Multiple Variable Linear Regression

1.0 Problem Statement

We will use the motivating example of housing price prediction. The training dataset contains four features (size, bedrooms, floors and, age) shown in the table below.

Size (sqft)	Number of Bedrooms	Number of floors	Age of Home	Price (1000s dollars)
2104	5	1	45	460
1416	3	2	40	232
852	2	1	35	178

Let's build a linear regression model using these values so you can then predict the price for other houses. For example, a house with 1200 sqft, 3 bedrooms, 1 floor, 40 years old.

1.1 Matrix X containing our example

Each row of datasets represent one example, with n number of features and m training examples, now $\mathbf{x}$ is an input matrix with dimensions (m, n) where m is row and n is column.

$$\mathbf{X} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & \cdots & x_{n-1}^{(0)} \\ x_0^{(1)} & x_1^{(1)} & \cdots & x_{n-1}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ x_0^{(m-1)} & x_1^{(m-1)} & \cdots & x_{n-1}^{(m-1)} \end{pmatrix}$$

notation:

- $\mathbf{x}^{(i)}$ is vector containing example i. $\mathbf{x}^{(i)} = (x_0^{(i)}, x_1^{(i)}, \cdots, x_{n-1}^{(i)})$
- $x_j^{(i)}$ is element j in example i. The superscript in parenthesis indicates the example number while the subscript represents an element.

1.2 Parameter vector $\mathbf{w}_j$, b

- $\mathbf{w}$ is a vector with n elements.
  - Each element contains the parameter associated with one feature.
  - in our dataset, n is 4.
  - notionally, we draw this as a column vector

$$\mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{n-1} \end{pmatrix}$$

- b is a scalar parameter.

2.0 Load datasets

As shown in a problem statement, our dataset contains five features (size, bedrooms, floors and, age, price of the house), $n = 5$ and ahundred examples, $m = 100$.

In this case our dataset will have the size of $(m, n) = (100, 5)$

$$\mathbf{data} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & \cdots & y_{n-1}^{(0)} \\ x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \cdots & y_{n-1}^{(1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0^{(m-1)} & x_1^{(m-1)} & x_2^{(m-1)} & \cdots & y_{n-1}^{(m-1)} \end{pmatrix} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & x_3^{(0)} & y_4^{(0)} \\ x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & y_4^{(1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0^{(99)} & x_1^{(99)} & x_2^{(99)} & x_3^{(99)} & y_4^{(99)} \end{pmatrix}$$

$$data \in \mathbb{R}_{m \times n} = \mathbb{R}_{100 \times 5}$$

2.1 Load input datasets, $\mathbf{x}_j^{(i)}$

Input datasets will comprise of five features (size, bedrooms, floors and, age), $n = 4$ and hundred exaples, $m = 100$.

$$\mathbf{X} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & \cdots & x_{n-1}^{(0)} \\ x_0^{(1)} & x_1^{(1)} & \cdots & x_{n-1}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ x_0^{(m-1)} & x_1^{(m-1)} & \cdots & x_{n-1}^{(m-1)} \end{pmatrix} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & x_3^{(0)} \\ x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ x_0^{(99)} & x_1^{(99)} & x_2^{(99)} & x_3^{(99)} \end{pmatrix}$$

$$\mathbf{X} \subset \mathbb{R}^{\quad} = \mathbb{R}^{\quad}$$

2.2 Load output datasets, $\mathbf{y}_j^{(i)}$

Input datasets will comprise of only one features (price), $n = 1$ and hundred exaples, $m = 100$.

$$\mathbf{Y} = \begin{pmatrix} y_0^{(0)} \\ y_0^{(1)} \\ \vdots \\ y_0^{(m-1)} \end{pmatrix} = \begin{pmatrix} y_0^{(0)} \\ y_0^{(1)} \\ \vdots \\ y_0^{(99)} \end{pmatrix}$$

$$Y \in \mathbb{R}_{m \times n} = \mathbb{R}_{100 \times 1}$$

2.3 Initialize parameters $\mathbf{w}_j$, b

- $\mathbf{w}_j$ is a vector with $n = 4$ elements.
    - Each element contains the parameter associated with one feature.

