**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования  
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ имени Н.Э.БАУМАНА  
(национальный исследовательский университет)»**

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

**Лабораторная работа № 3**

“Приближение функций. Сплайн-интерполяция”

по дисциплине «Численные методы»

Работу выполнил

студент группы ИУ9-62Б

Сербин Денис

Москва, 2022

# **1. Цель работы**

Целью работы является применение сплайн-интерполяции для заданной функции, вычисление значений сплайна 3-го порядка и абсолютной погрешности.

**2. Постановка задачи**

1. Протабулировать функцию f(x) на отрезке [а, b] с шагом h=(b-a) /10 и распечатать таблицу (x,y) i =0,..., n Для полученных узлов (x,y), i=0,..., n

2. Построить кубический сплайн (распечатать массивы а, b, с и d).

3. Вычислить значения f(x) в точках x = a + (1-1/2) \* h.

4. Вычислить значения оригинальной функции и cплайна в произвольной точке, задаваемой с экрана.

**3. Индивидуальный вариант**

y =

**4. Описание алгоритма**

Сплайном k-го порядка называется функция, проходящая через все узлы (хi, уi), i = 0, ..., n ямляющаяся многочленом k-й степени на каждом отрезке разбиения [,] в отдельности и имеющая (k - def) непрерыяных производных, на отрезке [,]. Наиболее употребительны сплайны третьего порядка с дефектом I (кубические сплайны).

Для каждого отрезка разбиения отыскиваем кубический сплайн в виде:

S(x) = , x € [, ], i=0, ...,n – 1

На частные условия накладываются условия:

(Сплайн проходит через все узлы);

(Непрерывность сплайна и его первых двух производных в промежуточных узлах);

; 0

(Краевые условия);

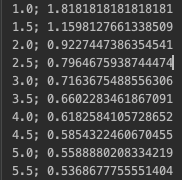
Эти условия приводят к трехдиагональной СЛАУ относительно коэффициента

Где h =

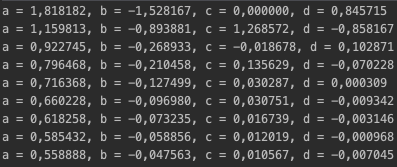
Остальные коэффициенты выражаются через :

**4. Реализация**

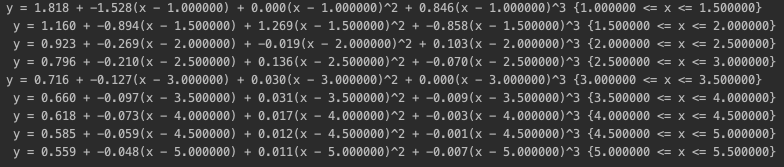
1. Протабулируем функцию и получим таблицу для значений функции с шагом 0.5:

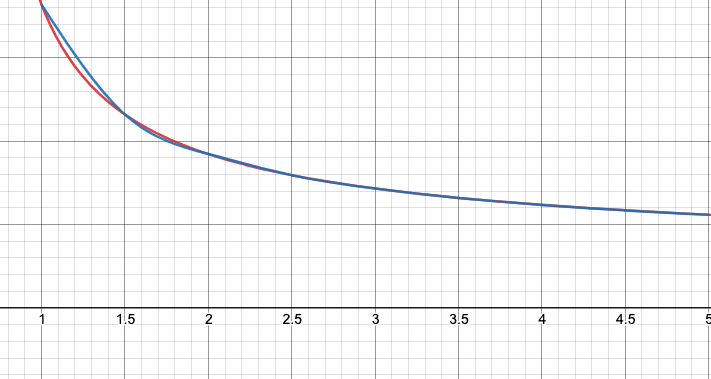


2. Распечатаем массивы а, b, с и d:

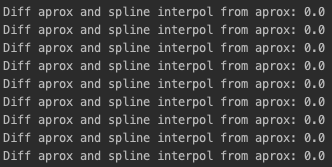


3. Вычислим значения f(x) в точках x = a + (1-1/2) \* h:



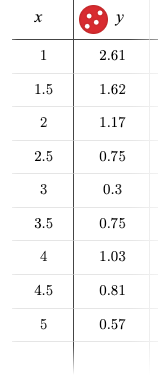


Разница между апроксимирующей функцией и сплайном в узлах интерполяции:

