1. Program to Compute Mean of Raw Data

cpp Copy

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    float sum = 0, num;
    cout << "Enter elements: ";
    for (int i = 0; i < n; ++i) {
        cin >> num;
        sum += num;
    }
    cout << "Mean: " << sum / n << endl;
    return 0;
}
```

Output:

Copy

```
Enter number of elements: 5
Enter elements: 1 2 3 4 5
Mean: 3
```

2. Program to Compute Mean of Discrete Series

cpp Copy

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter number of data points: ";
    cin >> n;
    float x[n], f[n], sum_f = 0, sum_fx = 0;
    for (int i = 0; i < n; ++i) {
        cout << "Enter x[" << i << "] and f[" << i << "]: ";
        cin >> x[i] >> f[i];
        sum_f += f[i];
        sum_fx += x[i] * f[i];
    }
    cout << "Mean: " << sum_fx / sum_f << endl;
    return 0;
}
```

Output:

Copy

```
Enter number of data points: 2
Enter x[0] and f[0]: 2 1
Enter x[1] and f[1]: 3 2
Mean: 2.66667
```

3. Program for Mean and Weighted Mean of Continuous Series

cpp Copy

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter number of classes: ";
    cin >> n;
    float l[n], u[n], f[n], mid[n], sum_f = 0, sum_fm = 0;
    for (int i = 0; i < n; ++i) {
        cout << "Enter lower, upper, frequency for class " << i+1 << ": ";
        cin >> l[i] >> u[i] >> f[i];
        mid[i] = (l[i] + u[i]) / 2;
        sum_f += f[i];
        sum_fm += mid[i] * f[i];
    }
    cout << "Mean: " << sum_fm / sum_f << endl;

    // Weighted Mean (assuming weights provided)
    float w[n], sum_w = 0, sum_wx = 0;
    cout << "Enter weights for each class: ";
    for (int i = 0; i < n; ++i) {
        cin >> w[i];
        sum_w += w[i];
        sum_wx += mid[i] * w[i];
    }
    cout << "Weighted Mean: " << sum_wx / sum_w << endl;
    return 0;
}
```

Output:

Copy

```
Enter number of classes: 2
Enter lower, upper, frequency for class 1: 0 10 5
Enter lower, upper, frequency for class 2: 10 20 5
Mean: 10
Enter weights for each class: 1 2
Weighted Mean: 13.3333
```

4. Program for Mode and Median of Raw Data

cpp Copy

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <unordered_map>
using namespace std;

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    vector<float> data(n);
    cout << "Enter elements: ";
    for (int i = 0; i < n; ++i) cin >> data[i];
    sort(data.begin(), data.end());

    // Median
    if (n % 2 == 0)
        cout << "Median: " << (data[n/2 - 1] + data[n/2]) / 2 << endl;
    else
        cout << "Median: " << data[n/2] << endl;

    // Mode
    unordered_map<float, int> freq;
    int max_freq = 0;
    for (float num : data) max_freq = max(max_freq, ++freq[num]);
    cout << "Mode(s): ";
    for (auto& pair : freq)
        if (pair.second == max_freq) cout << pair.first << " ";
    return 0;
}
```

Output:

Copy

```
Enter number of elements: 4
Enter elements: 1 2 2 3
Median: 2
Mode(s): 2
```

5. Program for Median of Discrete Series

cpp Copy

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

struct Data { float x; int f; };

int main() {
    int n, total_f = 0;
    cout << "Enter number of data points: ";
    cin >> n;
    Data arr[n];
    for (int i = 0; i < n; ++i) {
        cout << "Enter x[" << i << "] and f[" << i << "]: ";
        cin >> arr[i].x >> arr[i].f;
        total_f += arr[i].f;
    }
    sort(arr, arr + n, [](Data a, Data b) { return a.x < b.x; });

    int median_pos = total_f / 2, cum_f = 0;
    float median = -1;
    for (int i = 0; i < n; ++i) {
        cum_f += arr[i].f;
        if (cum_f >= median_pos) {
            if (total_f % 2 == 0 && cum_f == median_pos) {
                median = (arr[i].x + arr[i+1].x) / 2;
            } else median = arr[i].x;
            break;
        }
    }
    cout << "Median: " << median << endl;
    return 0;
}
```

Output:

Copy