$$\mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{n-1} \end{pmatrix} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

$$\mathbf{w} \in \mathbb{R}_{n\times 1} = \mathbb{R}_{4\times 1}$$

- b is a scalar parameter.

```c
float **load_data(const char *file_id){

    /*
    load dataset in '.txt' file

    Args:
      file_id (char): FILE_NAME

    Returns:
      data (ndarray):  shape(m, num_col_x)

    */

    FILE *file = fopen(file_id, "r");
    if (file==NULL)
    {
        perror("Error opening file");
        return NULL;
    }

    //alocate the memory for the data array dynamically
    float **data=(float **)malloc(m*sizeof(float *));
    if (data==NULL)
    {
        perror("Error allocating memory");
        fclose(file);
        return NULL;
    }

    for (int i = 0; i < m; i++)
    {
       data[i] = (float *)malloc(num_col_x*sizeof(float));

       if (data[i]==NULL)
       {
            perror("Error allocating memory");
            for (int j = 0; j < i; j++)
            {
                free(data[j]);
            }
            free(data);
            fclose(file);
            return NULL;
       }

    }

    //read the data into the array
    for (int i = 0; i < m; i++)
    {
        if (fscanf(file, "%e, %e, %e, %e, %e", &data[i][0], &data[i][1], &data[i][2], &data[i][3], &data[i][4]) != 5)
        {
            fprintf(stderr, "Error reading data at row: %d\n", i);

        //free allocated memory before exiting due to error
        for (int j = 0; j <= i; j++)
        {
            free(data[j]);
        }
        free(data);
        fclose(file);
        return NULL;
        }

    }

    fclose(file);



    return data;
}
```

3.0 Model Prediction With Multiple Variables

The model's prediction with multiple variables is given by the linear model:

$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = w_0x_0 + w_1x_1 + ... + w_{n-1}x_{n-1} + b \tag{1}$$

or in vector notation:

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{X} + \mathbf{b} \tag{2}$$

$$f_{\mathbf{w},b}(\mathbf{x}) \in \mathbb{R}_{m\times 1} = \mathbf{w} \in \mathbb{R}_{n\times 1} \cdot \mathbf{X} \in \mathbb{R}_{m\times n} + \mathbf{b} \in \mathbb{R}_1$$

$$f_{\mathbf{w},b}(\mathbf{x})_{m\times 1} = \mathbf{w}_{n\times 1} \cdot \mathbf{X}_{m\times n} + \mathbf{b}_1$$

$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) \in \mathbb{R}_{m\times 1}$$

$$f_{\mathbf{w},b}(\mathbf{x}) = \begin{pmatrix} f_{\mathbf{w},b}(\mathbf{x})_0^{(0)} \\ f_{\mathbf{w},b}(\mathbf{x})_0^{(1)} \\ \vdots \\ f_{\mathbf{w},b}(\mathbf{x})_0^{(m-1)} \end{pmatrix} = \begin{pmatrix} f_{\mathbf{w},b}(\mathbf{x})_0^{(0)} \\ f_{\mathbf{w},b}(\mathbf{x})_0^{(1)} \\ \vdots \\ f_{\mathbf{w},b}(\mathbf{x})_0^{(99)} \end{pmatrix}$$

where · is a vector `dot product`

To demonstrate the dot product, we will implement prediction using (1) and (2).

```c
double *predict(float **x, double w[][1], double b){

    /*
    single predict using linear regression

    Args:
      x (ndarray): Shape (m, n) example with multiple features
      w (ndarray): Shape (n, 1) model parameters
      b (scalar):  model parameter

    Returns:
      p (scalar):  prediction

    */

    double p;
    double *f_wb=(double *)malloc(m*sizeof(double));
    if (f_wb==NULL)
    {
        perror("Error allocating memory");
        free(f_wb);
    }


    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < 1; j++)
        {
            p=0;
            for (int k = 0; k < n; k++)
            {
                p += (double)x[i][k]*w[k][j];
            }
            p += b;
            f_wb[i]=p;

        }

    }

    return f_wb;
}
```

