МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э. БАУМАНА
Факультет информатики и систем управления
Кафедра теоретической информатики и компьютерных технологий

Лабораторная работа №3
по курсу «Численные методы»

«Построение для таблично-заданной функции кубического сплайна, сплайна Акимы, Б-сплайна»

Выполнил:
студент группы ИУ9-62
Беляев А. В.

Проверила:
Домрачева А. Б.

Москва 2016

Содержание

1 Постановка задачи **3**

2 Необходимые теоретические сведения **4**

2.1 Кубические сплайны 4

2.2 Б - сплайны 4

2.3 Сплайны Акимы 6

3 Текст программы **7**

4 Результаты **10**

5 Выводы **12**

1 Постановка задачи

Пусть имеется таблично-заданная функция $y_i = g(x_i), i = 1..n$, а также функция $f(x) = ae^{bx} = 2.477520 \times e^{0.232206 \times x}$, полученная из её аналитического представления в рамках Лабораторной работы №2.

Таблица 1: $y_i = g(x_i)$

X	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
Y	2.78	3.13	3.51	3.94	4.43	4.97	5.58	6.27	7.04	7.91

Необходимо:

1) Построить кубический сплайн $S_i(x)$ для табличного представления функции y(x);

2) Найти значения сплайн-функции в узлах интерполяции x_i, вычислить абсолютную погрешность;

3) Нати значения сплайн-функции в точках между узлами интерполяции, вычислить абсолютную погрешность;

4) Выполнить пункты 1 - 3 для для Б-сплайна;

5) Выполнить пункт 4 для сплайна Акимы;

2 Необходимые теоретические сведения

2.1 Кубические сплайны

Кубическим сплайном называется функция $S(x)$, которая:

- на каждом отрезке $[x_{i-1}, x_i]$ является многочленом степени не выше третьей
- дважды непрерывно дифференцируема на всем отрезке $[a, b]$
- в точках x_i выполняется равенство $S(x_i) = f(x_i)$

Функция $S_i(x)$ представляется в следующем виде:

$$S_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3, x \in [x_{i-1}, x_i]$$

Введем следующие обозначения: $h = \frac{x_n - x_0}{n}, y_i = y(x_i)$. Сплайн-функция удовлетворяет следующим условиям:

$$S_i(x_i) = y_i;$$
$$S_i'(x_i) = S_{i+1}'(x_i), i = 1, 2, ...n - 1;$$
$$S_i''(x_i) = S_{i+1}''(x_i), i = 1, 2, ...n - 1;$$
$$S_1'(x_0) = S_n'(x_n) = 0.$$

Из этих условий можно получить систему уравнений относительно коэффициентов c_i:

$$\begin{cases} c_i = 0, i = 0, n \\ c_i + 4c_{i+1} + c_{i+2} = \frac{3}{h^2}(y_{i+1} - 2y_i + y_{i-1}), i = 1, ..., n - 1 \end{cases}$$

Эту систему можно решить методом прогонки, а её решение использовать для получения остальных коэффициентов сплайна по следующим формулам:

$$a_i = y_i;$$
$$b_i = \frac{y_{i+1} - y_i}{h} - \frac{h}{3}(c_{i+1} + 2c_i);$$
$$d_i = \frac{c_{i+1} - c_i}{3h};$$

2.2 Б - сплайны

Любая сплайн-функция заданной степени, гладкости и области определения может быть представлена, как линейная комбинация Б-сплайнов той же степени и гладкости на той же области определения. Так, кубический сплайн $S(x)$ на отрезке $[x_0, x_n]$ можно записать, как линейную комбинацию функций B_k:

$$S(x) = \sum_{k=-1}^{n+1} a_k B_k(x)$$

Определим $B_k(x) = B_0(x-kh)$. Для интерполяции, проводимой на отрезке $[x_0, x_n]$, базовая функция B_0 определяется следующим образом:

$$B_0(x) = \begin{cases} 0 & x \le x_0 - 2h \\ \frac{1}{6}(2h + (x - x_0))^3 & x_0 - 2h < x \le x_0 - h \\ \frac{2h^3}{3} - \frac{1}{2}(x - x_0)^2(2h + (x - x0)) & x_0 - h < x \le x_0 \\ \frac{2h^3}{3} - \frac{1}{2}(x - x_0)^2(2h - (x - x0)) & x_0 < x \le x_0 + h \\ \frac{1}{6}(2h - (x - x_0))^3 & x_0 + h < x \le x_0 + 2h \\ 0 & x_0 + 2h < x \end{cases}$$

где $h = \frac{x_n - x_0}{n}$ – это расстояние между узлами.

Значение функции $S(x_k)$ узловой точки принимает ненулевое значение только для $B_{k-1}, B_k, B_{k+1}, B_{k+2}$:

$$S(x_k) = a_{k-1}B_{k-1}(x_k) + a_kB_k(x_k) + a_{k+1}B_{k+1}(x_k) + a_{k+2}B_{k+2}(x_k) = f(x_k)$$

Необходимо определить коээфициенты a_i. Для этого из определений $B_k(x_k)$ и $B_0(x_0)$:

$$B_{k-1}(x_k) = B_0(x_0 + h) = \frac{h^3}{6}$$

$$B_k(x_k) = B_0(x_0) = \frac{2h^3}{3}$$

$$B_{k+1}(x_k) = B_0(x_0 - h) = \frac{h^3}{6}$$

$$B_{k+2}(x_k) = B_0(x_0 - 2h) = 0$$

подставим значения $B_{k-1}, B_k, B_{k+1}, B_{k+2}$ в функцию $S(x_k)$ и получим рекуррентное соотношение для коэффициентов a_i:

$$a_{k-1} + 4a_k + a_{k+1} = \frac{6}{h^3}f(x_k), k = 0, ..., n$$

Рассмотрим $S_3''(x)$:

$$S_3''(x) = \sum_{k=-1}^{n+1} a_kB_k''(x)$$

Дважды дифференцируя $B - 0$, получим:

$$B_0''(x) = \begin{cases} 0 & x \le x_0 - 2h \\ 2h + (x - x_0) & x_0 - 2h < x \le x_0 - h \\ -2h - 3(x - x0)) & x_0 - h < x \le x_0 \\ -2h + 3(x - x_0) & x_0 < x \le x_0 + h \\ 2h - (x - x0) & x_0 + h < x \le x_0 + 2h \\ 0 & x_0 + 2h < x \end{cases}$$

Тогда, приравняв нулю $S_3''(x_0)$, прийдем к равенству $a_{-1}-2a_0+a_1=0$. Определим рекуррентное соотношение для $k=0$:

$$a_{-1}+4a_0+a_{k+1}=\frac{6}{h^3}f(x_k)$$

Вычтем из этого соотношения $a_{-1}-2a_0+a_1=0$ и получим значения для a_0:

$$a_0=\frac{1}{h^3}f(x_0)$$

По аналогии находим значение a_n:

$$a_n=\frac{1}{h^3}f(x_n)$$

Значения всех остальных коэффициентов a_i вычисляем из системы вида:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 1 & 4 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 4 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \dots & \dots & \vdots \\ 0 & \dots & 0 & 1 & 4 & 1 & 0 \\ 0 & \dots & 0 & 0 & 1 & 4 & 1 \\ 0 & \dots & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-2} \\ a_{n-1} \\ a_n \end{pmatrix} = \frac{1}{h^3} \begin{pmatrix} f(x_0) \\ 6f(x_1) \\ 6f(x_2) \\ \vdots \\ 6f(x_{n-2}) \\ 6f(x_{n-1}) \\ f(x_n) \end{pmatrix}$$

и дополнительно определим a_{-1} и a_{n+1}:

$$a_{-1}=2a_0-a_1$$

$$a_{n+1}=2a_n-a_{n-1}$$

2.3 Сплайны Акимы

Сплайн Акимы – это особый вид сплайна, предложенный Хироши Акимой, устойчивый к выбросам. Недостатком кубических сплайнов является то, что они склонны осциллировать в окрестностях точки, существенно отличающейся от своих соседей.

