Министерство образования и науки Российской Федерации

Новосибирский Государственный Технический Университет

Кафедра прикладной математики

Лабораторная работа №1

по дисциплине: «Непрерывные математические модели»

Факультет: ПМИ

Группа: ПММ-71

Студенты: Кочнев А.

Замащикова А.

Вариант: 2

Преподаватели: Персова М.Г.

Вагин Д.В.

Новосибирск

2017

*1. Цель работы*

Разработать программу построения сглаживающего сплайна с использованием кусочно-полиномиальных эрмитовых базисных функций третьего порядка в одномернных, двумерных или трехмерных областях и опробовать ее при решении задач фильтрации для произвольных наборов зашумленных данных и при решении задач выдачи численного решения и его производных по набору весов конечноэлементного решения для определенного типа конечных элементов.

*2. Вариант задания*

Решение задачи фильтрации в двумерной области. При построении сглаживающего сплайна использовать параметр регуляризации .

*3. Постановка задачи*

Пусть задан некоторый набор значений , где - точка в одномерной, двумерной или трехмерной области,  - значение функции в этой точке,  - некоторое значение вектора весов. Также заданы значения параметров регуляризации  и .

Под задачей сглаживания будем понимать построение достаточно гладкой функции, значения которой в точках  максимально близки к значениям .

Алгоритм действий:

1) Построение сплайна по зашумленным значениям (ошибка распределения по нормальному закону распределения);

2) Фильтрация полученных значений;

3) Построение сплайна по новым исходным данным, полученным на предыдущем этапе.

Построение сплайна

Коэффициенты сплайна находим из СЛАУ:

Ax=b

Где матрица A и вектор b определяются соотношениями:

Третье слагаемое может быть собранно из локальных матриц вида :

*.*

Компоненты локальных матриц матрицы первого слагаемого можно вычислить по формуле:

Суммирование ведется только по тем j, для которых точки принадлежат текущему элементу.

