# Lab 05 - Curve Fitting Using Regression Analysis

## Objective

1. Understand and implement Regression Analysis.

---

## Theory

Regression analysis is a statistical method used to examine the relationship between two or more variables. In its simplest form, linear regression, the goal is to model the relationship between a dependent variable $y$ and an independent variable $x$ by fitting a straight line to the observed data. This line, known as the regression line, is typically expressed as $y = mx + c$, where $m$ is the slope and $c$ is the intercept.

The method of least squares is commonly used to determine the best-fitting line by minimizing the sum of the squares of the vertical distances (errors) between the observed values and the values predicted by the line. Regression analysis helps in predicting the value of the dependent variable based on the independent variable and is widely used in data analysis, forecasting, and scientific research to identify trends and make informed decisions.

---

## Algorithm

1. Input the number of data points, $n$.
2. Input the data values for $x[]$ and $y[]$.
3. Compute the following summations:
   - $\sum x$
   - $\sum y$
   - $\sum x^2$
   - $\sum y^2$
   - $\sum xy$
4. Calculate the regression coefficients (slope $m$ and intercept $c$) using the least squares formulas:
   - $m = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2}$
   - $c = \frac{\sum y - m \sum x}{n}$
5. Formulate the regression equation: $y = mx + c$.
6. (Optional) Use the regression equation to predict $y$ for a given $x$.

---

## Source Code

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

// Define maximum size of the arrays
# define SIZE 20

// Macro to get the length of an array (for statically sized arrays)
#define ARRAY_LENGTH(arr) (sizeof(arr) / sizeof((arr)[0]))

// Enum for regression type
typedef enum {
    LINEAR = 1,
    EXPONENTIAL, // y = a * exp(bx)
    LOGARITHMIC, // y = a + b*ln(x)
    POWER        // y = a * x^b
} RegressionType;

// Global variables for regression coefficients
double slope_m = 0.0, intercept_c = 0.0;

// Function to calculate the summation of an array
double summation(double arr[], int n) {
    double sum = 0.0;
    for(int i = 0; i < n; i++) {
        sum += arr[i];
    }
    return sum;
}

// Function to calculate the summation of products of two arrays
double summation_product(double arr1[], double arr2[], int n) {
    double sum = 0.0;
    for(int i = 0; i < n; i++) {
        sum += arr1[i] * arr2[i];
    }
    return sum;
}

// Function to calculate the summation of squares of an array
double summation_square(double arr[], int n) {
    double sum = 0.0;
    for(int i = 0; i < n; i++) {
        sum += arr[i] * arr[i];
    }
    return sum;
}

// Regression analysis function: fits a regression model to the data points
void regression_fit(double x[], double y[], int n, RegressionType type) {
    if (n <= 0) {
        printf("No data points provided.\n");
        return;
    }

    double sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
    double m = 0.0, c = 0.0;

    // Arrays for transformed data (for transcendental regression)
    double X[SIZE], Y[SIZE];
```

```c
    switch(type) {
        case LINEAR:
            // Linear regression: y = m*x + c
            sum_x = summation(x, n);
            sum_y = summation(y, n);
            sum_xy = summation_product(x, y, n);
            sum_x2 = summation_square(x, n);

            m = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x * sum_x);
            c = (sum_y - m * sum_x) / n;

            printf("Linear Regression: y = (%.4lf)x + (%.4lf)\n", m, c);
            break;

        case EXPONENTIAL:
            // Exponential regression: y = a * exp(bx)
            // Take ln(y) = ln(a) + b*x
            for(int i = 0; i < n; i++) {
                X[i] = x[i];
                if (y[i] <= 0) {
                    printf("Error: y[%d] <= 0, cannot take log for exponential regression.\n", i);
                    return;
                }
                Y[i] = log(y[i]);
            }
            sum_x = summation(X, n);
            sum_y = summation(Y, n);
            sum_xy = summation_product(X, Y, n);
            sum_x2 = summation_square(X, n);

            m = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x * sum_x); // b
            c = (sum_y - m * sum_x) / n; // ln(a)
            printf("Exponential Regression: y = (%.4lf) * exp(%.4lf * x)\n", exp(c), m);
            break;

        case LOGARITHMIC:
            // Logarithmic regression: y = a + b*ln(x)
            for(int i = 0; i < n; i++) {
                if (x[i] <= 0) {
                    printf("Error: x[%d] <= 0, cannot take log for logarithmic regression.\n", i);
                    return;
                }
                X[i] = log(x[i]);
                Y[i] = y[i];
            }
            sum_x = summation(X, n);
            sum_y = summation(Y, n);
            sum_xy = summation_product(X, Y, n);
            sum_x2 = summation_square(X, n);

            m = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x * sum_x); // b
            c = (sum_y - m * sum_x) / n; // a
            printf("Logarithmic Regression: y = (%.4lf) + (%.4lf) * ln(x)\n", c, m);
            break;

        case POWER:
            // Power regression: y = a * x^b
            // Take ln(y) = ln(a) + b*ln(x)
            for(int i = 0; i < n; i++) {
                if (x[i] <= 0 || y[i] <= 0) {
                    printf("Error: x[%d] <= 0 or y[%d] <= 0, cannot take log for power regression.\n", i, i);
                    return;
                }
                X[i] = log(x[i]);
                Y[i] = log(y[i]);
            }
            sum_x = summation(X, n);
            sum_y = summation(Y, n);
            sum_xy = summation_product(X, Y, n);
            sum_x2 = summation_square(X, n);

            m = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x * sum_x); // b
            c = (sum_y - m * sum_x) / n; // ln(a)
            printf("Power Regression: y = (%.4lf) * x^(%.4lf)\n", exp(c), m);
            break;

        default:
            printf("Invalid regression type.\n");
            return;
    }