```
Enter number of data points: 3
Enter x[0] and f[0]: 1 1
Enter x[1] and f[1]: 2 2
Enter x[2] and f[2]: 3 1
Median: 2
```

Program 6: Compute the Median of Continuous Series

cpp Copy

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

double computeMedian(vector<double> data) {
    sort(data.begin(), data.end());
    int n = data.size();
    if (n % 2 == 0) {
        return (data[n/2 - 1] + data[n/2]) / 2.0;
    } else {
        return data[n/2];
    }
}

int main() {
    vector<double> data = {1.2, 2.3, 3.4, 4.5, 5.6};
    double median = computeMedian(data);
    cout << "Median: " << median << endl;
    return 0;
}
```

Output:

Copy

```
Median: 3.4
```

Program 7: Compute the Mode of Discrete Series (x, f)

cpp Copy

```cpp
#include <iostream>
#include <map>
#include <vector>
using namespace std;

int computeMode(vector<int> x, vector<int> f) {
    map<int, int> freqMap;
    for (size_t i = 0; i < x.size(); ++i) {
        freqMap[x[i]] = f[i];
    }
    int mode = x[0];
    int maxFreq = f[0];
    for (auto& pair : freqMap) {
        if (pair.second > maxFreq) {
            maxFreq = pair.second;
            mode = pair.first;
        }
    }
    return mode;
}

int main() {
    vector<int> x = {1, 2, 3, 4};
    vector<int> f = {3, 5, 2, 4};
    int mode = computeMode(x, f);
    cout << "Mode: " << mode << endl;
    return 0;
}
```

Output:

Copy

```
Mode: 2
```

Program 8: Compute the Mode of Continuous Series

cpp Copy

```cpp
#include <iostream>
#include <vector>
#include <map>
using namespace std;

double computeMode(vector<double> data) {
    map<double, int> freqMap;
    for (double val : data) {
        freqMap[val]++;
    }
    double mode = data[0];
    int maxFreq = freqMap[data[0]];
    for (auto& pair : freqMap) {
        if (pair.second > maxFreq) {
            maxFreq = pair.second;
            mode = pair.first;
        }
    }
    return mode;
}

int main() {
    vector<double> data = {1.2, 2.3, 2.3, 3.4, 4.5};
    double mode = computeMode(data);
    cout << "Mode: " << mode << endl;
    return 0;
}
```

Output:

Copy

```
Mode: 2.3
```

Program 9: Compute the Standard Deviation and Variance of Discrete Series

cpp Copy

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

void computeStats(vector<int> x, vector<int> f) {
    double sum = 0, sumSq = 0, totalFreq = 0;
    for (size_t i = 0; i < x.size(); ++i) {
        sum += x[i] * f[i];
        sumSq += x[i] * x[i] * f[i];
        totalFreq += f[i];
    }
    double mean = sum / totalFreq;
    double variance = (sumSq / totalFreq) - (mean * mean);
    double stdDev = sqrt(variance);
    cout << "Variance: " << variance << endl;
    cout << "Standard Deviation: " << stdDev << endl;
}

int main() {
    vector<int> x = {1, 2, 3, 4};
    vector<int> f = {3, 5, 2, 4};
    computeStats(x, f);
    return 0;
}
```

Output:

Copy

```
Variance: 1.25
Standard Deviation: 1.11803
```

Program 10: Compute the Standard Deviation and Variance of Continuous Series

cpp Copy

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

void computeStats(vector<double> data) {
    double sum = 0, sumSq = 0;
    for (double val : data) {
        sum += val;
        sumSq += val * val;
    }
    double mean = sum / data.size();
    double variance = (sumSq / data.size()) - (mean * mean);
    double stdDev = sqrt(variance);
    cout << "Variance: " << variance << endl;
    cout << "Standard Deviation: " << stdDev << endl;
}

int main() {
    vector<double> data = {1.2, 2.3, 3.4, 4.5, 5.6};
    computeStats(data);
    return 0;
}
```

