RAPPORT DE PROJET REGRESSION LINEAIRE

Présenté par : Coumba SY

Sous la direction de : Dr Mamadou BOUSSO

Régression linéaire simple

1. Simple Python and numpy function

```
def warmUpExercise(n):
    """
    Example function in Python which computes the identity matrix.

    Returns
    -------
    A : array_like
        The 5x5 identity matrix.

    Instructions
    ------------
    Return the 5x5 identity matrix.
    """
    # ======== YOUR CODE HERE ======
    A = np.eye(n)

    # ==============================
    return A
```

```
Entrée [4]: warmUpExercise(5)

Out[4]: array([[1., 0., 0., 0., 0.],
               [0., 1., 0., 0., 0.],
               [0., 0., 1., 0., 0.],
               [0., 0., 0., 1., 0.],
               [0., 0., 0., 0., 1.]])
```

2. Régression Linéaire à une seule variable

Chargement de données du fichier ex1data1.txt

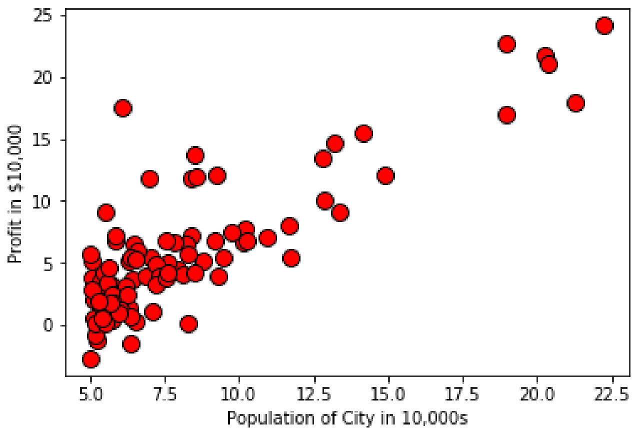
```python
# Read comma separated data
data = np.loadtxt(os.path.join('Data', 'ex1data1.txt'),delimiter=',')

X = data[:,0]
y = data[:,1]

m = X.size  # number of training examples
# print out some data points
print('{:>8s}{:>10s}'.format('Pop(X)', 'profit(y)'))
print('-'*26)
for i in range(10):
    print('{:8.0f}{:10.0f}'.format(X[i], y[i]))
```

```
  Pop(X) profit(y)
--------------------------
       6        18
       6         9
       9        14
       7        12
       6         7
       8        12
       7         4
       9        12
       6         7
       5         4
```

2.1. Plotting the data

```python
def plotData(x, y,xlabel,ylabel):
    """
    Plots the data points x and y into a new figure. Plots the data
    points and gives the figure axes labels of population and profit.

    Parameters
    ----------
    x : array_like
        Data point values for x-axis.

    y : array_like
        Data point values for y-axis. Note x and y should have the same size.

    """
    fig = plt.figure()  # open a new figure

    # ====================== YOUR CODE HERE =======================
    plt.plot(x, y, 'ro', ms=10, mec='k')
    plt.ylabel('Profit in $10,000')
    plt.xlabel('Population of City in 10,000s')
    # =============================================================
```

```
plotData(X, y,'Population of City in 10,000s','Profit in $10,000')
```



2.2. Gradient descent

- Implémentation de la fonction coût

```
def computeCost(X, y, theta):
    """
    Compute cost for linear regression. Computes the cost of using theta as the
    parameter for linear regression to fit the data points in X and y.

   """

    # initialize some useful values


    # You need to return the following variables correctly

    # ============================Your Code==============================
    m = len(X) #le nombre de valeurs
    J = 1/(2*m) * np.sum((X.dot(theta)-y)**2)
    # ==========================================================
    return J
```

```
J = computeCost(X, y, theta=np.array([0.0, 0.0]))
print('With theta = [0, 0] \nCost computed = %.2f' % J)
print('Expected cost value (approximately) 32.07\n')

# further testing of the cost function
J = computeCost(X, y, theta=np.array([-1, 2]))
print('With theta = [-1, 2]\nCost computed = %.2f' % J)
print('Expected cost value (approximately) 54.24')
```

```
With theta = [0, 0]
Cost computed = 32.07
Expected cost value (approximately) 32.07

With theta = [-1, 2]
Cost computed = 54.24
Expected cost value (approximately) 54.24
```

