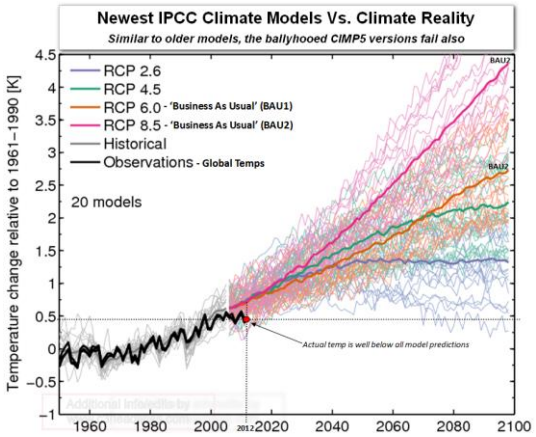
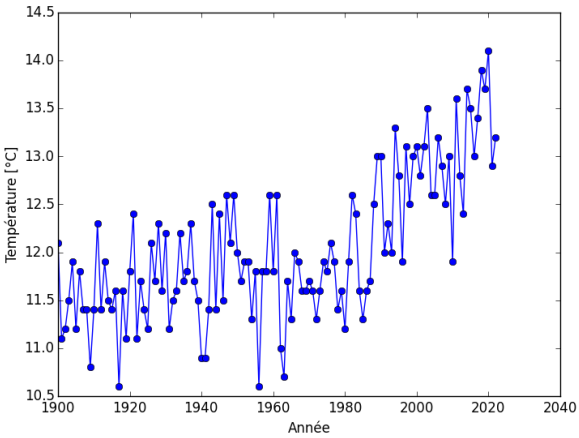
Modèle de prédiction de la température annuelle moyenne en France



Préparation des données

On s'intéresse à l'évolution de la moyenne de la température annuelle en France depuis l'année 1900, première année pour laquelle des relevés réguliers ont été réalisés.

Le fichier *Temp_annuelle_France.xlsx* comprend deux colonnes dans lesquelles on retrouve l'évolution de la température moyenne annuelle en fonction de l'année. Ces deux jeux de données seront appelés par la suite *Annee* et *Temp.* Les données sont tracées sur le graphique suivant :



Q1 : Après avoir calculé la moyenne et l'écart type des données *Annee* et *Temp*. Centrer ces données en stockant pour chacune d'entre elles la moyenne et écart type. On rappelle que le centrage des données est donné par la formule suivante : $X = \frac{X - moyenne}{ecart\ type}$.

Une régression linéaire multiple est envisagée, avec un modèle polynomial. Plusieurs degrés seront comparés (de 0 à 6) afin de valider l'évolution de la température, selon plusieurs modèles.

Régression avec Régression Linéaire Multiple

Afin de pouvoir effectuer l'apprentissage sur les données étudiées ici, nous allons devoir programmer trois fonctions intermédiaires :

- Fonction modèle *h* prenant en arguments une matrice *X* et le vecteur de paramètres *W* et qui retourne la valeur : *h(Annee.W)*, où *h* est la fonction polynomiale.
- Fonction coût *J* prenant en arguments la matrice des entrées, le vecteur des sorties et le vecteur des paramètres et calculant le coût (ou l'erreur) de la fonction modèle.
- Fonction *Gradient*, prenant les mêmes arguments que la fonction précédente et qui sera utilisée dans l'algorithme de descente de Gradient. Le calcul du gradient est donné par la formule : $Gradient(X,Y) = \frac{1}{N}.X^T.(\tilde{Y} - Y)$ avec N = Nombre d'éléments de X et $\tilde{Y}$ la fonction approchée de Y à l'aide du modèle polynomial.

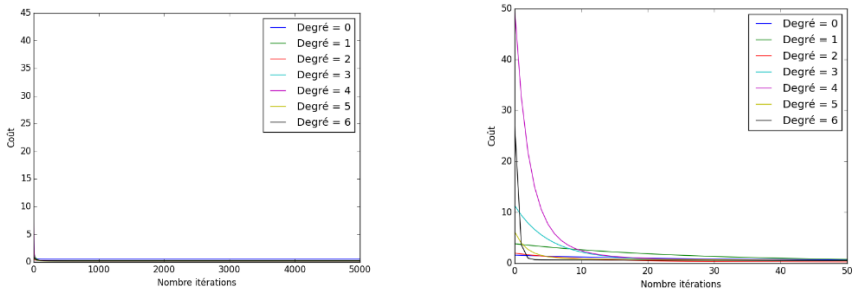
Q2 : Coder ces trois fonctions, ainsi qu'une fonction globale *Regression_Lineaire* prenant en arguments la matrice *Annee* des observations, le vecteur *Temp* des étiquettes, le degré *d* du modèle du modèle polynomial, le nombre d'itérations maximales, le taux d'apprentissage, et qui retourne le vecteur *W* des paramètres optimisant le coût, ainsi que l'historique de ce coût à chaque passage de boucle sous forme de liste.