Свойством сплайна Акимы является его локальность – значения функции на отрезке $[x_i, x_{i+1}]$ зависят только от значений $f_{i-2}, f_{i-1}, f_i, f_{i+1}, f_{i+2}$, вследствие чего на отрезках, граничащих с выбросом, практически отсутствуют признаки осцилляции.

Сплайн-функцию можно представить в следующем виде:

$$S(x)=c_1+c_2(x-x_i)+c_3(x-x_i)^2+c_4(x-x_i)^3$$

где $x\in[x_i,x_{i+1}], i\in[1,k]$, а коэффициенты c_i вычисляются подобным образом:

$$c_1=y_i$$

$$c_2=m_i$$

$$c_3 = \frac{3(m_i - t_i) + t_{i+1} - t_i}{h_i}$$

$$c_4 = \frac{\frac{t_{i+1}-t_i}{h_i} - 2(m_i - t_i)}{{h_i}^2}$$

где m является первой производной узла и вычисялется следующим образом:

$$m_i = \frac{f_{i+1} - f_i}{x_{i+1} - x_i}, i \in [0; n)$$

и где $f_i = f(x_i)$, $h_i = x_{i+1} - x_i$ и $m_i = \frac{y_{i+1}-y_i}{h_i}$.
Значения на концах отрезка функции вычисляются отдельно согласно формулам:

