Оглавление

[Задание 3](#_Toc536108557)

[Введение 4](#_Toc536108558)

[Теоретическая часть. 4](#_Toc536108559)

[Общие принципы 4](#_Toc536108560)

[Билинейная интерполяция 4](#_Toc536108561)

[Бикубическая интерполяция 5](#_Toc536108562)

[Сравнение методов интерполяции 7](#_Toc536108563)

[Выводы 10](#_Toc536108564)

[Приложение 1. 10](#_Toc536108565)

# Задание

Реализовать алгоритмы бикубической и билинейной интерполяции. Провести сравнение по точности и скорости на различных величинах шага интерполяционной сетки.

# Введение

Цель практикума: изучить и реализовать методы билинейной и бикубической интерполяции. Сравнить точность и скорость выполнения на интерполяционных сетках с различным шагом.

Работа выполняется в программной среде Matlab.

# Теоретическая часть.

## Общие принципы

Интерполя́ция, интерполи́рование (от лат. inter–polis — «разглаженный, подновлённый, обновлённый; преобразованный») — в вычислительной математике способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений. Термин «интерполяция» впервые употребил Джон Валлис в своём трактате «Арифметика бесконечных» (1656).

Основная идея всех методов – подбор некой функции в аналитическом виде, с достаточной точностью близкой к точкам имеющегося разбиения (в продвинутых алгоритмах также проверяется точность первой либо второй производной) на некотором участке, после чего легко вычисляются её значения на необходимой расширенной сетке в пределах рассматриваемого участка.

Соответственно, для одномерных случаев подбираются функции одного переменного, для двумерных – двух и так далее.

Интерполяция нужна, когда мы имеем ограниченный дискретный набор данных, и необходимо получить другой набор с меньшим шагом (увеличить размер изображения, получить значение функции в промежуточных точках и т.д.). Методов существует много, но рассмотрим два наиболее простых – билинейной и бикубической интерполяции.

## Билинейная интерполяция

Билинейная интерполяция — обобщение линейной интерполяции функций одной переменной для функций двух переменных.

Обобщение основано на применении обычной линейной интерполяции сначала в направлении одной из координат, а затем в перпендикулярном направлении.

Полученная функция билинейной интерполяции интерполирует значения исходной функции в произвольном прямоугольнике по четырём её значениям в вершинах прямоугольника и экстраполирует функцию на всю остальную поверхность.

Основной принцип: интерполируем прямой сначала по одной координате, потом по другой.

Из названия метода следует, что интерполяционная функция здесь – билинейная вида

Пусть мы интерполируем значение функции в точке P = (x,y). Значения функции в четырёх окружающих точку P точках исходной сетки , известны.

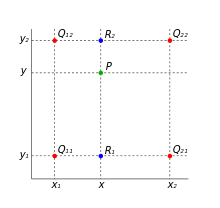


Рис. 1. Положение интерполируемой и опорных точек

Сперва интерполируется значение опорных точек

Потом проводится линейная интерполяция между опорными точками:

Это и есть интерполируемое значение функции , причём значения интерполирующей функции равны значениям интерполируемой функции в исходных точках , , , .

Очевидная особенность – независимость результата от порядка шагов (сначала можно интерполировать как по оси абсцисс, так и по оси ординат).

## Бикубическая интерполяция

Метод бикубической интерполяции – один из методов двумерной интерполяции повышенной точности. Основная его идея – аппроксимация данных двумерным полиномом третьего порядка.

Рассматривается набор из 16 точек (4х4), расположенных с единичным шагом на отрезках [-1; 2] по каждой из двух координат, при этом интерполяция происходит на участке [0;1] (т.е. этот метод не позволяет интерполировать весь набор данных). Дополнительные точки необходимы для нахождения производных в главных узлах простейшим методом. Это, помимо прочего, позволяет получить гладкую границу между соседними полиномами.

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

Представим нам полином в виде:

Из коэффициентов составляется матрица α=.

