Objectifs du TP :
- implémenter et analyser l'algorithme de régression linéaire par la méthode des moindres carrés,
- utiliser `scikit-learn` pour tester et comparer deux algorithmes de régression régularisée, Ridge et Lasso, extension de la régression par moindres carrés,
- tester les algorithmes sur des données simulées et des données réelles.

1 Régression linéaire par moindres carrés

L'algorithme de régression par la méthode des moindres carrés, dans sa version standard, est un algorithme de régression *linéaire*. Les données d'apprentissage utilisées dans cette section s'écrivent sous la forme $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ avec $\mathbf{x}_i \in \mathbb{R}^d$ (d est le nombre d'attributs des données d'entrée) et $y_i \in \mathbb{R}$.

À partir des données $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, l'objectif est d'apprendre un vecteur $\mathbf{w} = w_0, w_1, \ldots, w_d$ tel que $y_i \approx w_0 + \langle w_{1:d}, \mathbf{x}_i \rangle$ pour tout $(\mathbf{x}_i, y_i) \in S$ où la notation $\langle \mathbf{u}, \mathbf{v} \rangle := \sum_{j=1}^d u_j v_j$ désigne le produit scalaire entre les deux vecteurs $\mathbf{u}$ et $\mathbf{v}$.

L'algorithme de régression linéaire par la méthode des moindres carrés cherche à minimiser le risque empirique de l'échantillon d'apprentissage, c'est-à-dire à résoudre le problème d'optimisation suivant :

$$\text{Trouver le vecteur } \mathbf{w} \in \mathbb{R}^{\mathbf{d+1}} \text{ qui minimise } \sum_{i=1}^n \big(y_i - w_0 - \langle \mathbf{w}_{1:d}, x_i \rangle\big)^2.$$

Une forme analytique de la solution de ce problème est donnée par la formule suivante :

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y},$$

où $\mathbf{X} \in \mathbb{R}^{\mathbf{n \times (d+1)}}$ est la matrice contenant les données $(x_i)_{i=1}^n$, complétée par une première colonne de 1, et où $\mathbf{y} \in \mathbb{R}^n$ est le vecteur $(y_1, \ldots, y_n)^\top$.

Algorithme Régression linéaire par moindres carrés

Entrée : une liste S de données d'apprentissage, $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ où $\mathbf{x}_i \in \mathbb{R}^d$ et $y_i \in \mathbb{R}$,
Sortie : le vecteur de pondération w.

---

1. Ajouter le vecteur $\mathbf{1}$ à la matrice $\mathbf{X} \in \mathbb{R}^{\mathbf{n \times d}}$ contenant les données $(\mathbf{x}_i)_{i=1}^n$,
2. Calculer $\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$, uù $\mathbf{y} = (y_1, \ldots, y_n)^\top$
3. Retourner $\mathbf{w}$.

---

1) Implémentez l'algorithme de régression linéaire par moindres carrés décrit ci-dessus. Vous pourrez avoir à utiliser les commandes rappelées ci-dessous, et tester votre algorithme sur les données générées par la fonction `GenData`.

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

# commandes utiles
X = np.zeros([5,3]) # tableau de 0 de dimension 5x3
Y = np.ones([3,2]) # tableau de 1 de dimension 5x3
v = np.ones(3) # vecteur contenant trois 1
```

```
X[1:4,:2] = Y # remplacement d'une partie du tableau X
X.shape # dimensions de X
np.random.rand(10) # 10 nombres aléatoires entre 0 et 1
Z = np.random.random([4,4]) # matrice aléatoire
np.random.normal(0,1,10) # 10 nombres aléatoires générés par la Gaussienne N(0,1)
np.dot(X,Y) # produit matriciel
np.dot(X,v) # produit de la matrice X et du vecteur v
np.transpose(X) # transposée de X
np.linalg.inv(Z) # inverse de Z

def GenData(x_min, x_max, w, nbEx, sigma):
    """ génère aléatoirement n données du type (x,w0 + <w_1:n,x> + e) où
    - w est un vecteur de dimension d + 1
    - x_min <= |x_i| <= x_max pour les d coordonnées x_i de x
    - e est un bruit Gaussien de moyenne nulle et d'écart type sigma
    Retourne deux np.array de forme (nbEx,1)
    """
    d = len(w) - 1
    X = (x_max-x_min)*np.random.rand(nbEx,d) + x_min
    Y = np.dot(X,w[1:]) + w[0] + np.random.normal(0,sigma,nbEx)
    Y = Y.reshape(nbEx,1)
    return X,Y

def AddOne(X):
    """ X est un tableau n x d ; retourne le tableau n x (d+1) consistant à rajouter une colonne d
    (n,d) = X.shape
    Xnew = np.zeros([n,d+1])
    Xnew[:,1:]=X
    Xnew[:,0]=np.ones(n)
    return Xnew

def RegLin(X,Y):
    """ X est un tableau n x d ; Y est un tableau de dimension n x 1
    retourne le vecteur w de dimension d+1, résultat de la régression linéaire """
    Z = AddOne(X)
    return np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(Z),Z)),np.transpose(Z)),Y[:,0])

def RSS(X,Y,w):
    """ Residual Sum of Squares  """
    v = Y[:,0]- (np.dot(X,w[1:]) + w[0])
    return np.dot(v,v)

x_min = -5
x_max = 5
nbEx = 10
sigma = 1.0

w = np.array([-1,2])

X,Y = GenData(x_min, x_max, w,nbEx,sigma)
print(X.shape,Y.shape)
w_estime = RegLin(X,Y)
```

```
print(RSS(X,Y,w),RSS(X,Y,w_estime))

plt.scatter(X,Y)
plt.plot([x_min,x_max],[w[1]*x_min + w[0],w[1]*x_max + w[0]], label = 'Cible')
plt.plot([x_min,x_max],[w_estime[1]*x_min + w_estime[0],w_estime[1]*x_max + w_estime[0]], label =
plt.legend()
plt.show()
```

2) Exécutez les lignes de code suivantes, commentez ce que vous observez et dites à quelle propriétés de la régression elles font référence.