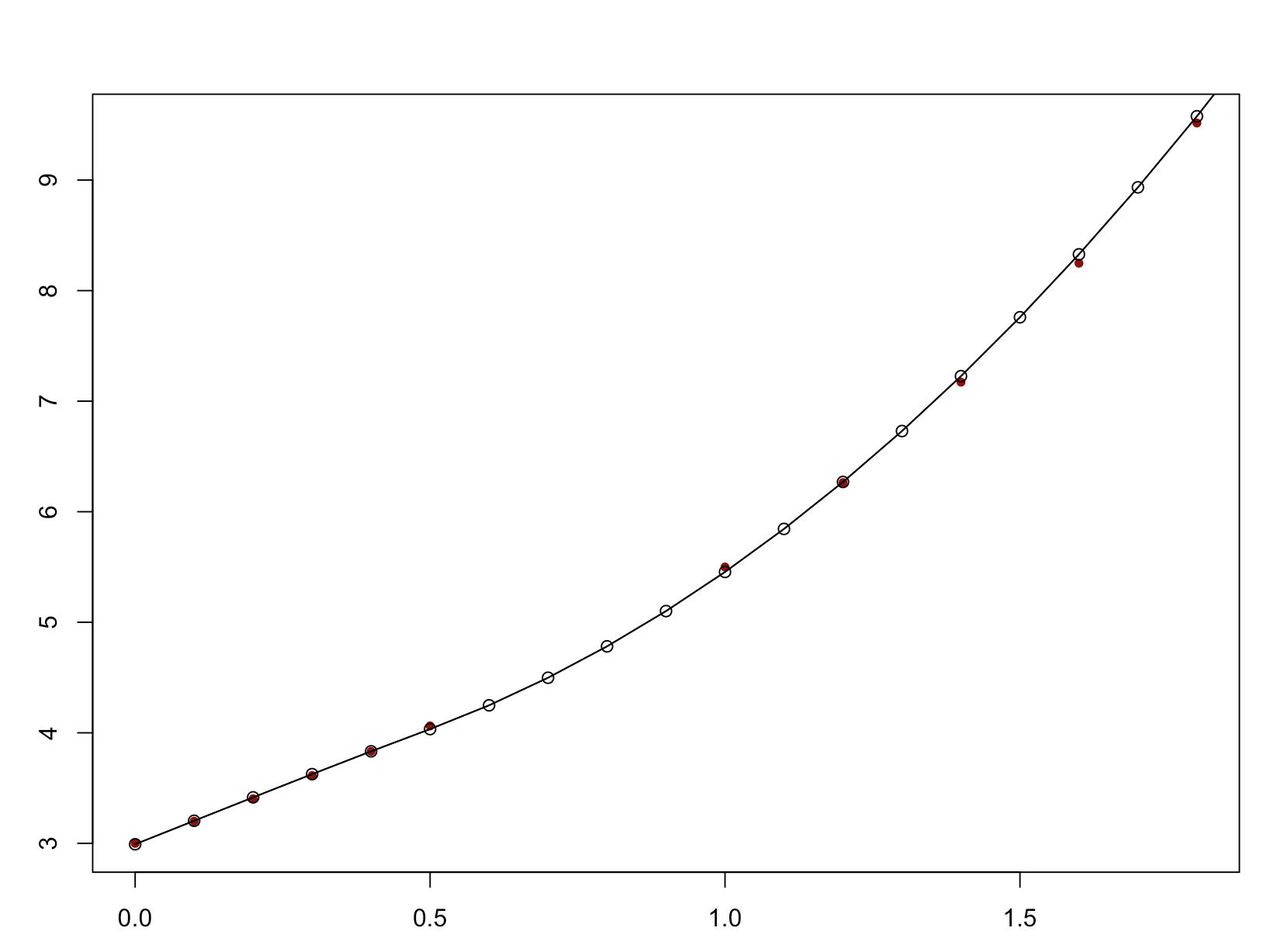
Фильтрация

Находим среднее разности значений сплайна и значений в точках , в тех точках, где такое отклонение больше среднего, значительно уменьшаем коэффициент

*4. Тесты*

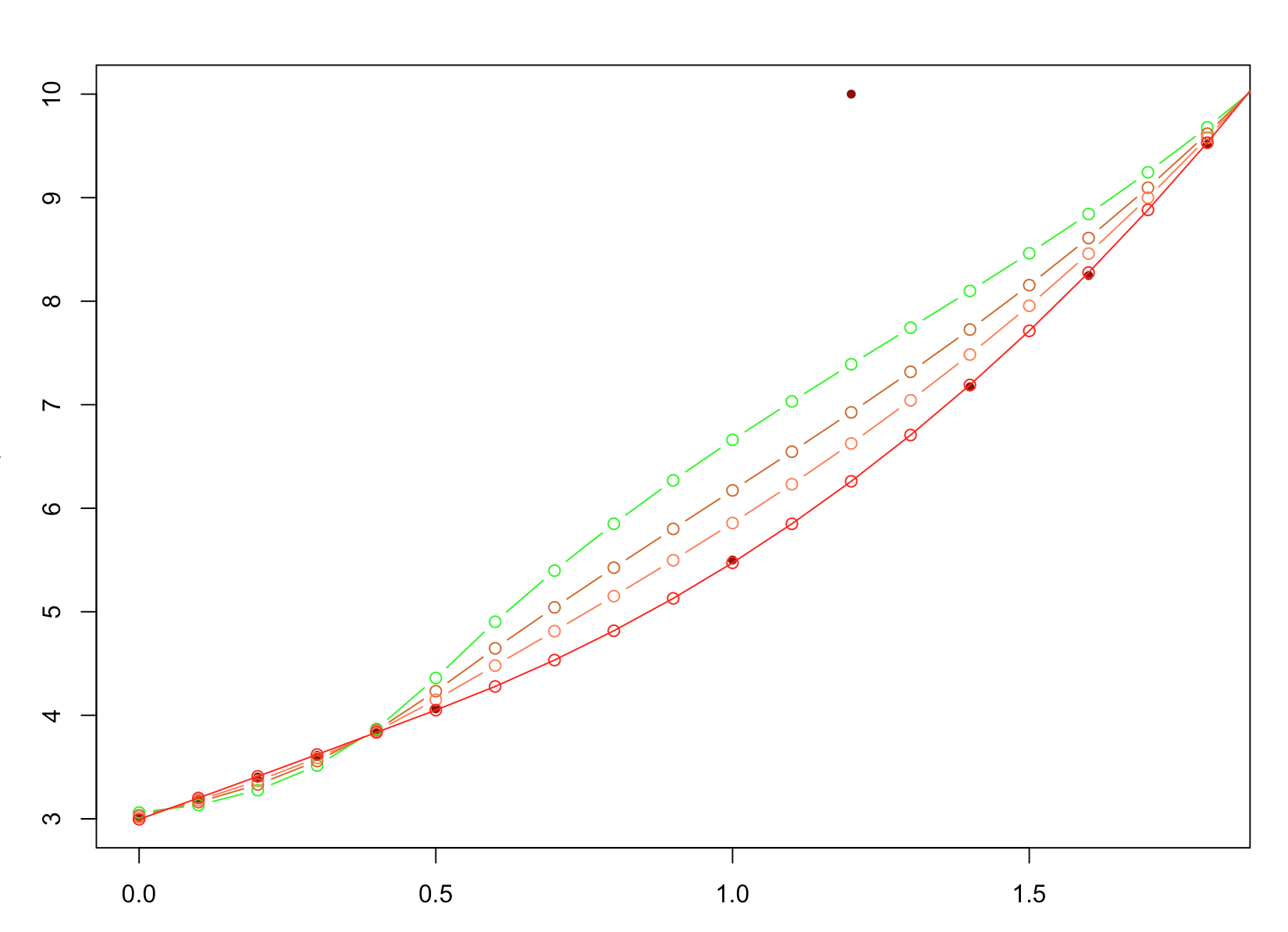
11 точек –, 2 КЭ, .