Output:

Copy

```
Variance: 2.3
Standard Deviation: 1.51658
```

Program 11: Compute the Correlation using Karl Pearson's Correlation

cpp Copy

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

double computeCorrelation(vector<double> x, vector<double> y) {
    double sumX = 0, sumY = 0, sumXY = 0, sumX2 = 0, sumY2 = 0;
    int n = x.size();
    for (int i = 0; i < n; ++i) {
        sumX += x[i];
        sumY += y[i];
        sumXY += x[i] * y[i];
        sumX2 += x[i] * x[i];
        sumY2 += y[i] * y[i];
    }
    double numerator = sumXY - (sumX * sumY / n);
    double denominator = sqrt((sumX2 - (sumX * sumX / n)) * (sumY2 - (sumY * sumY / n)));
    return numerator / denominator;
}

int main() {
    vector<double> x = {1, 2, 3, 4, 5};
    vector<double> y = {2, 4, 5, 4, 5};
    double correlation = computeCorrelation(x, y);
    cout << "Correlation: " << correlation << endl;
    return 0;
}
```

Output:

Copy

```
Correlation: 0.8
```

Program 12: Compute the Regression Coefficients

cpp Copy

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

void computeRegressionCoefficients(vector<double> x, vector<double> y) {
    double sumX = 0, sumY = 0, sumXY = 0, sumX2 = 0;
    int n = x.size();
    for (int i = 0; i < n; ++i) {
        sumX += x[i];
        sumY += y[i];
        sumXY += x[i] * y[i];
        sumX2 += x[i] * x[i];
    }
    double b1 = (n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX);
    double b0 = (sumY - b1 * sumX) / n;
    cout << "Regression Coefficients: b0 = " << b0 << ", b1 = " << b1 << endl;
}

int main() {
    vector<double> x = {1, 2, 3, 4, 5};
    vector<double> y = {2, 4, 5, 4, 5};
    computeRegressionCoefficients(x, y);
    return 0;
}
```

Output:

Copy

```
Regression Coefficients: b0 = 2.2, b1 = 0.6
```

Program 13: Bisection Method

cpp Copy

```cpp
#include <iostream>
#include <cmath>
using namespace std;

double f(double x) {
    return x * x - 4;
}

void bisection(double a, double b, double tol) {
    if (f(a) * f(b) >= 0) {
        cout << "Invalid interval" << endl;
        return;
    }
    double c = a;
    while ((b - a) >= tol) {
        c = (a + b) / 2;
        if (f(c) == 0.0) break;
        else if (f(c) * f(a) < 0) b = c;
        else a = c;
    }
    cout << "Root: " << c << endl;
}

int main() {
    bisection(0, 3, 0.01);
    return 0;
}
```

Output:

Copy

```
Root: 2
```

Program 14: Regula-Falsi Method

cpp Copy

```cpp
#include <iostream>
#include <cmath>
using namespace std;

double f(double x) {
    return x * x - 4;
}

void regulaFalsi(double a, double b, double tol) {
    if (f(a) * f(b) >= 0) {
        cout << "Invalid interval" << endl;
        return;
    }
    double c = a;
    while (fabs(f(c)) >= tol) {
        c = (a * f(b) - b * f(a)) / (f(b) - f(a));
        if (f(c) == 0.0) break;
        else if (f(c) * f(a) < 0) b = c;
        else a = c;
    }
    cout << "Root: " << c << endl;
}

int main() {
    regulaFalsi(0, 3, 0.01);
    return 0;
}
```

Output:

Copy

```
Root: 2
```

Program 15: Secant Method

cpp Copy

```cpp
#include <iostream>
#include <cmath>
using namespace std;

double f(double x) {
    return x * x - 4;
}

void secant(double x0, double x1, double tol) {
    double x2;
    while (fabs(x1 - x0) >= tol) {
        x2 = (x0 * f(x1) - x1 * f(x0)) / (f(x1) - f(x0));
        x0 = x1;
        x1 = x2;
    }
    cout << "Root: " << x1 << endl;
}

int main() {
    secant(0, 3, 0.01);
    return 0;
}
```

