**Методы численного анализа**

**Отчёт по лабораторной работе №3.1**

**Метод наименьших квадратов**

**Вариант 1**

**Студент 2 курса 8 группы**

Агинский Антон Викторович

**Преподаватель**

Будник Анатолий Михайлович

ФПМИ БГУ  
2022

**Постановка задачи**

Методом наименьших квадратов построить аппроксимирующий многочлен функции степени на отрезке, полагая, что

и весовая функция в рассматриваемом пространстве равна .

Изначально на вход поступают значения в точках,

Требуется найти:

1) исходную таблицу ,

2) аппроксимирующий многочлен и его значения в следующих точках:

3) истинную погрешность в вышеуказанных точках,

4) погрешность приближения , равная среднеквадратичному отклонению,

5) провести анализ результатов.

**Алгоритм**

Строим таблицу значений , высчитывая значение функции в данных точках.

Элемент наилучшего приближения будем искать в пространстве всех действительных квадратично интегрируемых по весу функций .

ЭНП определим, как линейную комбинацию вида , где .

Коэффициенты найдем, решив следующую систему методом Гаусса:

Погрешность найдём по формуле .

Истинную погрешность в точках равна модулю разности значения функции в точке и значения аппроксимирующего многочлена в рассматриваемой точке.

**Листинг**

Функция для нахождения :

double f(double x) {

return (0.15 \* Math.exp(x) + 0.85 \* Math.sin(x));

}

Функция для нахождения решения СЛАУ методом Гаусса:

double[] gaussianElimination(double[][] A, double[] B)

{

int N = B.length;

for (int k = 0; k < N; k++)

{

int max = k;

for (int i = k + 1; i < N; i++)

if (Math.abs(A[i][k]) > Math.abs(A[max][k]))

max = i;

double[] temp = A[k];

A[k] = A[max];

A[max] = temp;

double t = B[k];

B[k] = B[max];

B[max] = t;

for (int i = k + 1; i < N; i++)

{

double factor = A[i][k] / A[k][k];

B[i] -= factor \* B[k];

for (int j = k; j < N; j++)

A[i][j] -= factor \* A[k][j];

}

}

double[] solution = new double[N];

for (int i = N - 1; i >= 0; i--)

{

double sum = 0.0;

for (int j = i + 1; j < N; j++)

sum += A[i][j] \* solution[j];

solution[i] = (B[i] - sum) / A[i][i];

}

for (int i = 0; i < solution.length; i++) {

System.out.println(solution[i]);

}

return solution;

}

Функция для нахождения :

double approximation(double[] c, double x) {

return (c[0] + c[1] \* x + c[2] \* Math.pow(x, 2) + c[3] \* Math.pow(x, 3) + c[4] \* Math.pow(x, 4) + c[5] \* Math.pow(x, 5));

}

Функция для решения поставленной задачи:

void leastSquares() {

double[] x = new double[11]; double[] f = new double[11];

int n = 10; int m = n / 2;

//*заполнение таблицы*

for (int i = 0; i < n + 1; i++) {

x[i] = 0.15 + 0.1 \* i;

f[i] = f(x[i]);

}

double sum;

double[][] A = new double[m + 1][m + 1];

double[] b = new double[m + 1];

//нахождение системы

for(int l = 0; l < m + 1; l++) {

for(int k = 0; k < m + 1; k++) {

sum = 0;

for (int i = 0; i < n + 1; i++)

sum += Math.pow(x[i], (k + l));

A[l][k] = sum;

}

sum = 0;

for (int i = 0; i < n + 1; i++)

sum += f[i] \* Math.pow(x[i], l);

b[l] = sum;

}

double[] c = gaussianElimination(A, b);

double x1 = 0.15 + 1 / 15, x2 = 0.7, x3 = 1.15 - 1 / 30;

//значения и погрешности в точках

System.out.println(approximation(c, x1));

System.out.println(f(x1) - approximation(c, x1));

System.out.println(approximation(c, x2));

System.out.println(f(x2) - approximation(c, x2));

System.out.println(approximation(c, x3));

System.out.println(f(x3) - approximation(c, x3));

sum = 0;

for (int i = 0; i < n + 1; i++)

sum += Math.pow(f(x[i]) - approximation(c, x[i]), 2);

//погрешность

System.out.println(Math.pow(sum, 0.5));