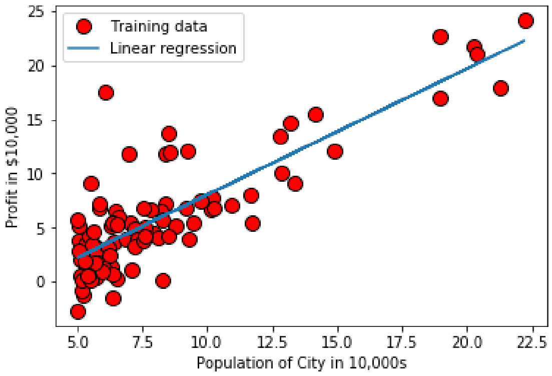
- Implémentation de la descente de gradient

```python
def gradientDescent(X, y, theta, alpha, num_iters):
    """
    Performs gradient descent to learn `theta`. Updates theta by taking `num_iters`
    gradient steps with learning rate `alpha`.


    """
    # Initialize some useful values


    # make a copy of theta, to avoid changing the original array, since numpy arrays
    # are passed by reference to functions
    theta = theta.copy()

    J_history = [] # Use a python list to save cost in every iteration

    m = len(y)

    # ==================== YOUR CODE HERE =================================

    for i in range(num_iters):
        J = computeCost(X, y, theta)
        A = X.dot(theta)-y
        theta = theta - alpha *(1/m * X.T.dot(A))
        J_history.append(J)

    # =====================================================================

    # save the cost J in every iteration


    return theta, J_history
```

```python
# initialize fitting parameters
theta = np.zeros(2)

# some gradient descent settings
iterations = 1500
alpha = 0.01

theta, J_history = gradientDescent(X ,y, theta, alpha, iterations)
print('Theta found by gradient descent: {:.4f}, {:.4f}'.format(*theta))
print('Expected theta values (approximately): [-3.6303, 1.1664]')
```

```
Theta found by gradient descent: -3.6303, 1.1664
Expected theta values (approximately): [-3.6303, 1.1664]
```

```python
# plot the linear fit
plotData(X[:, 1], y,'Population of City in 10,000s',"Profit in $10000")
plt.plot(X[:, 1], np.dot(X, theta), '-')
plt.legend(['Training data', 'Linear regression']);
```




```python
# Predict values for population sizes of 35,000 and 70,000
predict1 = np.dot([1, 3.5], theta)
print('For population = 35,000, we predict a profit of {:.2f}\n'.format(predict1*10000))

predict2 = np.dot([1, 7], theta)
print('For population = 70,000, we predict a profit of {:.2f}\n'.format(predict2*10000))
```

```
For population = 35,000, we predict a profit of 4519.77

For population = 70,000, we predict a profit of 45342.45
```

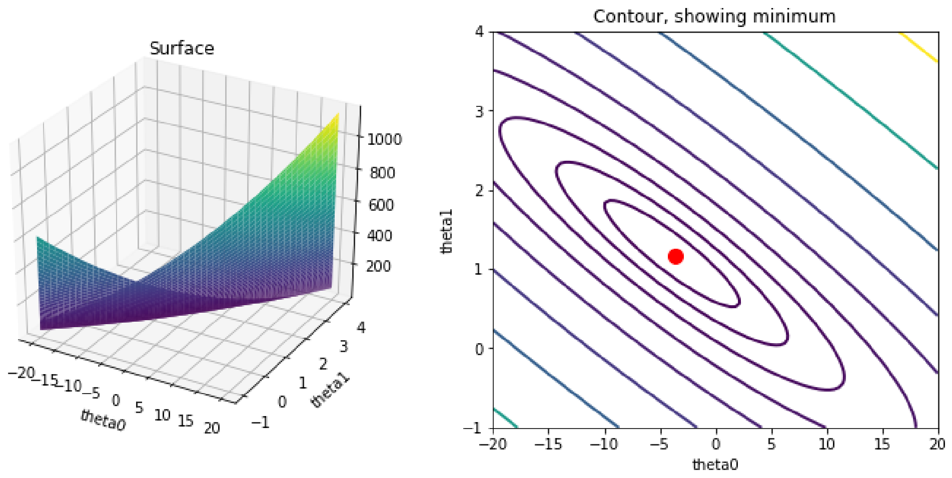

```python
# Predict values for population sizes of 35,000 and 70,000
predict1 = np.dot([1, 3.5], theta)
print('For population = 35,000, we predict a profit of {:.2f}\n'.format(predict1*10000))

predict2 = np.dot([1, 7], theta)
print('For population = 70,000, we predict a profit of {:.2f}\n'.format(predict2*10000))
```

```
For population = 35,000, we predict a profit of 4519.77

For population = 70,000, we predict a profit of 45342.45
```