$$m_{-2} = 3m_0 - 2m_1$$

$$m_{-1} = 2m_0 - m_1$$

$$m_{n+1} = 2m_n - m_{n-1}$$

$$m_{n+2} = 3m_n - 2m_{n-1}$$

3 Текст программы

Для написания программы был использован язык C++.

```
#define xtype float
#define N 10

xtype _a = 2.477520;
xtype _b = 0.232206;

xtype xs[] = {0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5};
xtype ys[] = {2.78, 3.13, 3.51, 3.94, 4.43, 4.97, 5.58, 6.27, 7.04, 7.91, 8.89};
xtype xsNew[] = {0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 2.00, 2.25, 2.50, 2.75, 3.00, 3.25, 3.50, 3.75, 4.00, 4.25, 4.50, 4
xtype ysNew[] = {2.78, 2.95, 3.13, 3.31, 3.51, 3.72, 3.94, 4.18, 4.43, 4.69, 4.97, 5.27, 5.58, 5.92, 6.27, 6.65, 7.04, 7
xtype xsNew1[2*N+1];
xtype ysNew1[2*N+1];
xtype f(xtype x) { return _a * exp(_b*x); }

xtype h = xs[1] - xs[0];

typedef struct spline {
    xtype a[N];
    xtype b[N];
    xtype c[N];
    xtype d[N];
} _spline;

typedef struct bispline {
    xtype x[N+3];
} _bispline;

typedef struct akame {
    xtype a[N+4];
    xtype b[N+4];
    xtype c[N+4];
    xtype d[N+4];
} _akame;

spline populateSpline(int n) {

    xtype ak[n-3], bk[n-2], ck[n-3], dk[n-2], ek[n];
    xtype a[n], b[n], c[n], d[n];

    xtype h = xs[1] - xs[0];

    for (int i = 0; i < n - 2; i++) {

        if (i < n-3) ak[i] = ck[i] = 1;

        bk[i] = 4;
        dk[i] = 3 / (h * h) * (ys[i+2] - 2*ys[i+1] + ys[i]);
    }
```

```
    solveMatrix(n-2, ak, bk, ck, dk, ek);

    for (int i = 1; i < n-1; i++) c[i] = ek[i-1];
    c[0] = c[n-1] = 0;

    for (int i = 0; i < n-1; i++) {
        a[i] = ys[i];
        b[i] = (ys[i+1] - ys[i]) / h - (h/3) * (c[i+1] + 2*c[i]);
        d[i] = (c[i+1] - c[i]) / (3*h);
    }


    spline spZ;

    for (int i = 0; i < n; i++) {
        spZ.a[i] = a[i];
        spZ.b[i] = b[i];
        spZ.c[i] = c[i];
        spZ.d[i] = d[i];
    }
    return spZ;
}

xtype countSplineAt(_spline spline, xtype x) {
    //S_j[x] = a_j + b_j(x-xj) + c_j(x-xj)^2 + d_j(x-xj)^3
    int i;
    for (i = 0; i < N; ++i)
        if (x >= xs[i] && x <= xs[i+1])
            break;

    return  spline.a[i] +
            spline.b[i]*(x - xs[i]) +
            spline.c[i]*(x - xs[i])*(x - xs[i]) +
            spline.d[i]*(x - xs[i])*(x - xs[i])*(x - xs[i]);
}


bispline populateBiSpline(int len) {

    int size = len;
    int n = size - 1;
    xtype a[size], b[size], c[size], d[size], e[size+2];

    for (int i = 1; i < n; i++) {
        a[i] = 1;
        b[i] = 4;
        c[i] = 1;
        d[i] = 6*ys[i];
    }

    a[0] = 0;           a[n] = 0;
    b[0] = 1;           b[n] = 1;
    c[0] = 0;           c[n] = 0;
    d[0] = ys[0];       d[n] = ys[n-1];

    for (int i = 0; i < n; i++)  d[i] /= (h*h*h);

    solveMatrixNew(len, a, b, c, d, e);


    _bispline bsp;

    for (int i = 0; i < size+2; i++)
        bsp.x[i] = e[i];

    return bsp;
}

xtype baseX(int k, xtype x, xtype kh) {
    if (0 == k) {

        xtype xz = xs[0];

        if (xz + 2*h <= x) return 0;
        if (xz + h <= x && x < xz + 2*h) return pow(2*h + xz - kh - (x - kh), 3)/6;