}

**Результаты и их анализ**

Исходная таблица значений:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.15 | 0.25 | 0.35 | 0.45 | 0.55 | 0.65 | 0.75 | 0.85 | 0.95 | 1.05 | 1.15 |
|  | 0.3012975490118018 | 0.4028971778695057 | 0.5043232686261222 | 0.6049675318180711 | 0.7042720972711696 | 0.801739569227718 | 0.8969429485117353 | 0.9895353721581475 | 1.0792596279683446 | 1.165957409464389 | 1.2495782856749078 |

Коэффициенты аппроксимирующего многочлена:

Значения в точках :

Истинная погрешность в данных точках:

Погрешность приближения :

Погрешности имеют порядок , значит с помощью найденного аппроксимирующего многочлена можно найти значение на отрезке с точностью .

**Методы численного анализа**

**Отчёт по лабораторным работам**

**№3.2 Интерполяционный многочлен в форме Лагранжа**

**№3.3 Минимизация остатка интерполирования**

**Вариант 1**

**Студент 2 курса 8 группы**

Агинский Антон Викторович

**Преподаватель**

Будник Анатолий Михайлович

ФПМИ БГУ  
2022

**Интерполяционный многочлен в форме Лагранжа**

**Постановка задачи**

Найти значения интерполяционного многочлена в форме Лагранжа , оценить истинную погрешность и остаток интерполирования в точках на отрезке . Сделать анализ полученных результатов.

**Алгоритм**

В каждой из точек найдем:

1) используя найденную в лабораторной работе №3.1 исходную таблицу значений, значения интерполяционного многочлена Лагранжа по следующей формуле:

2) истинную погрешность ,

3) остаток интерполирования, который вычисляется по формуле:

где

для оценки можем использовать:

**Листинг**

Функция для нахождения :

int factorial(int n) {

if (n == 0)

return 1;

return n \* factorial(n - 1);

}

Функция для нахождения остатка интерполирования в точке :

double interpolationReminder(double value, double[] x, double max) {

double w = 1;

for(int j = 0; j < x.length; j++)

w \*= (value - x[j]);

return (w \* max / factorial(x.length));

}

Функция для нахождения значения интерполяционного многочлена в форме Лагранжа в точке :

double lagrangePolinomial(double value, double[] x, double[] f) {

double sum = 0;

double product;

for(int k = 0; k < x.length; k++) {

product = 1;

for(int j = 0; j < x.length; j++) {

` if(j != k) {

product \*= (value - x[j])/(x[k] - x[j]);

}

}

product \*= f[k];

sum += product;

}

return sum;

}

**Результаты и их анализ**

Значения в точках :

Истинная погрешность в данных точках:

Значения остатка интерполирования в вышеуказанных точках:

Величина погрешностей показывает, что мы можем считать значения с помощью найденного интерполяционного многочлена на отрезке с точностью , что является лучшим приближением по сравнению с МНК. Это можно объяснить тем, что методом наименьших квадратов мы строили многочлен степени , а многочлен в форме Лагранжа степени .

**Минимизация остатка интерполирования**

**Постановка задачи**

Минимизировать остаток интерполирования с помощью узлов, являющихся корнями многочлена Чебышёва. Найти значения интерполяционного многочлена в форме Лагранжа , построенного с помощью новых узлов, оценить истинную погрешность на отрезке . Сделать анализ полученных результатов.

**Алгоритм**

Найдем значения новых узлов, которые являются корнями многочлена Чебышёва степени по следующей формуле ():

Используя найденные узлы составляем таблицу значений и с помощью программного кода из лабораторной работы №3.2 посчитаем значения интерполяционного многочлена Лагранжа по формуле и высчитаем погрешности по формуле в каждой из точек.