4.0 Compute Cost With Multiple Variables

The equation for the cost function with multiple variables $J(\mathbf{w}, b)$ is:

$$J(\mathbf{w}, b) = \frac{1}{2m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})^2 \tag{3}$$

where:

$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = \mathbf{w} \cdot \mathbf{X} + b \tag{4}$$

$$f_{\mathbf{w},b}(\mathbf{x}) \in \mathbb{R}_{m \times 1}$$

In contrast to previous labs, $\mathbf{w}$ and $\mathbf{x}^{(i)}$ are vectors rather than scalars supporting multiple features.

```
double compute_cost(float **x, float y[m], double w[][1], double b){

    /*
     compute cost
    Args:
      X (ndarray (m,n)): Data, m examples with n features
      y (ndarray (m,1)) : target values
      w (ndarray (n,1)) : model parameters
      b (scalar)       : model parameter

    Returns:
      cost (scalar): cost

    */

    double cost=0;
    double p;

    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < 1; j++)
        {
            p=0;
            for (int k = 0; k < n; k++)
            {
                p += (double)x[i][k]*w[k][j];
            }
            p +=b;
        }
        cost += pow((p - (double) y[i]), 2);
    }
    cost /= (2*m);

    return cost;
```

5.0 Gradient Descent With Multiple Variables

Gradient descent for multiple variables:

$$
\begin{aligned}
&\textbf{repeat until convergence: } \{ \\
&\quad w_j = w_j - \alpha \frac{\partial J(\mathbf{w},b)}{\partial w_j} \quad \text{for j} = 0..\text{n-1} \qquad (5) \\
&\quad b \ = b - \alpha \frac{\partial J(\mathbf{w},b)}{\partial b} \\
&\}
\end{aligned}
$$

where, n is the number of features, parameters w_j, b, are updated simultaneously and where

$$\frac{\partial J(\mathbf{w},b)}{\partial w_j} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)} \qquad (6)$$

$$\frac{\partial J(\mathbf{w},b)}{\partial b} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}) \qquad (7)$$

- m is the number of training examples in the data set
- $f_{\mathbf{w},b}(\mathbf{x}^{(i)})$ is the model's prediction, while $y^{(i)}$ is the target value

Note:

- $$\frac{\partial J(\mathbf{w},b)}{\partial w_j} \in \mathbb{R}_{1\times n} = \mathbb{R}_{1\times 4}$$

$$\frac{\partial J(\mathbf{w},b)}{\partial w_j} = \left(\frac{\partial J(\mathbf{w},b)}{\partial w_0}, \frac{\partial J(\mathbf{w},b)}{\partial w_1}, \cdots, \frac{\partial J(\mathbf{w},b)}{\partial w_{n-1}} \right) = \left(\frac{\partial J(\mathbf{w},b)}{\partial w_0}, \frac{\partial J(\mathbf{w},b)}{\partial w_1}, \frac{\partial J(\mathbf{w},b)}{\partial w_2}, \frac{\partial J(\mathbf{w},b)}{\partial w_3} \right)$$