    // Update global variables for prediction
    slope_m = m;
    intercept_c = c;
}

// Prediction function for different regression types
double regression_predict(double x, RegressionType type) {
    switch(type) {
        case LINEAR:
            return slope_m * x + intercept_c;
        case EXPONENTIAL:
            return exp(intercept_c) * exp(slope_m * x); // y = a * exp(bx)
        case LOGARITHMIC:
            if (x <= 0) {
                printf("Error: x <= 0, cannot take log for prediction.\n");
                return NAN;
            }
            return intercept_c + slope_m * log(x); // y = a + b*ln(x)
        case POWER:
            if (x <= 0) {
                printf("Error: x <= 0, cannot take log for prediction.\n");
                return NAN;
            }
            return exp(intercept_c) * pow(x, slope_m); // y = a * x^b
        default:
            printf("Invalid regression type for prediction.\n");
            return NAN;
    }
}

// Driver function
int main() {
    double x[SIZE], y[SIZE];
    int n, choice;

    // Input number of data points
    printf("Enter the number of data points (max %d): ", SIZE);
    scanf("%d", &n);

    // Input data points
    printf("Enter the x and y values:\n");
    for(int i = 0; i < n; i++) {
        printf("x[%d]: ", i);
        scanf("%lf", &x[i]);
        printf("y[%d]: ", i);
        scanf("%lf", &y[i]);
    }

    // Select regression type
    printf("\nSelect regression type:\n");
    printf("1. Linear (y = m*x + c)\n");
    printf("2. Exponential (y = a*exp(bx))\n");
    printf("3. Logarithmic (y = a + b*ln(x))\n");
    printf("4. Power (y = a*x^b)\n");
    printf("Enter your choice (1-4): ");
    scanf("%d", &choice);

    RegressionType reg_type = (RegressionType)choice;

    // Perform regression analysis (Also update global variables)
    regression_fit(x, y, n, reg_type);

    // Example prediction
    double x_predict;
    printf("\nEnter a value of x to predict y: ");
    scanf("%lf", &x_predict);
    double y_predict = regression_predict(x_predict, reg_type);
    if (!isnan(y_predict))
        printf("Predicted value of y for x = %.4lf is: %.4lf\n", x_predict, y_predict);
    else
        printf("Prediction failed due to invalid input.\n");

    return 0;
}
```

**Ouptut: Linear**

```
Enter the number of data points (max 20):   3
Enter the x and y values:
x[0]: 3
y[0]: 5
x[1]: 4
y[1]: 6
x[2]: 5
y[2]: 8

Select regression type:
1. Linear (y = m*x + c)
2. Exponential (y = a*exp(bx))
3. Logarithmic (y = a + b*ln(x))
4. Power (y = a*x^b)
Enter your choice (1-4): 1
Linear Regression: y = (1.5000)x + (0.3333)

Enter a value of x to predict y: 6
Predicted value of y for x = 6.0000 is: 9.3333
```

**Output: Power**

```
Enter the number of data points (max 20): 5
Enter the x and y values:
x[0]: 1
y[0]: 0.5
x[1]: 2
y[1]: 2
x[2]: 3
y[2]: 4.5
x[3]: 4
y[3]: 8
x[4]: 5
y[4]: 12.5

Select regression type:
1. Linear (y = m*x + c)
2. Exponential (y = a*exp(bx))
3. Logarithmic (y = a + b*ln(x))
4. Power (y = a*x^b)
Enter your choice (1-4): 4
Power Regression: y = (0.5000) * x^(2.0000)

Enter a value of x to predict y: 6
Predicted value of y for x = 6.0000 is: 18.0000
```

---

## Discussion

In this lab, various regression models: linear, exponential, logarithmic, and power were implemented and tested using C. The code allows users to input data points and select the regression type, after which it computes the best-fit equation and predicts values for new inputs. The results for both linear and power regression matched theoretical expectations, confirming the correctness of the implementation.

The flexibility to handle different regression types demonstrates the practical utility of regression analysis in modeling diverse real-world relationships. The use of logarithmic and exponential transformations for non-linear models was particularly insightful, as it enabled fitting curves that are not directly linearizable. Error handling for invalid inputs (such as negative values for logarithmic or exponential regression) was also incorporated, improving robustness.

Overall, the lab reinforced the importance of regression analysis in data science and engineering, and provided hands-on experience in translating mathematical formulas into working code.

---

## Conclusion

In this lab, we successfully explored and implemented different regression techniques to fit curves to data using C programming. By applying linear, exponential, logarithmic, and power regression models, we gained practical insights into how mathematical concepts are translated into algorithms and code. The results validated the effectiveness of regression analysis for modeling and predicting relationships between variables. This exercise enhanced our understanding of both the theoretical and practical aspects of regression, which is fundamental in data analysis and engineering applications.

---

*Name: Kushal Prasad Joshi*
*Roll No: 230345*
*Faculty: Science and Technology*
*Program: BE Computer*
*Group: B*