Кубическая функция



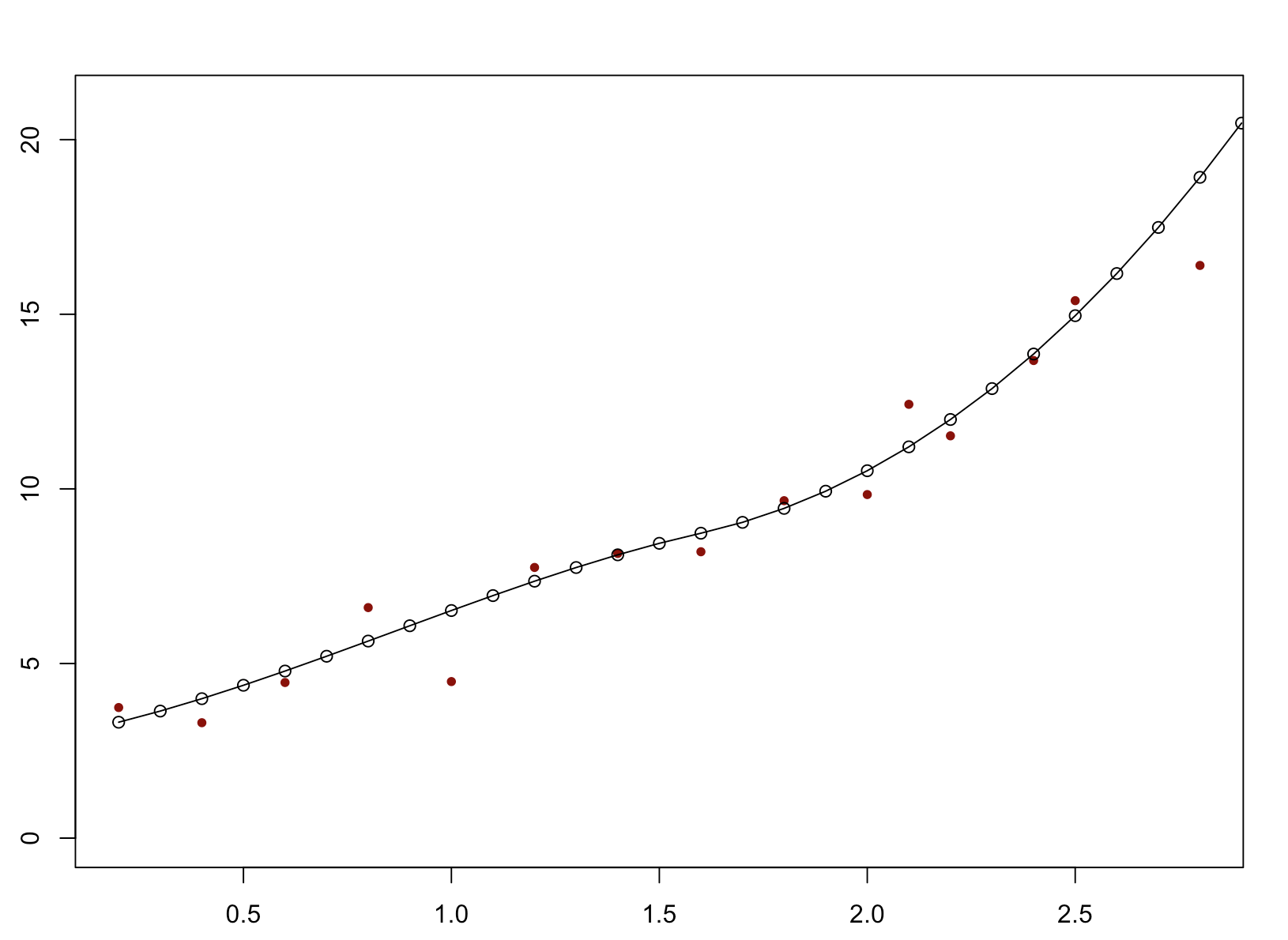
11 точек –, 2 КЭ, .