Представим матрицу α в виде столбца:

*,*

и введём столбец х:

*,*

где – производная по х функции в точке (1,0), – смешанная производная функции в данной точке (сначала, например, вычисляются производные по х точек выше и ниже данной, после чего считается производная по у. Т.е. нужно знать производные по х всех точек из серой зоны(см. рисунок)). Все производные в нашем случае считаются методом центральных конечных разностей как дающим неплохую точность на небольшом количестве вспомогательных точек.

Соотнеся элементы вектора х с интерполяционным многочленом, получим 16 линейных уравнений для вычисления коэффициентов (x={0,1}, y={0,1}):

Приведем эту систему к матричному виду:

A=х.

Отсюда можно найти необходимые коэффициенты, решив СЛАУ (например, методом обратной матрицы), и с их помощью легко вычислять все значения функции на интерполяционной сетке в пределах квадрата [0,1] с любым шагом.

Точно так же вычисляется матрица *{α}* для каждого квадрата исходной сетки.

Так как метод вычисляет производные, по границам исходного разбиения остаётся граница шириной в 1 шаг, где интерполяция не происходит.

Метод обеспечивает непрерывность самой функции и её второй производной на границах смежных квадратов.

# Сравнение методов интерполяции

Для сравнения были заготовлены 7 поверхностей: 6 сложных и одна простая (параболоид).

Для метода бикубической интерполяции сделано две модификации решения СЛАУ: методом обратной матрицы встроенными средствами Matlab и методом Крамера.

Для замеров времени выполнения делалось по 100 итераций каждого метода для каждой поверхности.

Методы сравнивались на двух сетках с разным шагом точек: с исходным разбиением интервала на Na1=5 точек и на Na2=30. Количество итоговых точек Nb одинаково и равно 150.

Для построения графиков поверхностей в целях наилучшей визуальной различимости работы методов использовалась сетка с Na=5 и Nb=100, а также режим shading-а “flat”.

Структура графиков поверхностей: верхний ряд – поверхность на исходной сетке, нижний – результат интерполяции. Для каждой поверхности приведена своя группа графиков.

При прорисовке графиков бикубического метода поверхность обрезана до той области, которая в итоге интерполируется (по 1 шагу исходной сетки с каждой стороны). За счёт изменения границы масштаб осей может различаться для двух методов.