```
w = np.array([-1,2,-1,3])

w_mean = np.zeros(4)
for i in range(10):
    X,Y = GenData(min, max, w, nbEx, sigma)
    w_mean += RegLin(X,Y)
w_mean = w_mean/10
print(w,w_mean)

nbEx = 1000
X,Y = GenData(min, max, w, nbEx, sigma)
w_estime = RegLin(X,Y)
print(w,w_estime)
```

3) Le fichier texte `TP3.data1` contient un jeu de données : chaque ligne décrit un exemple de la forme x, y. Utilisez la commande `loadtxt` de `numpy` pour accéder aux données sous python : `Z = np.loadtxt("./TP3.data1")`.

3.1) Affichez les données sur un graphique 2D au moyen de la commande `scatter` de `pyplot`.

3.2) Écrivez la fonction `poly(x,d)` qui, pour tout réel x et tout entier $d \geq 1$ retourne la liste $[x, x^2, \ldots, x^d]$. Écrivez la fonction `polyTab(X,d)` qui, pour tout vecteur X de longueur N et tout entier $d \geq 1$ retourne un tableau $N \times d$ dont la ligne i est égale à $[X[i], X[i]^2, \ldots, X[i]^d]$. On utilisera cette transformation pour réaliser une régression polynomiale sur les données.

3.3) Affichez les polynomes de régression calculés à partir de ces données pour un degré $1 \leq d \leq 10$. Tracez ces 10 fonctions polynomes sur le même graphique, en superposition des données. Indication : pour tracer un polynome de coefficients donnés, vous pouvez vous inspirer du code suivant :

```
abscisses = np.linspace(min, max, 50)
w = [1, 2, 3]
ordonnées = [np.dot([1,x,x*x],w) for x in abscisses]
plt.plot(abscisses,ordonnées)
```

3.4) Le fichier `TP3.data2` contient un jeu de données qui a été généré dans les mêmes conditions que `data1`. Il peut donc servir de test. Quel est le degré de polynome de régression que vous sélectionneriez à l'aide de ce fichier test ?

2 Régression linéaire avec Scikit-learn

L'objectif de cette partie est d'implémenter et analyser des algorithmes de régression utilisant Scikit-learn.

2.1 sur les mêmes données simulées

1) Utilisez la fonction `LinearRegression` de `sklearn.linear_model` pour appliquer l'algorithme de régression par moindres carrés sur les données simulées décrites dans la partie 1.2 (consultez la documentation en ligne `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression`). Obtient-on la même fonction de régression (affichez les attributs `coef_` et `intercept_`).
2) Utilisez la fonction `mean_squared_error` de `sklearn.metrics` pour évaluer la qualité des fonctions trouvées sur des données test générées dans les mêmes conditions que les données d'apprentissage. Comparez avec ce que vous obtenez via la fonction `RSS`.

2.2 sur des données réelles, avec régularisation : Ridge et Lasso

La régression ridge et lasso sont des extensions de la régression linéaire par moindres carrés permettant d'éviter le risque de sur-apprentissage. L'idée est d'ajouter une pénalisation au problème de régression par moindres carrés :

$$\arg\min_{w\in\mathbb{R}^d} \sum_{i=1}^{n} \big(y_i - \langle w, x_i\rangle\big)^2 + \lambda\Omega(w),$$

où $\lambda \in \mathbb{R}$ est un paramètre de régularisation et $\Omega(w) = \|w\|_2^2$ pour la régression ridge et $\Omega(w) = \|w\|_1$ pour le Lasso.

1) Appliquez la régression linéaire ordinaire, la régression Ridge et la régression Lasso sur le jeu de données `diabetes` avec $\alpha = 1.0$. Les deux fonctions `Ridge` et `Lasso` se trouvent dans `sklearn.linear_model`. Á partir des données `diabetes`, vous crééerez des données d'apprentissage et des données test. Vous entrainerez les trois méthodes de régression sur les données d'apprentissage et vous afficherez l'erreur quadratique moyenne - vue à la section précédente - des trois fonctions apprises, calculée sur l'échantillon test. Commentez les résultats que vous obtenez.

2) Le choix du paramètre α est primordiale pour avoir des résultats de prédiction optimaux. Une façon de procéder pour trouver une bonne valeur α est d'utiliser la méthode de cross-validation sur une grille de valeurs (voir la fonction `GridSearchCV`). Essayez les lignes de codes ci-dessous pour déterminer les valeurs de α permettant d'avoir les meilleurs taux de prédiction.

```
from sklearn.model_selection import GridSearchCV
alphas = np.logspace(start=-4, stop=-1, num = 100, base = 10)
gscv = GridSearchCV(Ridge(), dict(alpha=alphas), cv=10).fit(X_app,Y_app)
alpha_r = gscv.best_params_['alpha']
print(alpha_r)
gscv = GridSearchCV(Lasso(), dict(alpha=alphas), cv=10).fit(X_app,Y_app)
alpha_l = gscv.best_params_['alpha']
print(alpha_l)
```

3) Quel est le score des fonctions apprises avec ces nouveaux paramètres ? Commentaires ?