2.3. Visualisation de la fonction coût

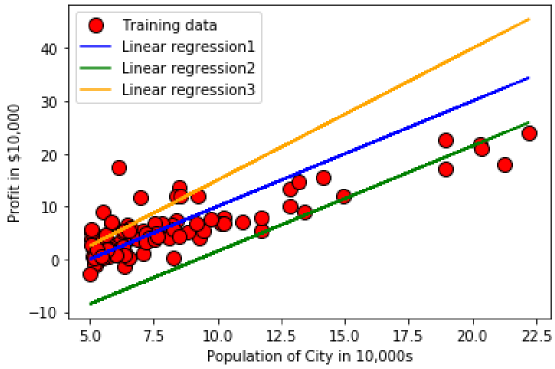
```python
# grille sur laquelle nous calculerons J
theta0_vals = np.linspace(-20, 20, 100)
theta1_vals = np.linspace(-1, 4, 100)

# initialiser J_vals à une matrice de 0
J_vals = np.zeros((theta0_vals.shape[0], theta1_vals.shape[0]))

# Remplissez J_vals
for i, theta0 in enumerate(theta0_vals):
    for j, theta1 in enumerate(theta1_vals):
        J_vals[i, j] = computeCost(X, y, [theta0, theta1])

# En raison du fonctionnement des maillages dans la commande Surf, nous devons
# transposer J_vals avant d'appeler surf, sinon les axes seront inversés
J_vals = J_vals.T

# surface plot
fig = plt.figure(figsize=(12, 5))
ax = fig.add_subplot(121, projection='3d')
ax.plot_surface(theta0_vals, theta1_vals, J_vals, cmap='viridis')
plt.xlabel('theta0')
plt.ylabel('theta1')
plt.title('Surface')

# contour plot
# Tracez J_vals en 15 contours espacés logarithmiquement entre 0,01 et 100
ax = plt.subplot(122)
plt.contour(theta0_vals, theta1_vals, J_vals, linewidths=2, cmap='viridis', levels=np.logspace(-2, 3, 20))
plt.xlabel('theta0')
plt.ylabel('theta1')
plt.plot(theta[0], theta[1], 'ro', ms=10, lw=2)
plt.title('Contour, showing minimum')
```

```
Text(0.5, 1.0, 'Contour, showing minimum')
```



EXERCICE : CHOISIR 3 POINTS DANS LE CONTOUR ET REPRESENTER LES DROITES QUI LEUR CORRESPONDENT DANS LE NUAGE DE POINTS

```
# plot the linear fit
plotData(X[:, 1], y,'Population of City in 10,000s',"Profit in $10000")
#---------Your Code ---------------------------------------------------------------------------
plt.plot(X[:, 1], np.dot(X, [-10,2]), '-', color="blue")
plt.plot(X[:, 1], np.dot(X, [-18.5,2]), '-', color="green")
plt.plot(X[:, 1], np.dot(X, [-10,2.5]), '-', color="orange")
plt.legend(['Training data', 'Linear regression1', 'Linear regression2', 'Linear regression3' ]);
```



Exercice optionnel

```python
def computeCost2(X, y, theta):
    m = len(X)

    J = (1/m)*np.sum(np.abs(np.dot(X,theta)-y))
    return J
```

```python
J = computeCost2(X, y, theta=np.array([0.0, 0.0]))
print('With theta = [0, 0] \nCost computed = %.2f' % J)


# further testing of the cost function
J = computeCost2(X, y, theta=np.array([-1, 2]))
print('With theta = [-1, 2]\nCost computed = %.2f' % J)
```

```
With theta = [0, 0]
Cost computed = 5.96
With theta = [-1, 2]
Cost computed = 9.61
```

```python
def gradientDescent2(X, y, theta, alpha, num_iters):

    theta = theta.copy()

    J_history = [] # Use a python list to save cost in every iteration

    m = len(y)

    # =================== YOUR CODE HERE ================================

    for i in range(num_iters):
        J = computeCost2(X, y, theta)
        A = X.dot(theta)-y
        theta = theta - alpha *(1/m * X.T.dot(A))
        J_history.append(J)

    # =================================================================

    # save the cost J in every iteration


    return theta, J_history
```

```python
# initialize fitting parameters
theta = np.zeros(2)

# some gradient descent settings
iterations = 1000
alpha = 0.001

theta, J_history = gradientDescent2(X ,y, theta, alpha, iterations)
print('Theta found by gradient descent: {:.4f}, {:.4f}'.format(*theta))
```

```
Theta found by gradient descent: -0.5761, 0.8595
```