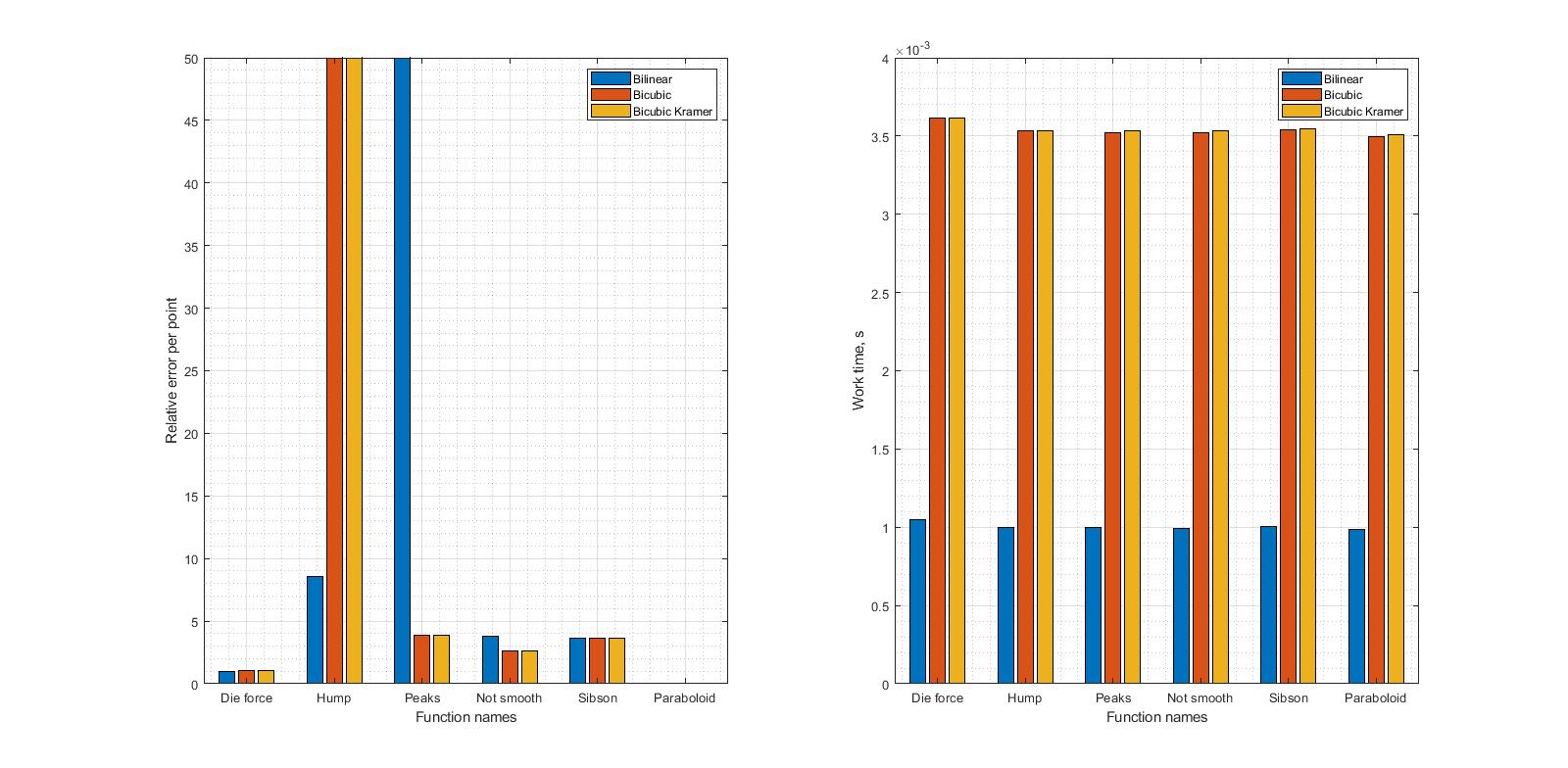
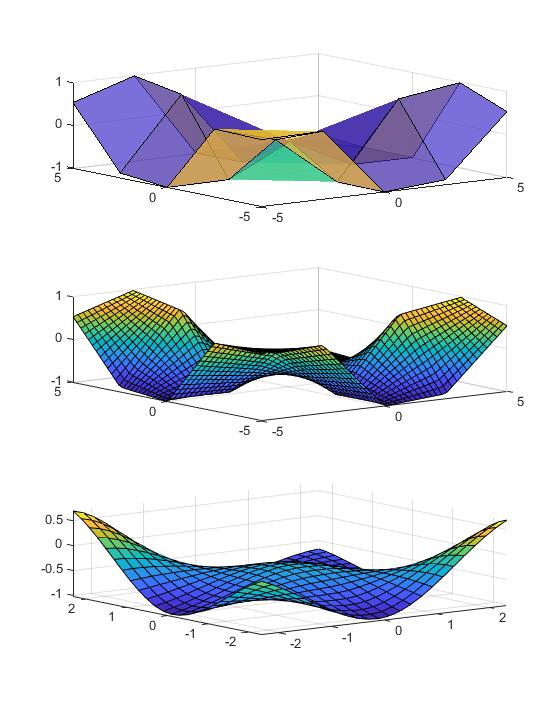
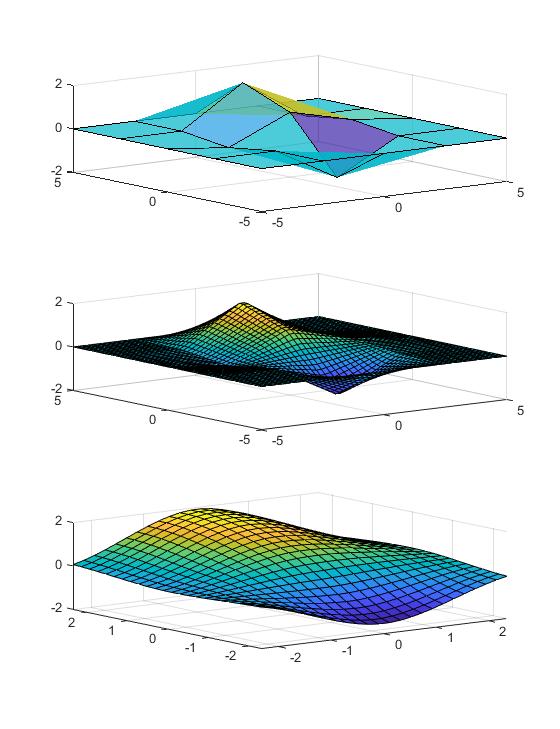