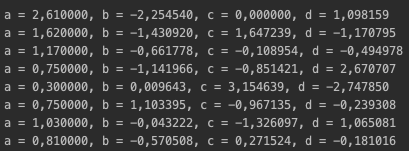


4. Вычислим значения оригинальной функции и cплайна в произвольной точке и вычислим их разницу:

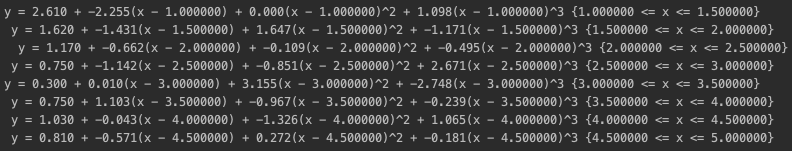
Возьмем точки из 2 лабораторной работы:

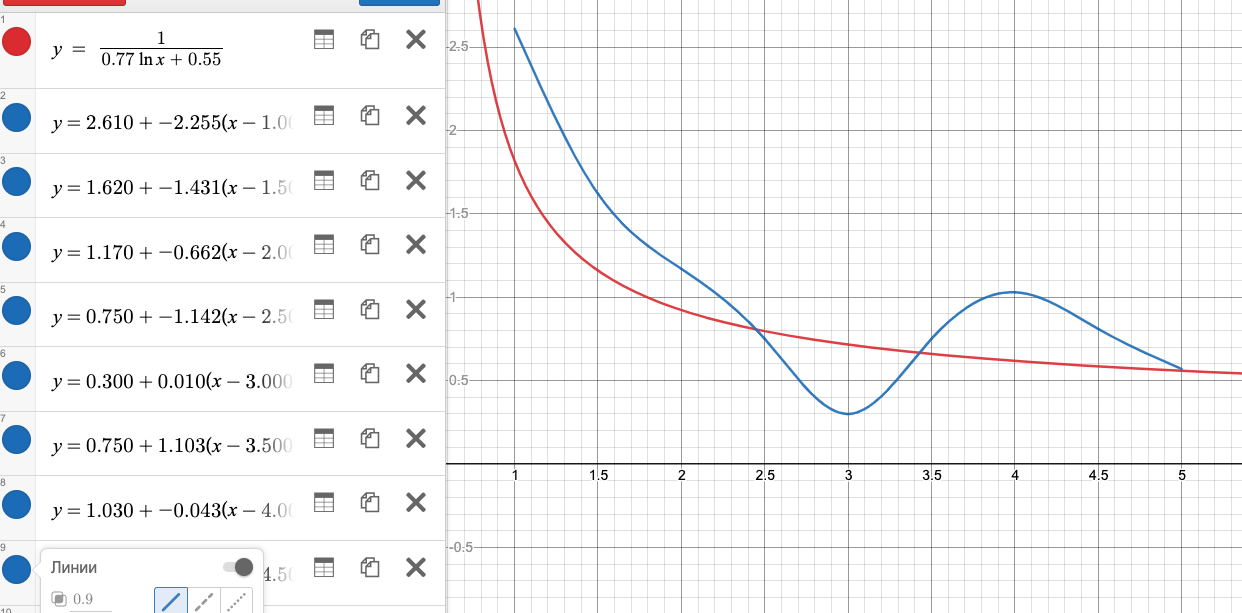
****

Коэффициенты:

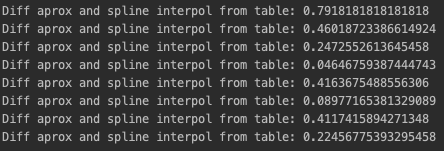


Значения сплайн функции в новых точках:





Разница между апроксимирующей функцией и сплайном в новых точках:



**5. Текст программы**

**public class** CubeSpline {  
  
 **private static double** func(**double** x){  
 **double** bCoef = 0.55;  
 **double** aCoef = 0.77;  
 **return** 1/(aCoef \*Math.*log*(x) + bCoef);  
 }  
  