3. Régression Linéaire avec plusieurs variables

3.1. Feature Normalization

```python
# Load data
data = np.loadtxt(os.path.join('Data', 'ex1data2.txt'), delimiter=',')
X = data[:, :2]
y = data[:, 2]
m = y.size

# print out some data points
print('{:>8s}{:>8s}{:>10s}'.format('X[:,0]', 'X[:, 1]', 'y'))
print('-'*26)
for i in range(10):
    print('{:8.0f}{:8.0f}{:10.0f}'.format(X[i, 0], X[i, 1], y[i]))
```

```
  X[:,0] X[:, 1]         y
--------------------------
    2104       3    399900
    1600       3    329900
    2400       3    369000
    1416       2    232000
    3000       4    539900
    1985       4    299900
    1534       3    314900
    1427       3    198999
    1380       3    212000
    1494       3    242500
```

```python
def  featureNormalize(X):
    """
    Normalise les fonctionnalités de X. renvoie une version normalisée de X où
    la valeur moyenne de chaque caractéristique est 0 et l'écart-type
    est 1. Ceci est souvent une bonne étape de prétraitement à faire lorsque vous travaillez avec
    algorithmes d'apprentissage.


    """
    # You need to set these values correctly
    X_norm = X.copy()
    mu = np.zeros(X.shape[1])
    sigma = np.zeros(X.shape[1])
    n = X_norm.shape[1]
    # =========================== YOUR CODE HERE =====================
    i=0
    for col in range(X.shape[1]):
        mu[i] = np.mean(X[:,col])
        sigma[i] = np.std(X[:, col])
        i = i+1

    for row in range(X.shape[0]):
        for col in range(X.shape[1]):
            X_norm[row,col] = (X[row, col] - mu[col])/sigma[col]



    # ================================================================
    return X_norm, mu, sigma
```

```python
# call featureNormalize on the loaded data
X_norm, mu, sigma = featureNormalize(X)

print('Computed mean:', mu)
print('Computed standard deviation:', sigma)
# Computed mean: [2000.68085106    3.17021277]
# Computed standard deviation: [7.86202619e+02 7.52842809e-01
```

```
Computed mean: [2000.68085106    3.17021277]
Computed standard deviation: [7.86202619e+02 7.52842809e-01]
```

3.2. Implémentation de la fonction de coût

```python
def computeCostMulti(X, y, theta):
    """
    Compute cost for linear regression with multiple variables.
    Computes the cost of using theta as the parameter for linear regression to fit the data points in X and y.

    """
    # Initialize some useful values
    m = y.shape[0] # number of training examples

    # You need to return the following variable correctly
    J = 0

    # ====================== YOUR CODE HERE ==========================
    A = np.dot(X,theta)-y
    J = 1/(2*m) * np.dot(A.T, A)
    # ================================================================
    return J
```