Remarque : La méthode de la descente du gradient utilise la méthode suivante dans la boucle d'itération : $W = W - \alpha.Gradient(X,Y,W)$ avec α le taux d'apprentissage.

Remarque : On prendra garde à ne pas oublier dans la fonction *Regression_Lineaire* l'ajout d'une colonne de 1 dans la matrice *Annee* pour prendre en compte le biais dans le vecteur *W*, ainsi qu'un nombre de colonnes correspondant au degré du modèle polynomial choisi (par exemple, on aura une colonne supplémentaire contenant les carrés des entrées pour un degré 2, etc.). Cette fonction est donnée par le code suivant :

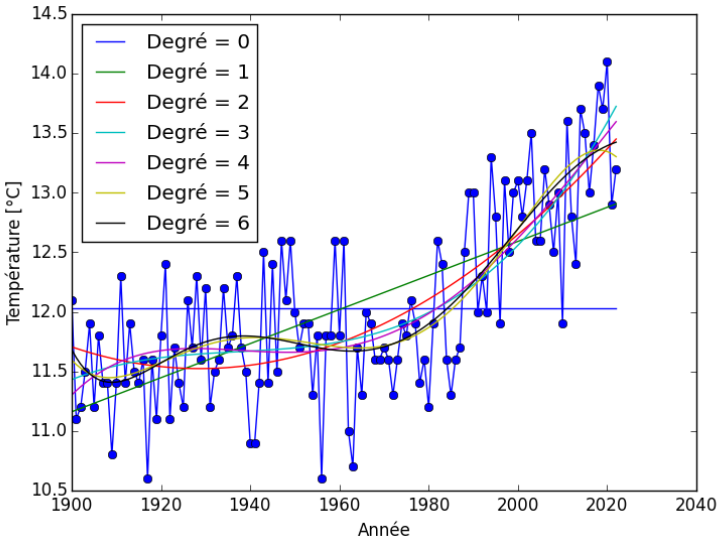
```
def Degre_multiple(X,d):
    N = np.shape(X)[0]
    U = np.ones((N,1))
    if d==0:
        R=U
    else:
        R=np.hstack((U,X)) # np.hstack Permet d'empiler des séquences horizontalement
        k=2
        while k<=d:
            R=np.hstack((R,X**k))
            k+=1
    return R
```

En utilisant les fonctions précédemment créées, on trace l'évolution du coût (qui est, on le rappelle pour une régression linéaire, une erreur quadratique) en fonction du nombre d'itérations. La convergence est estimée atteinte lorsque ce coût tend vers une asymptote horizontale. La courbe obtenue pour 5000 itérations est obtenue ci-après.



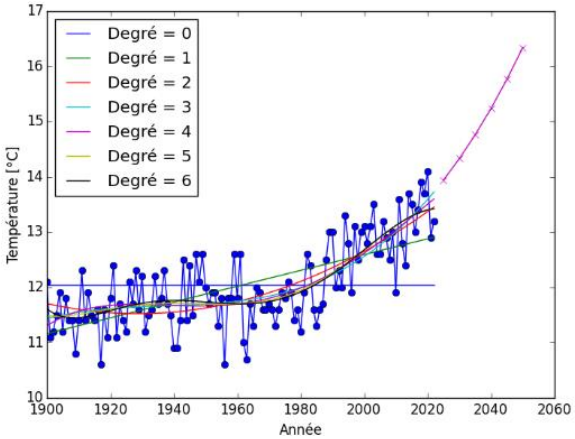
Q3 : Coder une fonction *Prediction* qui prend en entrées une matrice *X* et le vecteur de paramètres *W* et qui retourne la fonction modèle *h* avec en arguments une matrice *X* (avec la bonne taille en fonction du degré) et le vecteur de paramètres *W.*

Les différentes prédictions sont visibles sur la courbe ci-dessous.



Q4 : Déterminer le degré du modèle polynomial qui vous semble le mieux adapté.