        if (xz <= x && x < xz + h)
            return 2*pow(h, 3)/3 - pow(-xz + kh + (x - kh), 2)*(2*h + xz - kh - (x - kh))/2;
        if (xz - h <= x && x < xz)
            return 2*pow(h, 3)/3 - pow(-xz + kh + (x - kh), 2)*(2*h - xz + kh + (x - kh))/2;
        if (xz - 2*h <= x && x < xz - h) return pow(2*h - xz + kh + (x - kh), 3)/6;
        if (x < xz - 2*h) return 0;

    } else {
        return baseX(0, x - k*h, -k*h);
    }

}

xtype countBiSplineAt(_bispline bsp, xtype val) {
    int i;
    for (i = 0; i < N+2; ++i)
        if (xs[i] <= val && val <= xs[i+1])
            break;
```

```
    return  bsp.x[i + 0] * baseX(i - 1, val, 0) +
            bsp.x[i + 1] * baseX(i + 0, val, 0) +
            bsp.x[i + 2] * baseX(i + 1, val, 0) +
            bsp.x[i + 3] * baseX(i + 2, val, 0);
}


akame populateAkame(int n) {
    int i;
    xtype m[n + 4];

    //-2;-1;    0.. n-1    ;n;n+1
    for (i = 2; i < n + 2; i++)
        m[i] = (ys[i+1-2] - ys[i-2]) / (xs[i+1-2] - xs[i-2]);

    m[0] = 3*m[2] - 2*m[3]; //m[-2]=m[0]-m[1]
    m[1] = 2*m[2] - m[3];
    m[n+2] = 2*m[n+1] - 2*m[n];
    m[n+3] = 3*m[n];

    xtype ne, alpha_i, h_i;
    xtype t_l[n], t_r[n];

    for (i = 2; i < n + 2; i++) {
        ne = fabs(m[i+1] - m[i]) + fabs(m[i-1] - m[i-2]);

        if (ne > 0) {
            alpha_i = fabs(m[i-1] - m[i-2]) / ne;
            t_l[i] = m[i-1] + alpha_i*(m[i] - m[i-1]);
            t_r[i] = t_l[i];

        } else {
            t_l[i] = m[i-1];
            t_r[i] = m[i];
        }
    }


    akame ga_spline;

    for (i = 2; i < n+2; i++) {
        h_i = xs[i+1-2] - xs[i-2];

        ga_spline.a[i] = ys[i-2];
        ga_spline.b[i] = t_r[i];
        ga_spline.c[i] = (3*m[i] - 2*t_r[i] - t_l[i+1]) / h_i;
        ga_spline.d[i] = (t_r[i] + t_l[i+1] - 2*m[i]) / (h_i*h_i);
    }
    return ga_spline;
}

xtype countAkameAt(akame ak, xtype x, int i) {

    for (i = 0; i < N; ++i)
        if (x >= xs[i] && x <= xs[i+1])
            break;

    return  ak.a[i+2] +
            ak.b[i+2]*(x - xs[i]) +
            ak.c[i+2]*(x - xs[i])*(x - xs[i]) +
            ak.d[i+2]*(x - xs[i])*(x - xs[i])*(x - xs[i]);
}

int main()
{
    for (int i = 0; i <= 2*N; i++)  xsNew1[i] = (i % 2) == 0 ? xsNew[i] : xsNew[i-1] + 0.15;
    for (int i = 0; i <= 2*N; i++) ysNew1[i] = (i % 2) == 0 ? ysNew[i] : f(xsNew1[i]);

    printf("populating Spline\n");
    spline sp = populateSpline(N+1);

    printf("populating Bispline\n");
    bispline bs = populateBiSpline(N+2);

    printf("populating Akame ga Spline\n");
    akame as = populateAkame(N);

    for ( int i = 0; i <= 2*N; i++) {
        xtype sVal = countSplineAt(sp, xsNew[i]);
        xtype bsVal = countBiSplineAt(bs, xsNew[i]);
        xtype asVal = countAkameAt(as, xsNew[i], i);

        printf("x[%d]=%.2f y=%.5f \ts=%.5f d=%.5f \tbi=%.5f d=%.5f \ta=%.5f d=%.5f\n",
               i, xsNew[i], ysNew[i],
               sVal, fabs(ysNew[i] - sVal),
               bsVal, fabs(ysNew[i] - bsVal),
               asVal, fabs(ysNew[i] - asVal));
    }
    return 0;
}
```