3.3. Implémentation de la descente de gradient

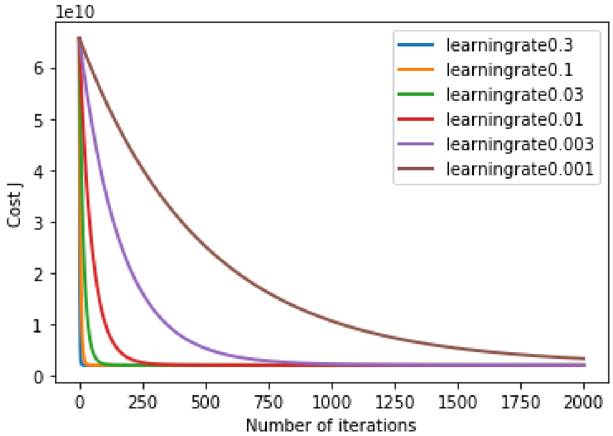
```python
def gradientDescentMulti(X, y, theta, alpha, num_iters):
    """
    Performs gradient descent to learn theta.
    Updates theta by taking num_iters gradient steps with learning rate alpha.

    """
    # Initialize some useful values
    m = y.shape[0] # number of training examples

    # make a copy of theta, which will be updated by gradient descent
    theta = theta.copy()

    J_history = []

    for i in range(num_iters):
        J = computeCostMulti(X, y, theta)
        A = X.dot(theta)-y
        theta = theta - (alpha/m * np.dot(A,X))
        J_history.append(J)


    return theta, J_history
```

```python
# Choose some alpha value
alpha = 0.02
num_iters = 400

# init theta and run gradient descent
theta = np.zeros(3)
theta, J_history = gradientDescentMulti(X, y, theta, alpha, num_iters)

# Plot the convergence graph
plt.plot(np.arange(len(J_history)), J_history, lw=2)
plt.xlabel('Number of iterations')
plt.ylabel('Cost J')

# Display the gradient descent's result
print('theta computed from gradient descent: {:s}'.format(str(theta)))

# Estimate the price of a 1650 sq-ft, 3 br house
# ======================= YOUR CODE HERE ===========================
# Recall that the first column of X is all-ones.
# Thus, it does not need to be normalized.

price = np.dot(np.array([1.,(1650. - mu[0])/sigma[0] , (3. - mu[1])/sigma[1]]),np.array(theta) )  # You should change this
print(price)
# ===================================================================

print('Predicted price of a 1650 sq-ft, 3 br house (using gradient descent): ${:.0f}'.format(price))
```

```
theta computed from gradient descent: [340307.35772969 107757.47433209  -4888.35338493]
293348.0221781551
Predicted price of a 1650 sq-ft, 3 br house (using gradient descent): $293348
```

```python
# Choose some alpha value
alpha = [0.3, 0.1, 0.03, 0.01, 0.003, 0.001]
num_iters = 2000

# init theta and run gradient descent
for i in range(len(alpha)):
    theta = np.zeros(3)
    theta, J_history = gradientDescentMulti(X, y, theta, alpha[i], num_iters)

    # Plot the convergence graph
    plt.plot(np.arange(len(J_history)), J_history, lw=2,label = 'learningrate'+str(alpha[i]))
    plt.xlabel('Number of iterations')
    plt.ylabel('Cost J')
    plt.legend()
    print('theta computed with from gradient descent: {:s}'.format(str(theta)))
# Display the gradient descent's result


# Estimate the price of a 1650 sq-ft, 3 br house
# ======================= YOUR CODE HERE ===========================
# Recall that the first column of X is all-ones.
# Thus, it does not need to be normalized.

price =  np.dot(np.array([1.,(1650. - mu[0])/sigma[0] , (3. - mu[1])/sigma[1]]),np.array(theta) )   # You should change this

# ===================================================================

print('Predicted price of a 1650 sq-ft, 3 br house (using gradient descent): ${:.0f}'.format(price))
```

```
theta computed with from gradient descent: [340412.65957447 109447.79646964  -6578.35485416]
theta computed with from gradient descent: [340412.65957447 109447.79646964  -6578.35485416]
theta computed with from gradient descent: [340412.65957447 109447.79646948  -6578.35485399]
theta computed with from gradient descent: [340412.65894002 109439.22578243  -6569.78416695]
theta computed with from gradient descent: [339576.43615572 105311.60418477  -2450.82887525]
theta computed with from gradient descent: [294388.89339564  83125.36792731  15212.40521995]
Predicted price of a 1650 sq-ft, 3 br house (using gradient descent): $253872
```



3.4. ECRIVEZ UNE METHODE QUI PERMET DE PREDIRE LE PRIX D'UNE MAISON EN FONCTION DE LA SUPERFICIE ET DU NOMBRE DE PIECES

```python
def predict(superficie, pieces, mu, sigma,theta):
    price =  np.dot(np.array([1.,(superficie - mu[0])/sigma[0] , (pieces - mu[1])/sigma[1]]),np.array(theta) )
    return price
```

```python
price = predict(1650,3,mu,sigma,theta)
print(price)
```

```
253871.90946905076
```