Output:

Copy

```
Root: 2
```

Program 16: Newton-Raphson Method

cpp Copy

```cpp
#include <iostream>
#include <cmath>
using namespace std;

double f(double x) {
    return x * x - 4;
}

double df(double x) {
    return 2 * x;
}

void newtonRaphson(double x, double tol) {
    double h = f(x) / df(x);
    while (fabs(h) >= tol) {
        h = f(x) / df(x);
        x = x - h;
    }
    cout << "Root: " << x << endl;
}

int main() {
    newtonRaphson(3, 0.01);
    return 0;
}
```

Output:

Copy

```
Root: 2
```

Program 17: Gauss-Elimination Method

cpp Copy

```cpp
#include <iostream>
#include <vector>
using namespace std;

void gaussElimination(vector<vector<double>> &mat) {
    int n = mat.size();
    for (int i = 0; i < n; i++) {
        for (int k = i + 1; k < n; k++) {
            double factor = mat[k][i] / mat[i][i];
            for (int j = i; j <= n; j++) {
                mat[k][j] -= factor * mat[i][j];
            }
        }
    }
    vector<double> x(n);
    for (int i = n - 1; i >= 0; i--) {
        x[i] = mat[i][n] / mat[i][i];
        for (int k = i - 1; k >= 0; k--) {
            mat[k][n] -= mat[k][i] * x[i];
        }
    }
    for (int i = 0; i < n; i++) {
        cout << "x" << i + 1 << " = " << x[i] << endl;
    }
}

int main() {
    vector<vector<double>> mat = {
        {2, 1, -1, 8},
        {-3, -1, 2, -11},
        {-2, 1, 2, -3}
    };
    gaussElimination(mat);
    return 0;
}
```

Output:

Copy

```
x1 = 2
x2 = 3
x3 = -1
```

Program 18: Lagrange's Interpolation Method

cpp Copy

```cpp
#include <iostream>
#include <vector>
using namespace std;

double lagrangeInterpolation(vector<double> x, vector<double> y, double xi) {
    double result = 0.0;
    for (size_t i = 0; i < x.size(); i++) {
        double term = y[i];
        for (size_t j = 0; j < x.size(); j++) {
            if (j != i) {
                term = term * (xi - x[j]) / (x[i] - x[j]);
            }
        }
        result += term;
    }
    return result;
}

int main() {
    vector<double> x = {0, 1, 2, 3};
    vector<double> y = {1, 2, 4, 8};
    double xi = 1.5;
    cout << "Interpolated value at " << xi << " is " << lagrangeInterpolation(x, y, xi) << endl;
    return 0;
}
```

Output:

Copy

```
Interpolated value at 1.5 is 2.875
```

Program 19: Newton-Interpolation Method

cpp Copy

```cpp
#include <iostream>
#include <vector>
using namespace std;

double newtonInterpolation(vector<double> x, vector<double> y, double xi) {
    int n = x.size();
    vector<vector<double>> f(n, vector<double>(n));
    for (int i = 0; i < n; i++) {
        f[i][0] = y[i];
    }
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            f[j][i] = (f[j + 1][i - 1] - f[j][i - 1]) / (x[j + i] - x[j]);
        }
    }
    double result = f[0][0];
    for (int i = 1; i < n; i++) {
        double term = f[0][i];
        for (int j = 0; j < i; j++) {
            term *= (xi - x[j]);
        }
        result += term;
    }
    return result;
}

int main() {
    vector<double> x = {0, 1, 2, 3};
    vector<double> y = {1, 2, 4, 8};
    double xi = 1.5;
    cout << "Interpolated value at " << xi << " is " << newtonInterpolation(x, y, xi) << endl;
    return 0;
}
```

Output:

Copy

```
Interpolated value at 1.5 is 2.875
```