Q5 : Calculer, avec le modèle choisi, les prédictions pour les années 2025, 2030, 2035, 2040, 2045 et 2050.



Utilisation de la bibliothèque *scikit-learn*

Il est possible d'utiliser des fonctions toutes faites, au lieu de re-coder l'ensemble des fonctions. En effet, les modèles linéaires sont implémentés dans le module *linear_model*. La régression linéaire elle-même est implémentée dans la classe *linear_model.LinearRegression*.

Pour la régression linéaire polynomiale, il est nécessaire de préparer les données en générant des données d'entrées compatible avec une régression polynomiale grâce à *PolynomialFeatures*, utilisé ainsi :

```
from sklearn.preprocessing import PolynomialFeatures
transformation_polynomiale = PolynomialFeatures(degree = degre)
X_t = transformation_polynomiale.fit_transform(X)
```

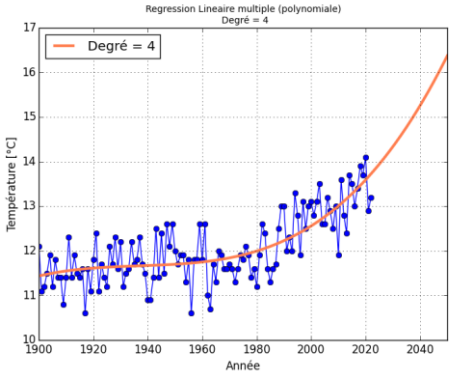
Dans les lignes de code ci-dessus, *degre* est un entier indiquant le degré du polynôme, tandis que *X_t* sera le résultat de la transformation générant une matrice incluant un nombre de colonne correspondant au degré. *X* est le vecteur des entrées.

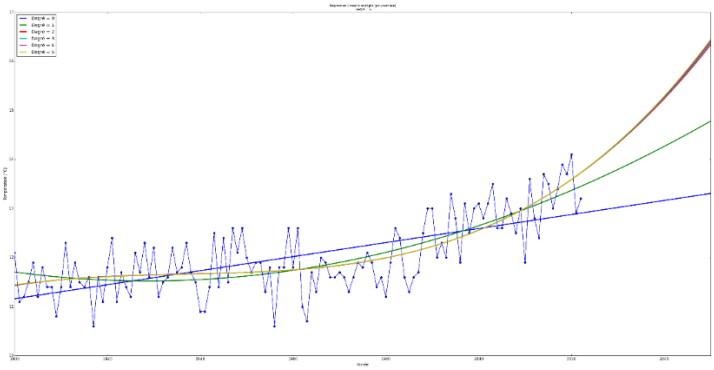
Pour la régression polynomiale, on instancie un objet de la classe de modèle qui nous intéresse, ici *LinearRegression()*. De même, on entraînera cet objet sur les données d'entraînement (*(X_t,Y)* et non *(X,Y)*) avec la méthode *.fit*.

Enfin, pour prédire des résultats, une fois le modèle entraîné, on utilisera la méthode *.predict*. On remarque alors que quel que soit le type d'apprentissage utilisé, les méthodes pour entraîner le modèle puis faire des prédictions restent les mêmes…

Enfin, il est possible d'évaluer les performances du modèle d'apprentissage grâce au module *metrics*, qui s'importe. On utilisera la fonction *mean_squared_error* qui prend en arguments *Y*, les étiquettes à prédire, et *Ym*, les sorties prédites par le modèle. On aura alors l'erreur quadratique entre les données prédites et à prédire par le modèle entraîné.