Рис.2. Средняя ошибка на точку и среднее время выполнения для сетки с Na=5



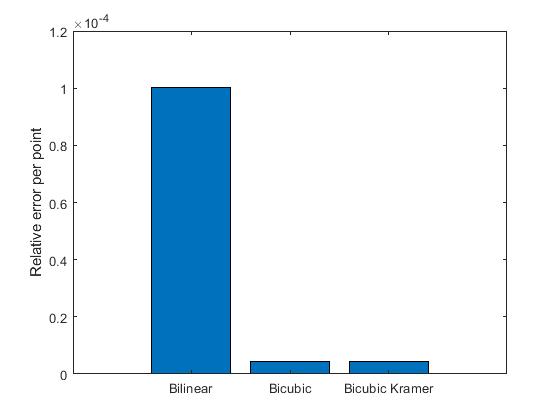
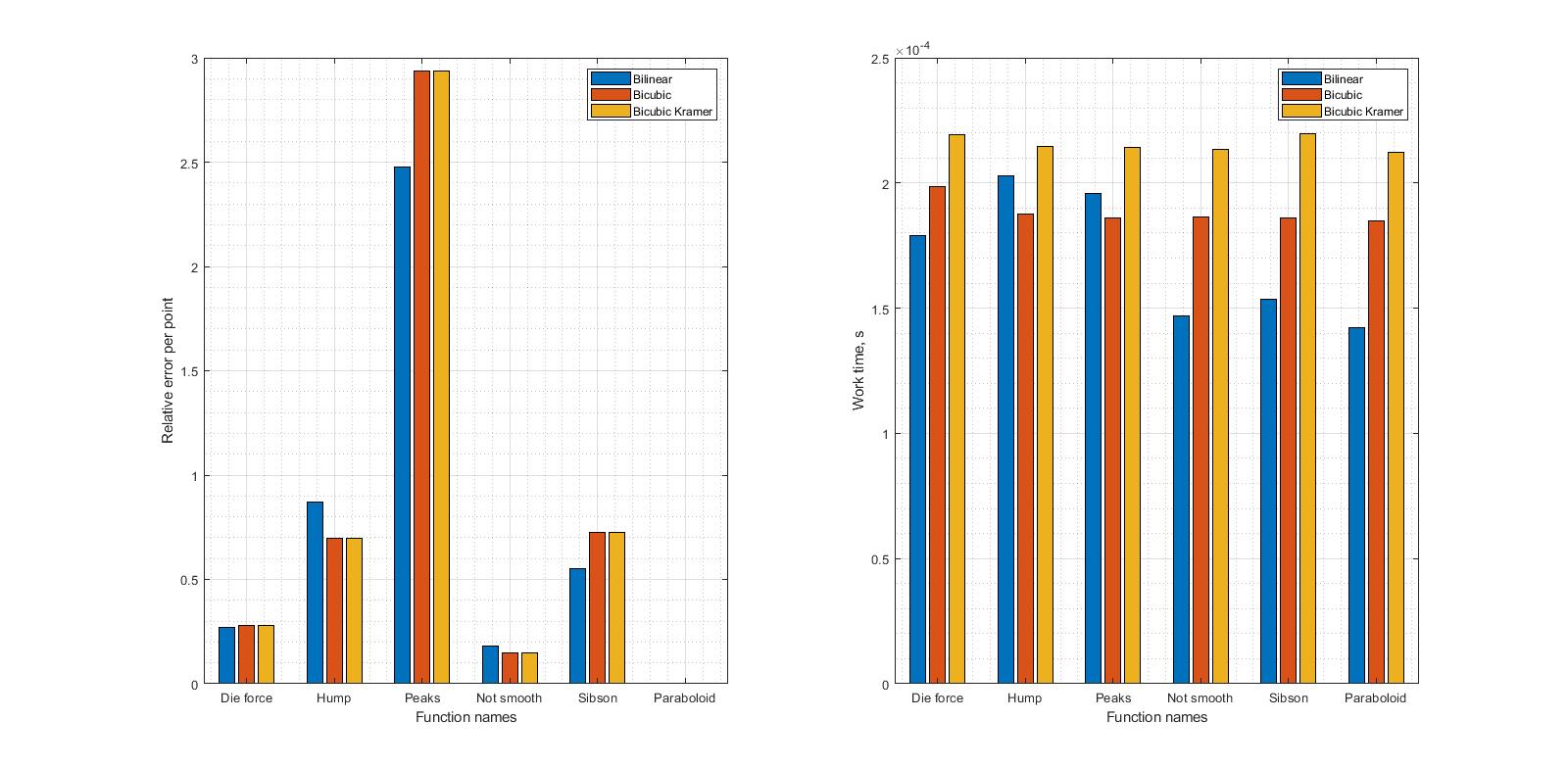
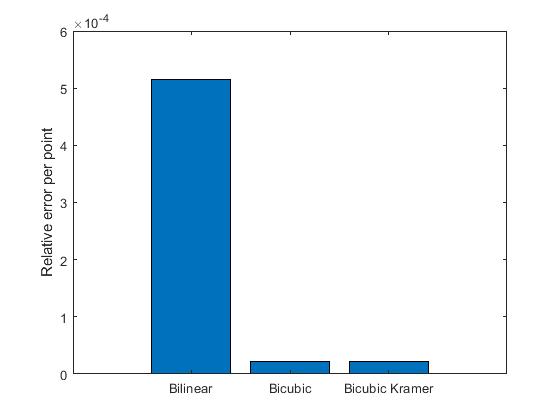
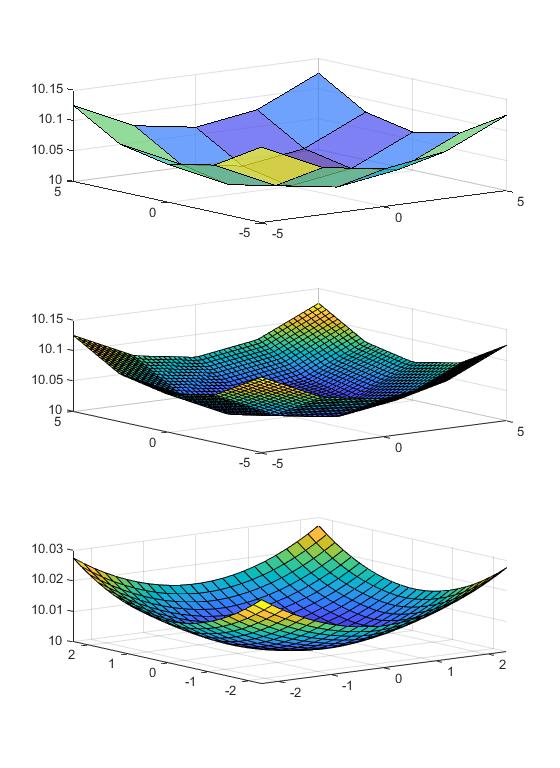