- $$\frac{\partial J(\mathbf{w},b)}{\partial b} \in \mathbb{R}_1$$

```c
double *compute_gradient(float **x, float y[m], double w[][1], double b){
    /*

    Computes the gradient for linear regression
    Args:
      X (ndarray (m,n)): Data, m examples with n features
      y (ndarray (m,1)) : target values
      w (ndarray (n,1)) : model parameters
      b (scalar)       : model parameter

    Returns:
      dj_dw (ndarray (n,1)): The gradient of the cost w.r.t. the parameters w.
      dj_db (scalar):       The gradient of the cost w.r.t. the parameter b.

      grads (ndarray ((n+1), 1)): dj_dw and dj_db.
    */

    double *grads=(double *)malloc((n + 1)*sizeof(double));
    if (grads ==NULL)
    {
        perror("Error in allocating memory");
        free(grads);
    }

    double *f_wb=(double *)malloc(m*sizeof(double));
    if (f_wb ==NULL)
    {
        perror("Error in allocating memory");
        free(f_wb);
    }

    double p;
    double dj_dw;
    double dj_db=0;

    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < 1; j++)
        {
            p=0;
            for (int k = 0; k < n; k++)
            {
                p += x[i][k]*w[k][j];
            }
            p += b;
            f_wb[i]=p;
        }

        dj_db += f_wb[i] - y[i];
    }
    dj_db /= m;


    for (int i = 0; i < n; i++)
    {
        dj_dw=0;
        for (int j = 0; j < m; j++)
        {
            dj_dw += (f_wb[j] - y[j])*x[j][i];
        }
        dj_dw /= m;

        grads[i]=dj_dw;
    }

    grads[n]=dj_db;

    free(f_wb);

    return grads;
}

```

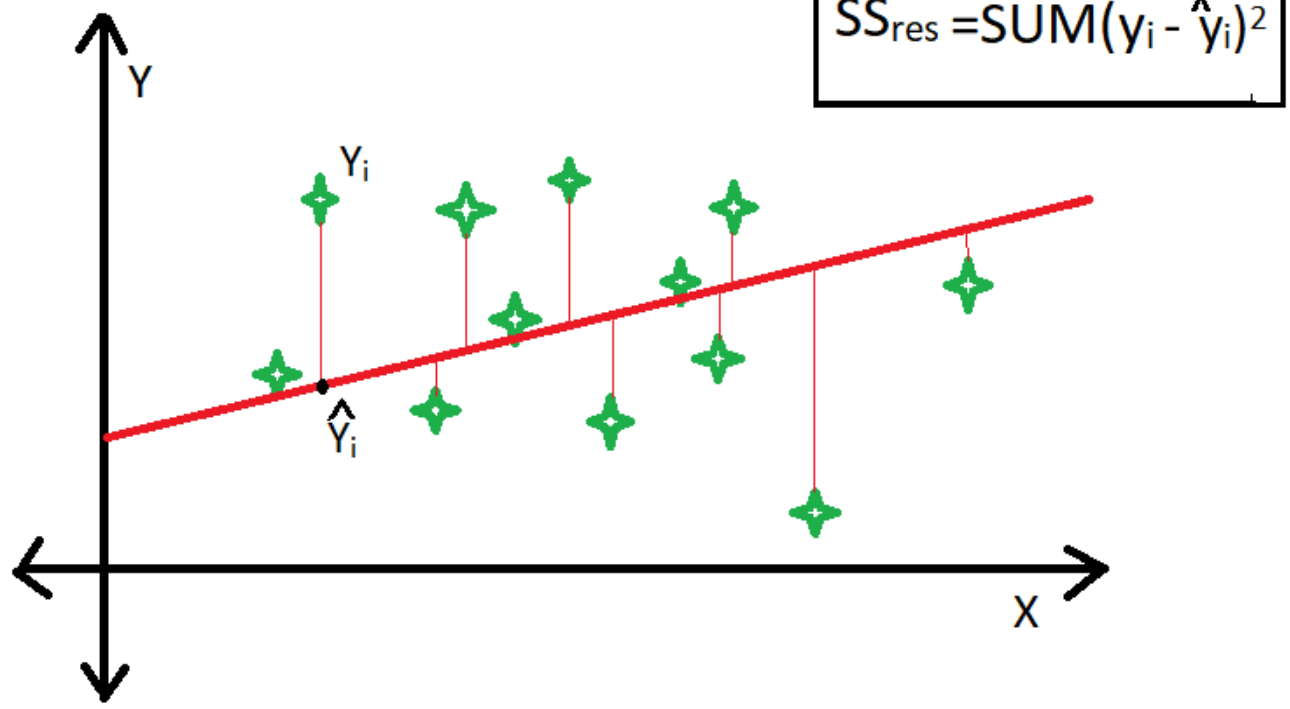
6.0 Evaluating our model by compute R-squared