Q6 : Compléter les étapes 2, 3, 4 et 6 afin de permettre d'entraîner le modèle, de prédire l'évolution de la température en fonction des années de 1900 à 2050 et d'évaluer les performances du modèle d'apprentissage en calculant l'erreur quadratique. On pourra utiliser le meilleur degré du polynôme choisi (0 à 6) déterminé dans la partie précédente.





```python
# -*- coding: utf-8 -*-
import os
os.chdir('CHEMIN DES FICHIERS')
import numpy as np
import matplotlib.pyplot as plt

# Extraction des données et création des 2 jeux de données
dataset=np.loadtxt('Temp_annuelle_France.csv', skiprows=1,delimiter=';')
Annee=np.array([dataset[i][0] for i in range(len(dataset))])
Annee = Annee.reshape((len(Annee)),1)
Temp=np.array([dataset[i][1] for i in range(len(dataset))])
Temp = Temp.reshape((len(Temp)),1)

# TRACE Dataset
plt.figure(1)
plt.plot(Annee,Temp,'bo-')
plt.xlabel('Année')
plt.ylabel('Température [°C]')

# Calcul des moyennes et écrat types
mu_Annee =
sigma_Annee =
mu_Temp =
sigma_Temp =

# Centrage des données
Annee =
Temp =

def h(X,W):
    return

def J(X,Y,W):
    N=np.shape(X)[0]
    cout =
    return

def Gradient(X,Y,W):
    N = np.shape(X)[0]
    return

def Degre_multiple(X,d):
    N = np.shape(X)[0]
    U = np.ones((N,1))
    if d==0:
        R=U
    else:
        R=np.hstack((U,X)) # np.hstack Permet d'empiler des séquences horizontalement
        k=2
        while k<=d:
            R=np.hstack((R,X**k))
            k+=1
    return R

def Regression_Lineaire(X,Y,degre,nbre_iter=1000,alpha=0.01):
    W = np.random.randn(degre+1,1)
    X = Degre_multiple(X,degre)
    Histoire_cout = [J(X,Y,W)]
    k=0
    while k<nbre_iter :


        k+=1
    return W,Histoire_cout

L_W = []
L_cout = []

for k in range(0,7):
    W,ch = Regression_Lineaire(Annee,Temp,k,nbre_iter=5000,alpha=0.02)
    L_W.append(W)
    L_cout.append(ch)

plt.figure(2)
for k in range(len(L_cout)):
    labell = 'Degré = {}'.format(k)
    plt.plot(L_cout[k],label=str(labell))
plt.xlabel("Nombre itérations")
plt.ylabel("Coût")
plt.legend()

plt.figure(3)
for k in range(len(L_cout)):
    labell = 'Degré = {}'.format(k)
    plt.plot(L_cout[k],label=str(labell))
plt.xlabel("Nombre itérations")
plt.ylabel("Coût")
plt.xlim(0,50)
plt.legend()

def Prediction(X,W):
    degre = len(W)-1

    return

plt.figure(1)
for k in range(len(L_W)):
    ym = Prediction(Annee,L_W[k])
    X = Annee*sigma_Annee + mu_Annee
    ym = ym*sigma_Temp + mu_Temp
    labell = 'Degré = {}'.format(k)
    plt.plot(X,ym,label=str(labell))
plt.legend(loc="upper left")

# Un degré 4 semble être la solution offrant le modèle le plus proche
# prédiction températures en 2030, 2050 et 2100...

Xp = np.array([[2025,2030,2035,2040,2045,2050]]).T
Xpn =                       # On normalise pour coller au modèle
yp =                        # On prend le modèle de degré ??
yp =

plt.figure(1)
plt.plot(Xp,yp,'mx-')
```

```python
# -*- coding: utf-8 -*-
import os
os.chdir('CHEMIN DES FICHIERS')
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

# Etape 1 : Recuperation du dataset
dataset=np.loadtxt('Temp_annuelle_France.csv', skiprows=1,delimiter=';')
Annee=np.array([dataset[i][0] for i in range(len(dataset))])
X = Annee.reshape((len(Annee)),1)
Temp=np.array([dataset[i][1] for i in range(len(dataset))])
Y = Temp.reshape((len(Temp)),1)

plt.figure(1)
plt.plot(X,Y,'bo-')
plt.xlabel('Année')
plt.ylabel('Température [°C]')

# Etape 2 : Preparation des donnees
degre =



# Etape 3: Definition et entraînement du modèle
Modele_RP =


# Etape 4 : Prediction
x_new_min = 1900
x_new_max = 2050
Xp = np.linspace(x_new_min, x_new_max, 2050-1900)
Xp = Xp[:,np.newaxis]
Xp_t =
Yp =

# Etape 5 : Graphiques
plt.figure(1)
plt.plot(Xp, Yp,label=str(labell), color='coral', linewidth=3)
plt.grid()
labell = 'Degré = {}'.format(degre)
plt.xlim(x_new_min,x_new_max)
title = 'Degré = {}'.format(degre)
plt.title("Regression Lineaire multiple (polynomiale) \n " + title, fontsize=10)
plt.xlabel('Année')
plt.ylabel('Température [°C]')
plt.legend(loc="upper left")
plt.show()

# Etape 6 : Calculs des erreurs
Ym =
rmse =
print('RMSE: ', rmse)
```