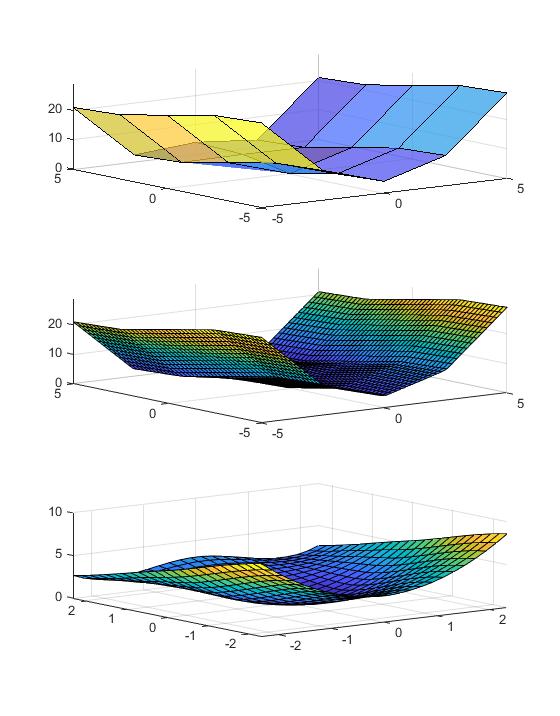
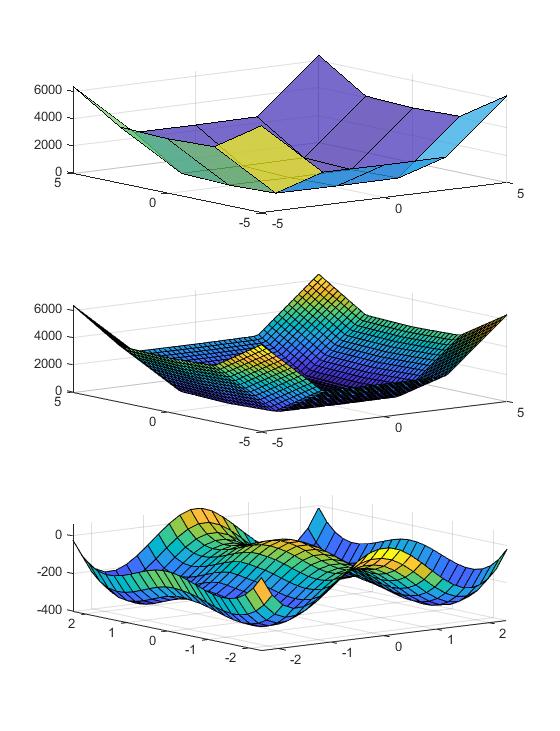
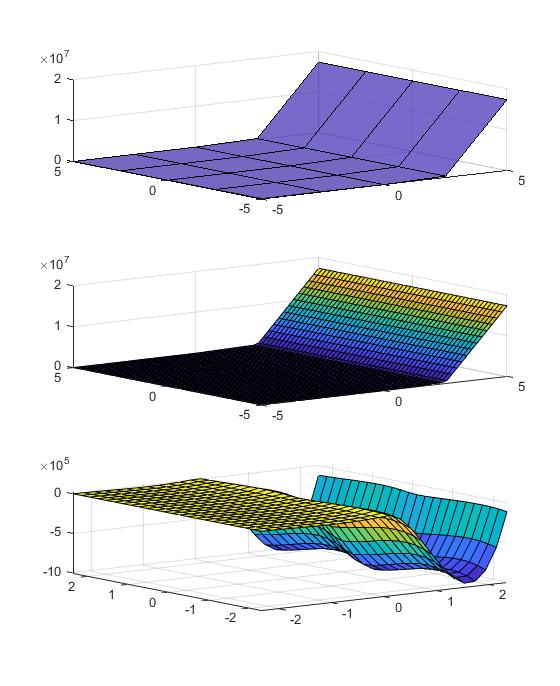