Кубическая функция с одним выбросом



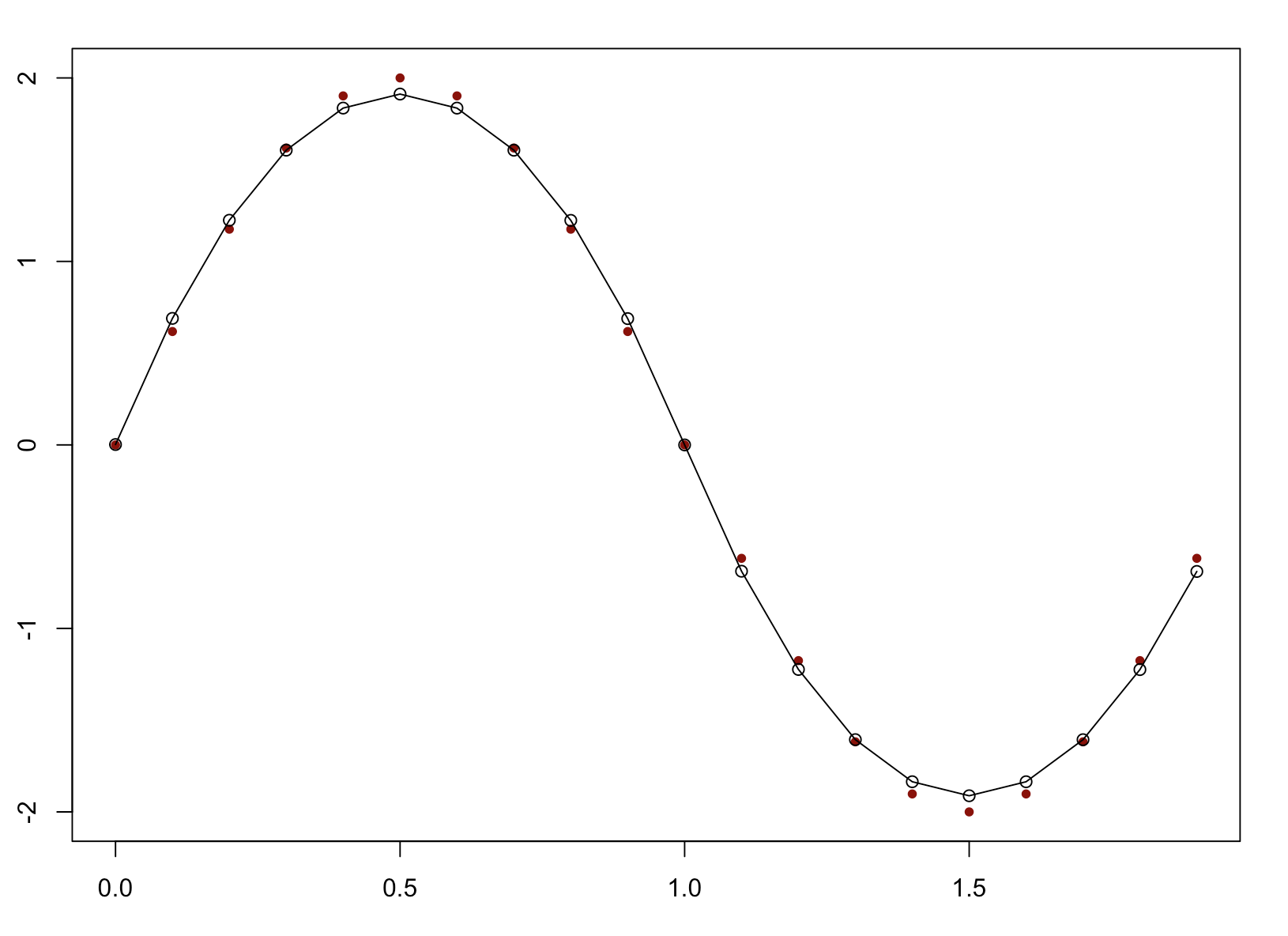
Кубическая функция с нормальным стандартным зашумлением зашумлением

16 точек –, 2 КЭ, .