 **private static void** solveBySLAY(ArrayList<Double> xs, ArrayList<Double> ys, **int** n, ArrayList<Double> cs, ArrayList<Double> as, ArrayList<Double> bs, ArrayList<Double> ds){  
 ArrayList<Double> ap = **new** ArrayList<>();  
 ArrayList<Double> bp = **new** ArrayList<>();  
 ArrayList<Double> dp = **new** ArrayList<>();  
 ArrayList<Double> cp = **new** ArrayList<>();  
 **double** h = (xs.get(1) - xs.get(0));  
 **for**(**int** i = 0; i < n - 2; i++){  
 ap.add((**double**)1);  
 cp.add((**double**)1);  
 }  
 **for**(**int** i = 1;i < n - 1; i++){  
 bp.add((**double**) 4);  
 **double** yh = 3\*(ys.get(i+1) - 2\*ys.get(i) + ys.get(i-1));  
 dp.add(yh / (h\*h));  
 }  
  
 ArrayList<Double> alpha = **new** ArrayList<>();  
 ArrayList<Double> beta = **new** ArrayList<>();  
 ArrayList<Double> x = **new** ArrayList<>();  
  
 *straightRun*(ap, bp, cp, dp, alpha, beta, n-2);  
 *reverseRun*(x, alpha, beta, n-2);  
  
 cs.add((**double**)0);  
 **for**(**int** i = 1; i < n - 1; i++){  
 cs.add(x.get(i - 1));  
 }  
 cs.add((**double**)0);  
  
 **for**(**int** i = 0; i < n-1; i++){  
 as.add(ys.get(i));  
 **double** bFirst = (ys.get(i+1) - ys.get(i)) / h;  
 **double** bSecond = ((h/3) \* (cs.get(i+1) + 2\*cs.get(i)) );  
 bs.add(bFirst - bSecond);  
 ds.add((cs.get(i+1) - cs.get(i)) / (3\*h));  
 }  
 }  
  
 **private static void** straightRun(ArrayList<Double> a, ArrayList<Double> b,  
 ArrayList<Double> c, ArrayList<Double> d,  
 ArrayList<Double> alpha, ArrayList<Double> beta, **int** size){  
 alpha.add(- c.get(0) / b.get(0));  
 beta.add(d.get(0) / b.get(0));  
 **for**(**int** i = 1; i < size; i ++){  
 **double** divider = (a.get(i - 1) \* alpha.get(i - 1)) + b.get(i);  
 **double** betaDivisible = (d.get(i) - (a.get(i - 1) \* beta.get(i - 1))) / divider;  
 **if**(i != size - 1) {  
 **double** alphaDivisible = -c.get(i) / divider;  
 alpha.add(alphaDivisible);  
 }  
 beta.add(betaDivisible);  
 }  
 }  
  
 **private static void** reverseRun(ArrayList<Double> x, ArrayList<Double> alpha, ArrayList<Double> beta, **int** size){  
 **for**(**int** i = 0; i < size; i++){  
 x.add((**double**) 0);  
 }  
 x.set(x.size() - 1, beta.get( beta.size() - 1 ));  
 **for** (**int** i = size - 2; i > -1; i--){  
 **double** xIter = alpha.get(i) \* x.get(i + 1) + beta.get(i);  
 x.set(i, xIter);  
 }  
 }  
  
 **private static void** checkRightSpline(){  
 ArrayList<Double> xs = **new** ArrayList<>(Arrays.*asList*(1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5));  
 ArrayList<Double> ys = **new** ArrayList<>();  
  
 ArrayList<Double> cs = **new** ArrayList<>();  
 ArrayList<Double> as = **new** ArrayList<>();  
 ArrayList<Double> bs = **new** ArrayList<>();  
 ArrayList<Double> ds = **new** ArrayList<>();  
  
 **int** n = xs.size() - 1;  
 **for** (Double x : xs) {  
 ys.add(*func*(x));  
 }  
  
 *solveBySLAY*(xs, ys,n+1, cs, as, bs, ds);  
 **for**(**int** i = 0; i < xs.size() - 1; i++){  
 *countSpline*(i, 0.5, xs, cs, as, bs, ds);  
 }  
  