Рис.5. Средняя ошибка на точку для поверхности «Параболоид» для сетки с Na=30

Рис.4. Средняя ошибка на точку и среднее время выполнения для сетки с Na=30

Рис.3. Средняя ошибка на точку для поверхности «Параболоид» для сетки с Na=5





# Выводы

Бикубический метод более сложен вычислительно и дольше работает. Но в большом количестве случаев он дает лучший результат (на простых поверхностях – на порядок лучше), чем билинейный. Но при уменьшении числа исходных точек при постоянном количестве результирующих алгоритм билинейной интерполяции по времени выполнения приближается к бикубической. Точность также постепенно сравнивается.

В системах реального времени при ожидании сложных поверхностей лучше применять билинейный метод. Если же резких скачков не ожидается, то бикубический, несомненно, выигрывает в точности как вообще, так и относительно затраченного времени.

Можно заметить, что области интерполяции в этих методах независимы друг от друга. Таким образом, алгоритмы могут эффективно выполняться в многопоточных системах.

# Приложение 1.

Листинг 1. Билинейный метод

function [F, C] = my\_bilinear(F0,C0,C)

p = size(C0);

X0 = C0{1};

Y0 = C0{2};

X=C{1};

Y=C{2};

F = zeros(size(X));

nX = 2;

nY = 2;

for i=1:1:size(X,1) %along Y

if Y(i,1) > Y0(nY,1)

nY = nY + 1;

end

for j=1:1:size(X,2) %along X

if X(1,j) > X0(1,nX)

nX = nX + 1;

end

f1 = (X0(nY,nX)-X(i,j))/(X0(nY,nX)-X0(nY,nX-1))\*F0(nY-1,nX-1);

f1 = f1 + (X(i,j)-X0(nY-1,nX-1))/(X0(nY,nX)-X0(nY,nX-1))\*F0(nY-1,nX);

f2 = (X0(nY,nX)-X(i,j))/(X0(nY,nX)-X0(nY,nX-1))\*F0(nY,nX-1);

f2 = f2 + (X(i,j)-X0(nY,nX-1))/(X0(nY,nX)-X0(nY,nX-1))\*F0(nY,nX);

F(i,j) = (Y0(nY,nX)-Y(i,j))/(Y0(nY,nX)-Y0(nY-1,nX))\*f1;

F(i,j) = F(i,j) + (Y(i,j)-Y0(nY-1,nX))/(Y0(nY,nX)-Y0(nY-1,nX))\*f2;

end

nX=2;

end

end

Листинг 2. Бикубический метод

function [F1, C] = my\_bicubic(F0,C0,C)

%работаем с квадратом 4х4 изначальной сетки

%интерполируема¤ сетка, X0(i,j) - координата по x

tic;

X0 = C0{1};

Y0 = C0{2};

%интерпол¤ционна¤ сетка

X1=C{1};

Y1=C{2};

F1 = nan(size(X1));

A = zeros(16);

%minSize = 6

if size(X0,1) < 4 || size(X0,2) < 4

ME = MException('MyComponent:noEnougthSize', ...

'Size must be at least 6x6!');

throw(ME);

end

p=1; q=1; %индексы интерполируемой точки на интерпол¤ционной сетке

%i,j - координаты верхней левой точки квадрата интерполируемого

%квадрата на интерпол¤ционной сетке

%дл¤ нахождени¤ значени¤ производных нужны боковые границы на начальной

%сетке в 1 шаг.

for i=2:1:size(X0,1)-2

p=1;

for j=2:1:size(Y0,2)-2

tic;

%ищем начальную точку на интерпол¤ционной сетке

while(X1(p,p)<X0(i,j))

p = p + 1;

end

while(Y1(q,q)<Y0(i,j))

q = q + 1;

end

%представл¤ем alpha={a(i,j)} в виде строки [a11,a21,...]