3.5. Equations normales

```python
# Load data
data = np.loadtxt(os.path.join('Data', 'ex1data2.txt'), delimiter=',')
X = data[:, :2]
y = data[:, 2]
m = y.size
X = np.concatenate([np.ones((m, 1)), X], axis=1)
```

```python
def normalEqn(X, y):
    """
    Computes the closed-form solution to linear regression using the normal equations.

    """
    theta = np.zeros(X.shape[1])

    # ===================== YOUR CODE HERE ============================
    A = 1/(np.dot(X.T,X))
    B = np.dot(X.T, y)
    theta = np.dot(A,B)

    # =================================================================
    return theta
```

ECRIVEZ UNE METHODE QUI PERMET DE PREDIRE LE PRIX D'UNE MAISON EN FONCTION DE LA SUPERFICIE ET DU NOMBRE DE PIECES UTILISANT L'EQUATION NORMALE.

```python
def predictnorm(superficie, pieces,theta):
    predict(superficie, pieces, mu, sigma,theta)
    return price
```

```python
# Calculate the parameters from the normal equation
theta = normalEqn(X, y);

# Display normal equation's result
print('Theta computed from the normal equations: {:s}'.format(str(theta)));

# Estimate the price of a 1650 sq-ft, 3 br house
# ====================== YOUR CODE HERE ======================

price = predictnorm(1650,3,theta) # You should change this

# ============================================================

print('Predicted price of a 1650 sq-ft, 3 br house (using normal equations): ${:.0f}'.format(price))
```

```
Theta computed from the normal equations: [1.07579177e+06 5.03401587e+02 3.27409153e+05]
Predicted price of a 1650 sq-ft, 3 br house (using normal equations): $253872
```

4. TROUVEZ DES DONNEES ET FAITES DE LA PREVISION EN UTILISANT LES ALGORITHMES IMPLEMENTES DANS CE NOTEBOOK.

REGRESSION LINEAIRE SIMPLE: PREVISION DU SALAIRE EMPLOYE

DONNEES: SALAIRE_EMPLOYES.TXT

```python
# Load data
data = np.loadtxt(os.path.join('Data', 'salaire_employes.txt'),delimiter=',')

X = data[:,0]
y = data[:,1]

m = X.size  # number of training examples
# print out some data points
print('{:>8s}{:>10s}'.format('Expérience(X) ', ' Salaire(y)'))
print('-'*26)
for i in range(m):
    print('{:8.1f}{:10.0f}'.format(X[i], y[i]))
```

Expérience(X)	Salaire(y)
1.1	39343
1.3	46205
1.5	37731
2.0	43525
2.2	39891
2.9	56642
3.0	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4.0	55794
4.0	56957
4.1	57081
4.5	61111
4.9	67938
5.1	66029
5.3	83088
5.9	81363
6.0	93940
6.8	91738
7.1	98273
7.9	101302
8.2	113812
8.7	109431
9.0	105582
9.5	116969
9.6	112635
10.3	122391
10.5	121872

```
X = np.stack([np.ones(m), X], axis=1)
```

```
J = computeCost(X, y, theta=np.array([0.0, 0.0]))
print('With theta = [0, 0] \nCost computed = %.2f' % J)
J = computeCost(X, y, theta=np.array([-1, 2]))
print('With theta = [-1, 2]\nCost computed = %.2f' % J)
```

```
With theta = [0, 0]
Cost computed = 3251477635.37
With theta = [-1, 2]
Cost computed = 3250598902.87
```

```
# initialize fitting parameters
theta = np.zeros(2)

# some gradient descent settings
iterations = 1500
alpha = 0.01

theta, J_history = gradientDescent(X ,y, theta, alpha, iterations)
print('Theta found by gradient descent: {:.4f}, {:.4f}'.format(*theta))
print('Expected theta values (approximately): [-3.6303, 1.1664]')
```


```
# initialize fitting parameters
theta = np.zeros(2)

# some gradient descent settings
iterations = 1500
alpha = 0.01

theta, J_history = gradientDescent(X ,y, theta, alpha, iterations)
print('Theta found by gradient descent: {:.4f}, {:.4f}'.format(*theta))
print('Expected theta values (approximately): [-3.6303, 1.1664]')
```


```
Theta found by gradient descent: 24796.0216, 9597.7911
Expected theta values (approximately): [-3.6303, 1.1664]
```

```
# Predict values for population sizes of 35,000 and 70,000
predict1 = np.dot([1, 3.2], theta)
print('Pour une Experience = 3.2 on prédit un salaire de ${:.2f}\n'.format(predict1))
```

```
Pour une Experience = 3.2 on prédit un salaire de $55508.95
```