 *makeDiffSpline*(ys, xs, cs, as, bs, ds);  
 }  
  
 **private static void** solveSpline(){  
  
 ArrayList<Double> xs2 = **new** ArrayList<>(Arrays.*asList*(1.00, 1.50, 2.00, 2.50, 3.00, 3.50, 4.00, 4.50, 5.00));  
 ArrayList<Double> ys = **new** ArrayList<>();  
 **for** (Double x : xs2) {  
 ys.add(*func*(x));  
 }  
 ArrayList<Double> ys2 = **new** ArrayList<>(Arrays.*asList*(2.61, 1.62, 1.17, 0.75, 0.3, 0.75, 1.03, 0.81, 0.57));  
  
 ArrayList<Double> cs = **new** ArrayList<>();  
 ArrayList<Double> as = **new** ArrayList<>();  
 ArrayList<Double> bs = **new** ArrayList<>();  
 ArrayList<Double> ds = **new** ArrayList<>();  
  
 **int** n = xs2.size() - 1;  
  
 *solveBySLAY*(xs2, ys2,n+1, cs, as, bs, ds);  
  
 **for**(**int** i = 0; i < xs2.size() - 1; i++){  
 *countSpline*(i, 0.5, xs2, cs, as, bs, ds);  
 }  
  
 *makeDiffSplineTable*(ys, xs2, cs, as, bs, ds);  
 }  
  
 **private static void** countSpline(**int** myIter, **double** step, ArrayList<Double> xs, ArrayList<Double> cs, ArrayList<Double> as, ArrayList<Double> bs,ArrayList<Double> ds){  
 *printSplineGraf*(xs.get(myIter), as.get(myIter), bs.get(myIter), cs.get(myIter), ds.get(myIter), step);  
 }  
  
 **private static void** makeDiffSpline( ArrayList<Double> ys, ArrayList<Double> xs, ArrayList<Double> cs, ArrayList<Double> as, ArrayList<Double> bs,ArrayList<Double> ds){  
 **for** (**int** i = 0; i < xs.size() - 1; i++){  
 **double** ySpline = 0;  
 **double** xDiff = xs.get(i) - xs.get(i);  
 ySpline += as.get(i);  
 ySpline += bs.get(i) \* xDiff;  
 ySpline += cs.get(i) \* xDiff \* xDiff;  
 ySpline += ds.get(i) \* xDiff \* xDiff \* xDiff;  
 System.*out*.println("Diff aprox and spline interpol from aprox: " + Math.*abs*(ys.get(i) - ySpline));  
 }  
 }  
  
 **private static void** makeDiffSplineTable(ArrayList<Double> ys, ArrayList<Double> xs, ArrayList<Double> cs, ArrayList<Double> as, ArrayList<Double> bs,ArrayList<Double> ds){  
 **for** (**int** i = 0; i < xs.size() - 1; i++){  
 **double** ySpline = 0;  
 **double** xDiff = xs.get(i) - xs.get(i);  
 ySpline += as.get(i);  
 ySpline += bs.get(i) \* xDiff;  
 ySpline += cs.get(i) \* xDiff \* xDiff;  
 ySpline += ds.get(i) \* xDiff \* xDiff \* xDiff;  
 System.*out*.println("Diff aprox and spline interpol from table: " + Math.*abs*(ys.get(i) - ySpline));  
 }  
 }  
  
 **private static void** printSplineGraf(Double x, Double a, Double b, Double c, Double d, **double** step){  
 System.*out*.printf("a = %f, b = %f, c = %f, d = %f ", a, b, c, d);  
 System.*out*.println(String.*format*("y = %.3f + %.3f(x - %f) + %.3f(x - %f)^2 + %.3f(x - %f)^3 {%f <= x <= %f}", a, b, x, c, x, d, x, x, x+step).replaceAll(",","."));  
 }  
  
 **public static void** main(String[] args) {  
 *checkRightSpline*();  
 System.*out*.println();  
 *solveSpline*();  
 }  
}

**7. Вывод**

В ходе работы было изучено применение сплайн-интерполяции для заданной функции, вычисление значений сплайна 3-го порядка и абсолютной погрешности. Из результатов тестирования можно сделать вывод, что значения функции и сплайна совпадают в узлах интерполяции, а в новых узлах погрешность составлет не более 1 порядка.