%разбиваем ј на группы по 4

temp=1; %temp - номер в группе

for m=[0,1]

for n=[0,1]

pointX = X0(j+m,i+n); %current Work Point

pointY = Y0(j+m,i+n);

A(:,temp) = P(pointX,pointY);

A(:,temp+4) = px(pointX,pointY);

A(:,temp+8) = py(pointX,pointY);

A(:,temp+12) = pxy(pointX,pointY);

temp = temp+1;

end

end

%X=[f(0,0),f(1,0),f(0,1),f(1,1),fx(0,0), ... ,fy(0,0), ...,fxy(0,0), ...]'

X=zeros(16,1);

X(1:4)=[F0(j,i),F0(j,i+1),F0(j+1,i),F0(j+1,i+1)];

%0-------> x, j, p

%|

%|

%V y, i, q

% Fx(0,0), Fx(1,0), Fx(0,1), Fx(1,1)

% Fy(0,0), Fy(1,0), Fy(0,1), Fy(1,1)

% Fxy(0,0), Fxy(1,0), Fxy(0,1), Fxy(1,1)

%A(y,x)

start=5;

for m=[0,1]

for n=[0,1]

X(start) = (F0(j+m,i+n+1)-F0(j+m,i-1+n))/(X0(j+m,i+n+1)-X0(j+m,i-1+n)); %dF/dx

X(start+4) = (F0(j+m+1,i+n)-F0(j+m-1,i+n))/(Y0(j+m+1,i+n)-Y0(j+m-1,i+n)); %dF/dy

start = start + 1;

end

end

%second derivatives

ders = zeros(1,2\*4);

temp=1;

%производные по ’ дл¤ узлов квадрата, а также дл¤ узлов на 1

%выше и ниже него

%заполн¤ем по строкам

for m=(-1:1:2)

for n=[0,1]

%current point: x=i+n, y=j+m

ders(temp) = (F0(j+m,i+n+1)-F0(j+m,i+n-1))/(X0(j+m,i+n+1)-X0(j+m,i-1+n));

temp = temp+1;

end

end

temp=13;

for m=[0,1]

for n=[0,1]

X(temp) = (ders(2\*(2+m)+n)-ders((1+m)+n))/(Y0(j+m+1,i+n)-Y0(j+m-1,i+n));

temp = temp + 1;

end

end

A = A';

A = A^(-1);

alpha = A\*X;

p0 = p;

q0 = q;

while Y1(q,q)<Y0(i+1,j+1)

while X1(p,p)<X0(i+1,j+1)

value=0;

for k=0:1:3

for l=0:1:3

value = value + alpha(k\*4+l+1)\*(X1(q,p)^k)\*(Y1(q,p)^l);

end

end

F1(p,q) = value;

p = p + 1;

end

p=p0;

q = q + 1;

end

q = q0;

end

end

end

function [x,ok] = my\_kramer(A,b)

[n,m] = size(A);

d=det(A);

x = zeros(n,1);

if (n ~= m || d==0)

ok=false;

else

for i=(1:1:n)

T = A;

T(:,i)=b;

x(i) = det(T)/d;

end

ok=true;

end

end

function t=P(x,y)

t = zeros(1,16);

for i=0:1:3

for j=0:1:3

t(j\*4+i+1) = (x.^i).\*(y.^j);

end

end

end

function t = px(x,y)

t = zeros(1,16);

for i=1:1:3

for j=0:1:3

t(j\*4+i+1) = i\*(x.^(i-1)).\*(y.^j);

end

end

end

function t = py(x,y)

t = zeros(1,16);

for i=0:1:3

for j=1:1:3

t(j\*4+i+1) = (x.^i).\*(y.^(j-1))\*j;

end

end

end

function t = pxy(x,y)

t = zeros(1,16);

for i=1:1:3

for j=1:1:3

t(j\*4+i+1) = i\*(x.^(i-1)).\*(y.^(j-1))\*j;

end

end

end

Ссылки:

1. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. Учебник. И.: Бином. Лаборатория знаний. 2017. 640 с.
2. <https://en.wikipedia.org/wiki/Bicubic_interpolation>
3. https://www.emse.fr/enbis-emse2009/pdf/articles/Muehlenstaedt%20Kuhnt.pdf