R-squared is the proportion of the variance in the dependent variable that is explained by the independent variables in a regression model. It ranges from 0 to 1, where 0 means no relationship and 1 means a perfect fit

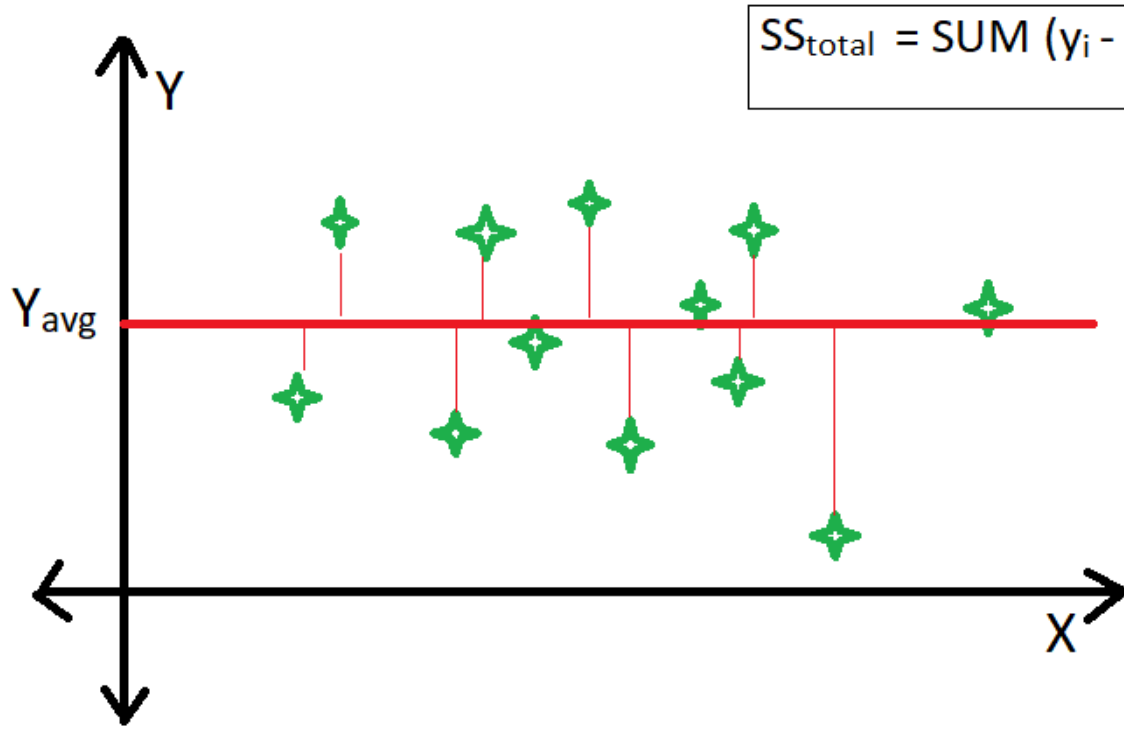
R-squared(coefficient of Determination) tells us How well our model is performing or how well our model's predictions match the real results. A higher R-squared means our model is doing a better job predicting.