4 Результаты

Обозначим значения кубических сплайнов, как CubicSpline или C. Значения Б-сплайнов, как BasisSpline или B. Значения сплайнов Акимы, как AkimaSpline или A. Проведем тестирование. В результате работы программы были получены следующие результаты в узлах интерполяции:

Таблица 2: Значения в узлах интерполяции

x	f(x)	CubicSpline(x)	BasisSpline(x)	AkimaSpline(x)
0.50	2.780000	2.780000	2.780000	2.780000
1.00	3.130000	3.130000	3.130000	3.130000
1.50	3.510000	3.510000	3.510000	3.510000
2.00	3.940000	3.940000	3.940000	3.940000
2.50	4.430000	4.430000	4.430000	4.430000
3.00	4.970000	4.970000	4.970000	4.970000
3.50	5.580000	5.580000	5.580000	5.580000
4.00	6.270000	6.270000	6.270000	6.270000
4.50	7.040000	7.040000	7.040000	7.040000
5.00	7.910000	7.910000	7.910000	7.910000

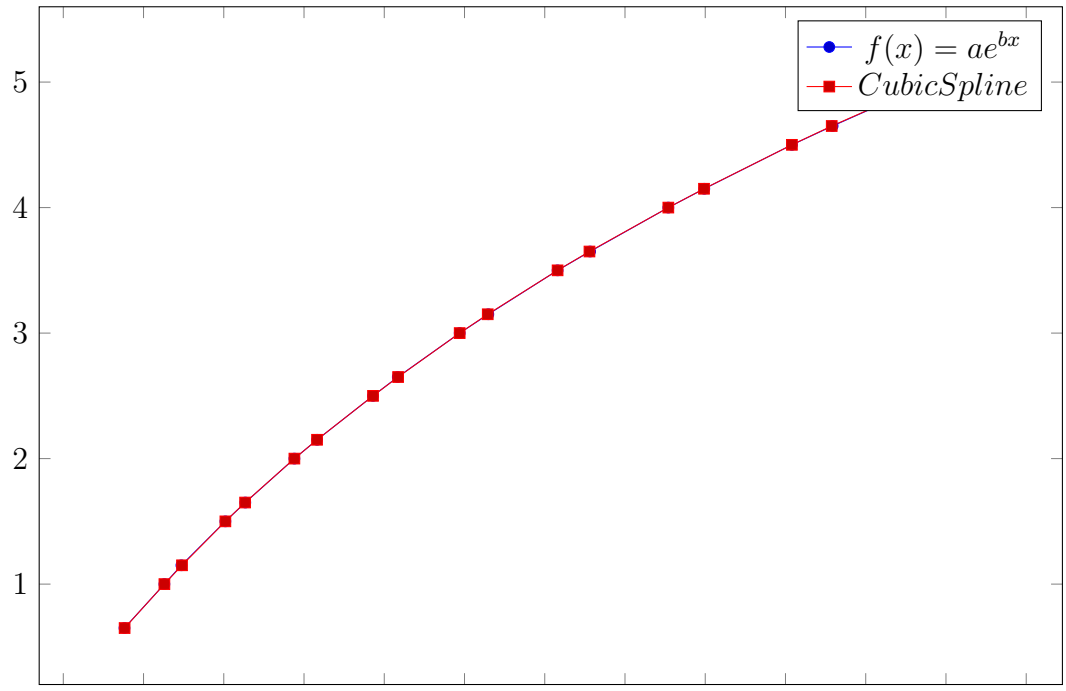
Как следует из результатов тестирования, значения всех сплайн-функций полностью совпадают между собой и с “ожидаемым” значением в узлах интерполяции. Это говорит о корректности построения сплайн-функций.

