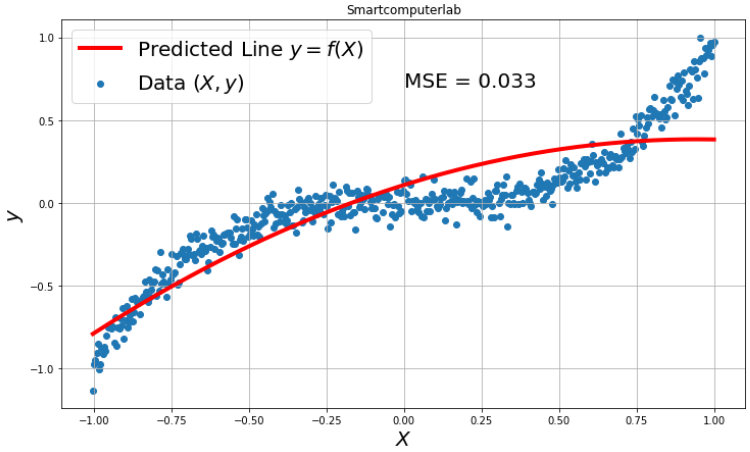



Et l'affichage de la courbe - pas très satisfaisant avec MSE élevée de **0.365**

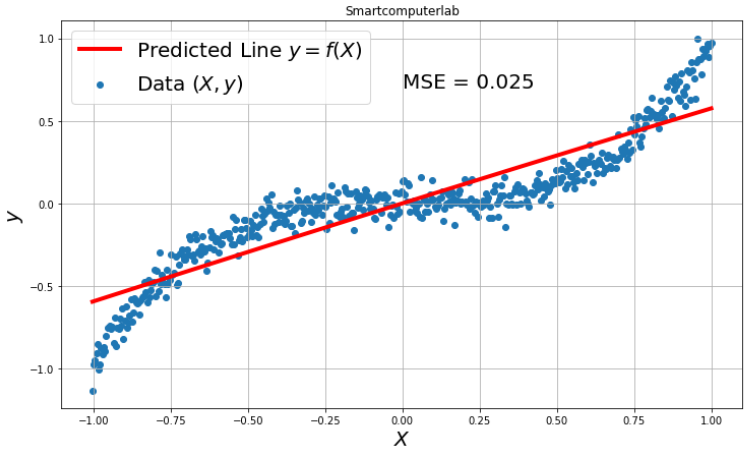
```
mse = tf_history.history['loss'][-1]
y_hat = model.predict(X_2)

plt.figure(figsize=(12,7))
plt.title('Smartcomputerlab')
plt.scatter(X_2[:, 1], y_scaled, label='Data $(X, y)$')
plt.plot(X_2[:, 1], y_hat, color='red', label='Predicted Line $y = f(X)
$',linewidth=4.0)
plt.xlabel('$X$', fontsize=20)
plt.ylabel('$y$', fontsize=20)
plt.text(0,0.70,'MSE = {:.3f}'.format(mse), fontsize=20)
plt.grid(True)
plt.legend(fontsize=20)
plt.show()
```



Essayons plus d'époques d'entraînement – **100**.




Essayons plus d'époques d'entraînement – 500.