$$R^2 = 1 - \frac{Residual\ Sum\ of\ Squares\ (\mathbf{ss}_{res})}{Total\ Sum\ of\ Squares\ (\mathbf{ss}_{total})}$$

$$SS_{res} = SUM(y_i - \hat{y}_i)^2$$



$$SS_{total} = SUM\ (y_i - y_{avg})^2$$



$$R^2 = 1 - \frac{\mathbf{SS}_{res}}{\mathbf{SS}_{total}}$$

$$R^2 = 1 - \frac{\sum_{i=0}^{(m-1)} (f_{w,b}(x^{(i)}) - y^i)^2}{\sum_{i=0}^{(m-1)} (y_{mean}^{(i)} - y^{(i)})^2}$$

```
double r_squared(float **x, float y[m], double w[][1], double b){

    /*

    Computes the gradient for linear regression
    Args:
      X (ndarray (m,n)): Data, m examples with n features
      y (ndarray (m,1)) : target values
      w (ndarray (n,1)) : model parameters
      b (scalar)       : model parameter

    Returns:
      r_square (scalar): r square score.
      r_square = (1 - rss/tss)
      rss : measure of sum diff btn actual(y) value and predicted(f_wb) value.(y - f_wb)
      tss : measure of total variance in actual(y) value. (y - f_mean)

    */

    double r_square;
    double f_wb;
    double y_mean;
    double y_sum;
    double rss;
    double tss;

    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < 1; j++)
        {
            f_wb=0;
            for (int k = 0; k < n; k++)
            {
                f_wb += x[i][k]*w[k][j];
            }
            f_wb +=b;
        }

        rss += pow((y[i] - f_wb), 2);
        y_sum += y[i];

    }
    y_mean = y_sum/m;

    for (int i = 0; i < m; i++)
    {
        tss += pow((y[i] - y_mean), 2);
    }

    r_square = 1 - (rss/tss);


    return r_square;
}
```

The main function

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

#define m 100
#define n 4
#define lr 1e-7
#define num_iter 1000
#define num_col_x 5
#define col_y 4
#define FILE_NAME "data/houses.txt"

float **load_data(const char *file_id);

double *predict(float **x, double w[][1], double b);
double compute_cost(float **x, float y[m], double w[][1], double b);
double *compute_gradient(float **x, float y[m], double w[][1], double b);
double r_squared(float **x, float y[m], double w[][1], double b);


int main(){

    float **data=load_data(FILE_NAME);
    if (data == NULL)
    {
        return EXIT_FAILURE;
    }

    //alocate the memory for the input array dynamically
    float **x=(float **)malloc(m*sizeof(float *));
    for (int i = 0; i < m; i++)
    {
       x[i] = (float *)malloc(n*sizeof(float));
       if (x==NULL)
       {
        perror("Error in locatting memory");
        for (int j = 0; j < i; j++)
        {
           free(x[j]);
        }
        free(x);
       }
    }

    //allocate the memory for the output array dynamically
    float *y=(float *)malloc(m*sizeof(float));
    if (y==NULL)
    {
        perror("Error in allocating memory");
        free(y);
    }

    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            x[i][j] = data[i][j];
        }
        y[i] = data[i][col_y];
    }

    double b_init = 785.1811367994083;
    double w_init[][1] = { 0.39133535, 18.75376741, -53.36032453, -26.42131618};

    for (int i = 0; i < num_iter; i++)
    {
        //compute cost
        double J_wb=compute_cost(x, y, w_init, b_init);

        // Calculate the gradient and update the parameters using gradient_function
        double *grads=compute_gradient( x, y, w_init, b_init);

        //R_Squared
        double r_square=r_squared( x, y, w_init, b_init);

        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < 1; j++)
            {
                w_init[i][j] = w_init[i][j] - (lr * grads[i]);
            }
        }
        b_init =b_init - (lr * grads[n]);

    }

    //free the allocated memory
    for (int i = 0; i < m; i++)
    {
        free(data[i]);
        free(x[i]);
    }
    free(data);
    free(x);
    free(y);
    return 0;
}

```