Оценку остатка интерполирования вычислим по следующей формуле:

**Листинг**

Функция для нахождения остатка интерполирования:

double interpolationReminder(double a, double b, int n, double max) {

return ((max \* Math.pow(b - a, n + 1))/(factorial(n + 1) \* Math.pow(2,2 \* n + 1)));

}

Функция для нахождения узла Чебышёва :

double ChebyshevNode(double a, double b, double k, double n) {

return ((a + b + (b - a) \* Math.cos(((2 \* k + 1) \* Math.PI)/

(2 \* (n + 1)))) / 2);

}

**Результаты и их анализ**

Таблица найденных узлов и значений функции в них:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Значения в точках :

Истинная погрешность в данных точках:

Остаток интерполирования:

Погрешности, полученные при интерполировании данным методом, получились лучшими, чем в предыдущей лабораторной работе. Мы можем считать значения с помощью найденного интерполяционного многочлена на отрезке с точностью . Для построения интерполяционного многочлена в обоих случаях мы использовали метод Лагранжа, но для минимизации остатка интерполирования вместо данных по условию узлов были взяты корни многочлена Чебышёва. Полиномы Чебышёва имеют наименьшее отклонение от нуля среди всех многочленов степени , поэтому погрешность при использовании чебышёвских полиномов будет также мала во всех точках. Также при интерполяции многочленами Чебышёва мы можем оценить остаток во всех точках сразу, так как в отличии от МНК и метода Лагранжа он не зависит от точки, которую мы рассматриваем.

**Методы численного анализа**

**Отчёт по лабораторной работе**

**№3.4 Интерполирование при равностоящих узлах**

**Вариант 1**

**Студент 2 курса 8 группы**

Агинский Антон Викторович

**Преподаватель**

Будник Анатолий Михайлович

ФПМИ БГУ  
2022

**Постановка задачи**

Вычислить приближенные значения в точке используя формулы интерполирования многочленом 3-ей степени при равностоящих узлах в начале таблицы. Найти таблицу конечных разностей, истинную погрешность и остаток интерполирования в вышеуказанной точке.

**Алгоритм**

Находим таблицу конечных разностей, храня её в матрице, состоящей из

столбца и четырёх строк (максимальный порядок разности, который понадобится для решения задачи – . В нулевой строчке храним значения , вычисленные в предыдущих лабораторных. Остальные строки заполняем следующим образом: в ячейке с индексом находится . Формулы для вычисления конечных разностей:

Интерполирование в точке :

Так как точка находится близко к левой границе отрезка, для нахождения приближённого значения функции в этом узле используем формулу интерполирования при равностоящих узлах в начале таблицы:

Формула остатка интерполирования при равностоящих узлах в начале таблицы:

Исходя из условия на , выберем эту величину равной .

**Листинг**

Функция для нахождения таблицы конечных разностей:

double[][] finiteDifferences(double[] f) {

int k = f.length;

double[][] finiteDifferences = new double[4][k];

for (int i = 0; i < k; i++) {

finiteDifferences[0][i] = f[i];

}

k--;

for (int i = 1; i < 4; i++) {

for (int j = 0; j < k; j++) {

finiteDifferences[i][j] =

finiteDifferences[i - 1][j + 1] - finiteDifferences[i - 1][j];

}

k--;

}

return finiteDifferences;

}

Функция для нахождения значения многочлена при интерполировании в начале таблицы при равностоящих узлах в точке :

double interpolationAtTheBeginning(double x0, double h, double[] f, double x, double[][] delta) {

double t = (x - x0) / h;

return (

f[0] + t \* delta[1][0] +

(t \* (t - 1) \* 0.5 \* delta[2][0]) +

(t \* (t - 1) \* (t - 2) \* (1.0 / 6.0) \* delta[3][0])

);

}

Функция для нахождения при интерполировании в начале таблицы:

double reminderForInterpolation(double y, double x0, int k, double h, double x) {

double t = (x – x0) / h;

double reminder = Math.pow(h, k + 1) \* f(y)/factorial(k + 1);

for (int i = 0; i < k + 1; i++)

reminder \*= (t - i);

return reminder;

}

**Результаты и их анализ**

Таблица конечных разностей ():

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Значение в точке :

Истинная погрешность в данных точках:

Значения остатка интерполирования в вышеуказанных точках:

Величина погрешностей показывает, что мы можем считать значения с помощью найденного интерполяционного многочлена на отрезке с точностью . Погрешности получились больше, чем при нахождении приближения другими методами, однако сейчас мы строили многочлен третьей степени и если сравнивать, например, с МНК, то несмотря на меньшую степень, погрешности имеют одинаковый порядок. Из этого можно сделать вывод, что по указанным формулам приближенные значения можно находить многочленами меньшей степени, но условием их применения является равностоящая сетка узлов, тогда как другими методами можно строить приближение по любым узлам.