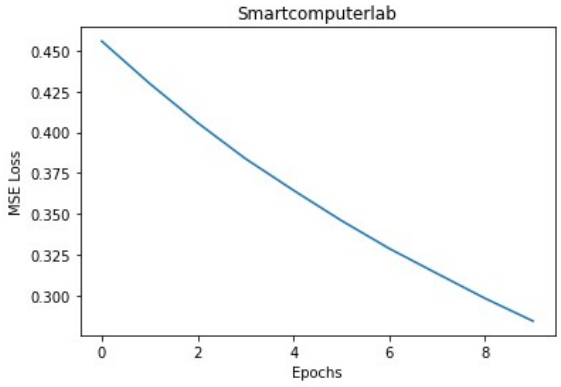

**Conclusion est :**

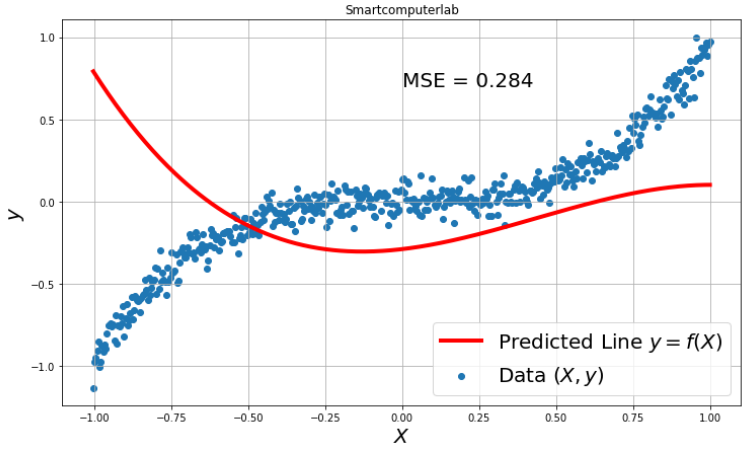
## 1.3 Régression polynomiale (3ème degré)

Voici le code complet pour le modèle polynomial de 3ème degré.

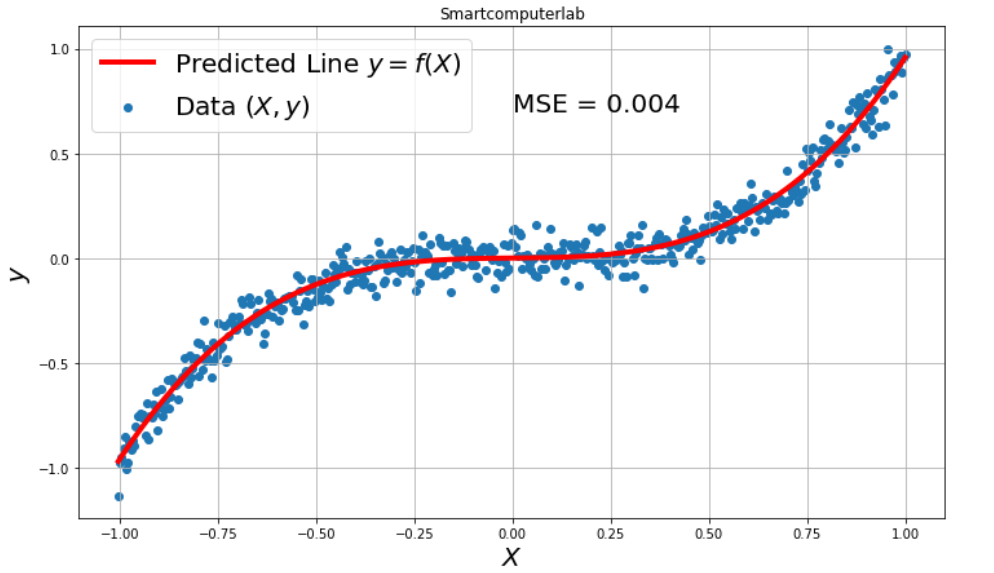
```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=3)
X_3 = poly.fit_transform(X_scaled.reshape(-1,1))
print(X_3.shape)
print(X_3[0])

model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[4])])
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)

model.compile(optimizer=optimizer, loss='mean_squared_error')
model.summary()

tf_history = model.fit(X_3, y_scaled, epochs=10, verbose=False)
plt.plot(tf_history.history['loss'])
plt.xlabel('Epochs')
plt.ylabel('MSE Loss')
plt.title('Smartcomputerlab')
plt.show()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_5 (Dense)              (None, 1)                 5
=================================================================
Total params: 5
Trainable params: 5
Non-trainable params: 0
```




Le modèle de 3ème degré entraîné avec 10 époques.




Le modèle de 3ème degré entraîné avec 10 époques, quasiment parfait , pourquoi ?