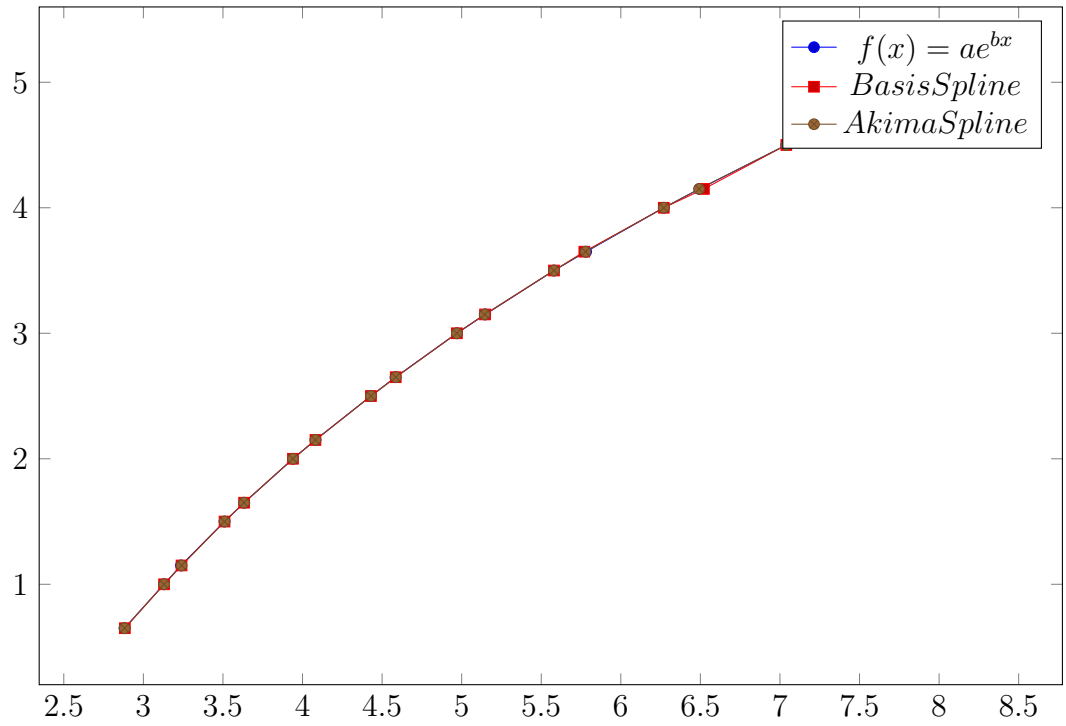
Теперь вычислим значения между узлами интерполяции в точках $x_{new} = x_i + 0.15$:

Таблица 3: Значения между узлами интерполяции

x	f(x)	C(x)	$\lvert f(x) - C(x) \rvert$	B(x)	$\lvert f(x) - B(x) \rvert$	A(x)	$\lvert f(x) - A(x) \rvert$
0.65	2.881157	2.882504	0.001347	2.883517	0.002360	2.882086	0.000929
1.15	3.235861	3.239793	0.003932	3.239804	0.003943	3.240194	0.004333
1.65	3.634233	3.632949	0.001283	3.632908	0.001325	3.632210	0.002023
2.15	4.081649	4.081181	0.000468	4.081334	0.000325	4.081136	0.000513
2.65	4.584146	4.586258	0.002112	4.585686	0.001543	4.586346	0.002200
3.15	5.148508	5.144826	0.003682	5.146964	0.001354	5.144316	0.004192
3.65	5.782349	5.778608	0.003741	5.769631	0.012718	5.778488	0.003861
4.15	6.494223	6.492341	0.001882	6.467652	0.026571	6.491814	0.002409
4.65	7.293736	7.288169	0.005568	7.246946	0.046790	7.289189	0.004547
5.15	8.191680	8.194182	0.002502	8.108945	0.082735	8.14436	0.047320

Как видно из таблицы результатов выше, Б-сплайны и сплайны Акимы дают погрешность примерно одного порядка, что на порядок меньше, чем погрешность кубического сплайна.





Выше приведены графики для наглядного сравнения результатов. Заметно отсутствие “выбросов” значений (нужен больший масштаб чтобы полностью оценить различия результатов) а также заметно “перекрытие” графика исходной функции графиками Б-сплайна и сплайна Акимы, что наглядно демонстрирует значения их погрешностей по таблице.

5 Выводы

В ходе работы были реализованы построения кубических сплайнов, Б-сплайнов и сплайнов Акимы по таблично-заданной функции. В вычислениях присутствует методологическая погрешность. Было вычислено ее абсолютное значение для данных методов Так, было выявлено, что погрешность кубического сплайна не превышает 0.006, погрешность Б-сплайна выше, но не превосходит 0.083 и погрешность сплайна Акимы не выше 0.048. Отсюда следует, что наилучший результат дает кубический сплайн. Результаты Б-сплайна и сплайна Акимы примерно одинаковы и оба уступают кубическом сплайну. Отсюда следует вывод, что кубический сплайн больше подходит для решения реальных практических задач, нежели остальные методы.