## 1.4 Régression polynomiale (4ème degré)

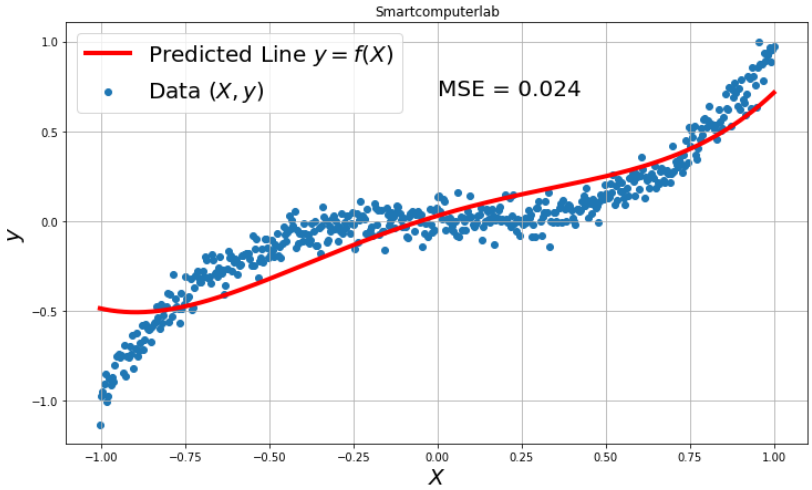
Pour finir nos exercices essayons encore un polynôme de 4ème degré.

Voici le code:

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=4)
X_4 = poly.fit_transform(X_scaled.reshape(-1,1))
print(X_4.shape)
print(X_4[0])

model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[5])])
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)

model.compile(optimizer=optimizer, loss='mean_squared_error')
model.summary()
tf_history = model.fit(X_4, y_scaled, epochs=100, verbose=False)

plt.plot(tf_history.history['loss'])
plt.xlabel('Epochs')
plt.ylabel('MSE Loss')
plt.title('Smartcomputerlab')
plt.show()
```

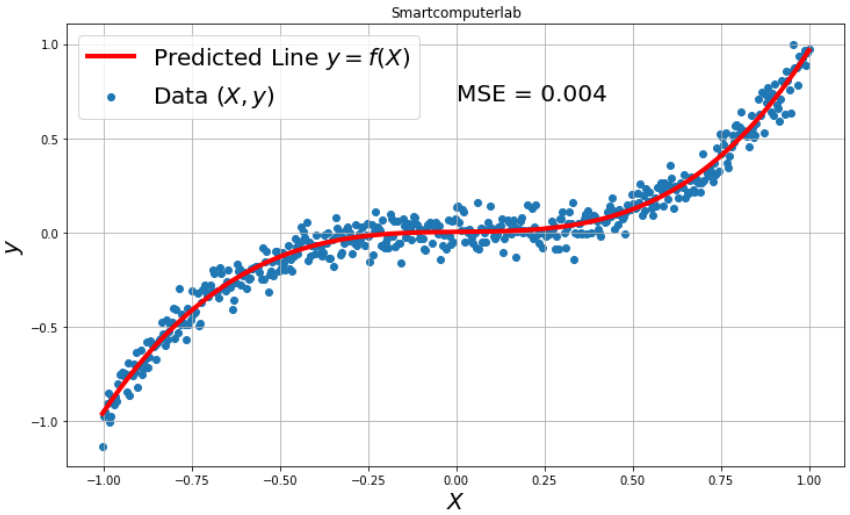
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 1)                 6
=================================================================
Total params: 6
Trainable params: 6
Non-trainable params: 0
```

```
mse = tf_history.history['loss'][-1]
y_hat = model.predict(X_4)

plt.figure(figsize=(12,7))
plt.title('Smartcomputerlab')
plt.scatter(X_4[:, 1], y_scaled, label='Data $(X, y)$')
plt.plot(X_4[:, 1], y_hat, color='red', label='Predicted Line $y = f(X)
$',linewidth=4.0)
plt.xlabel('$X$', fontsize=20)
plt.ylabel('$y$', fontsize=20)
plt.text(0,0.70,'MSE = {:.3f}'.format(mse), fontsize=20)
plt.grid(True)
plt.legend(fontsize=20)
plt.show()
```




Ci-dessus , le résultat du modèle entraîné pendant 100 époques.

Ci-dessous pour le modèle entraîné pendant 1000 époques ; **conclusion ?**



## 1.5 Sujet final

Dans le sujet final il faut développer une application (code) avec les arguments permettant le paramétrage du code.

Par exemple nous pouvons saisir le degré du polynôme comme en paramètre:

```
degre=int(input("Donne le degre du polynôme [2,3,4,5,..]: "))
learning_rate=float(input("Donne le taux d'apprentissage [1e-1,1e-2,1e-3,..]:
"))

print(degre)
print(learning_rate)
```