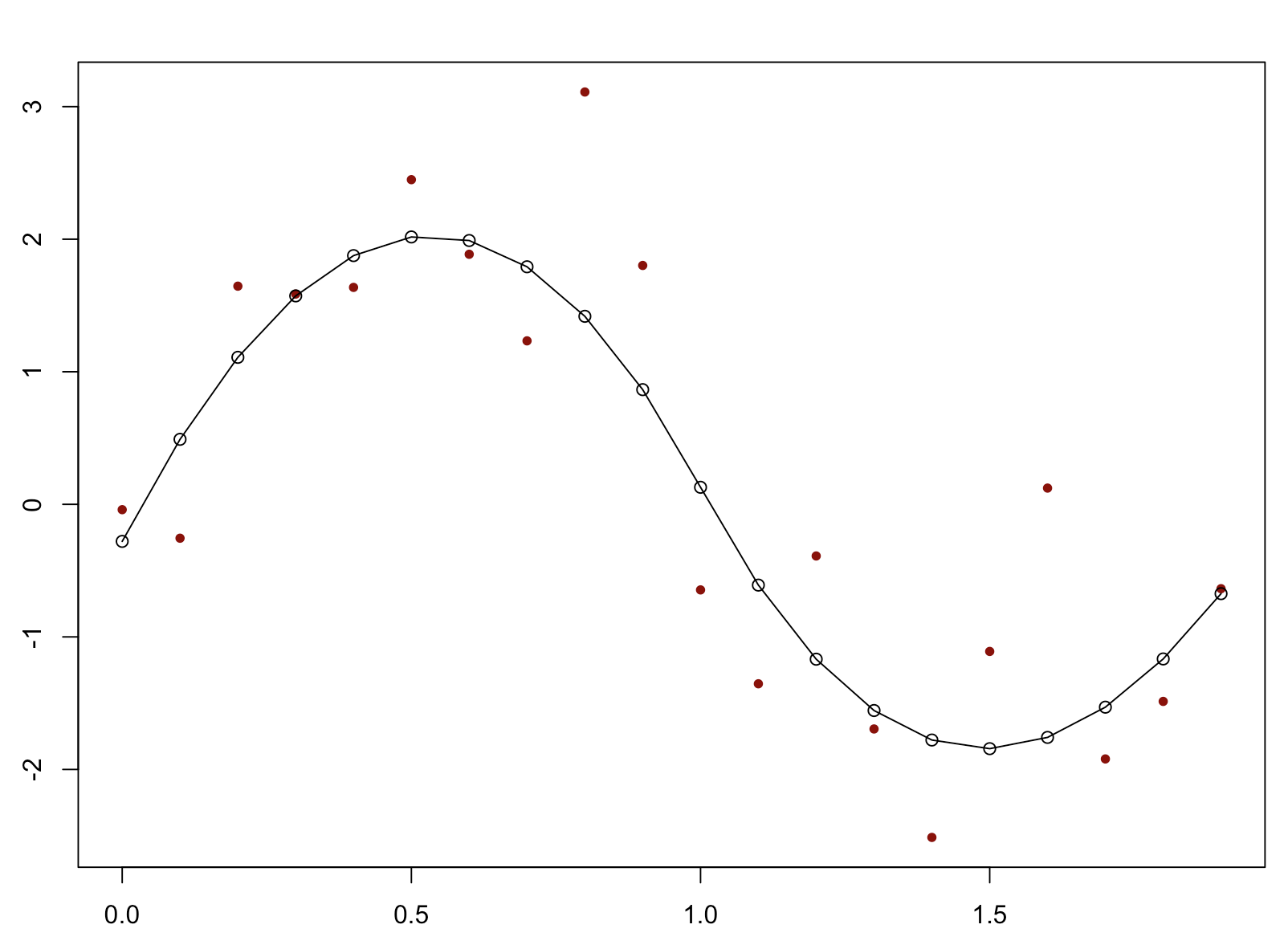
Не представимая в кусочно-кубическом базисе функция ()

21 точка –, 2 КЭ, .



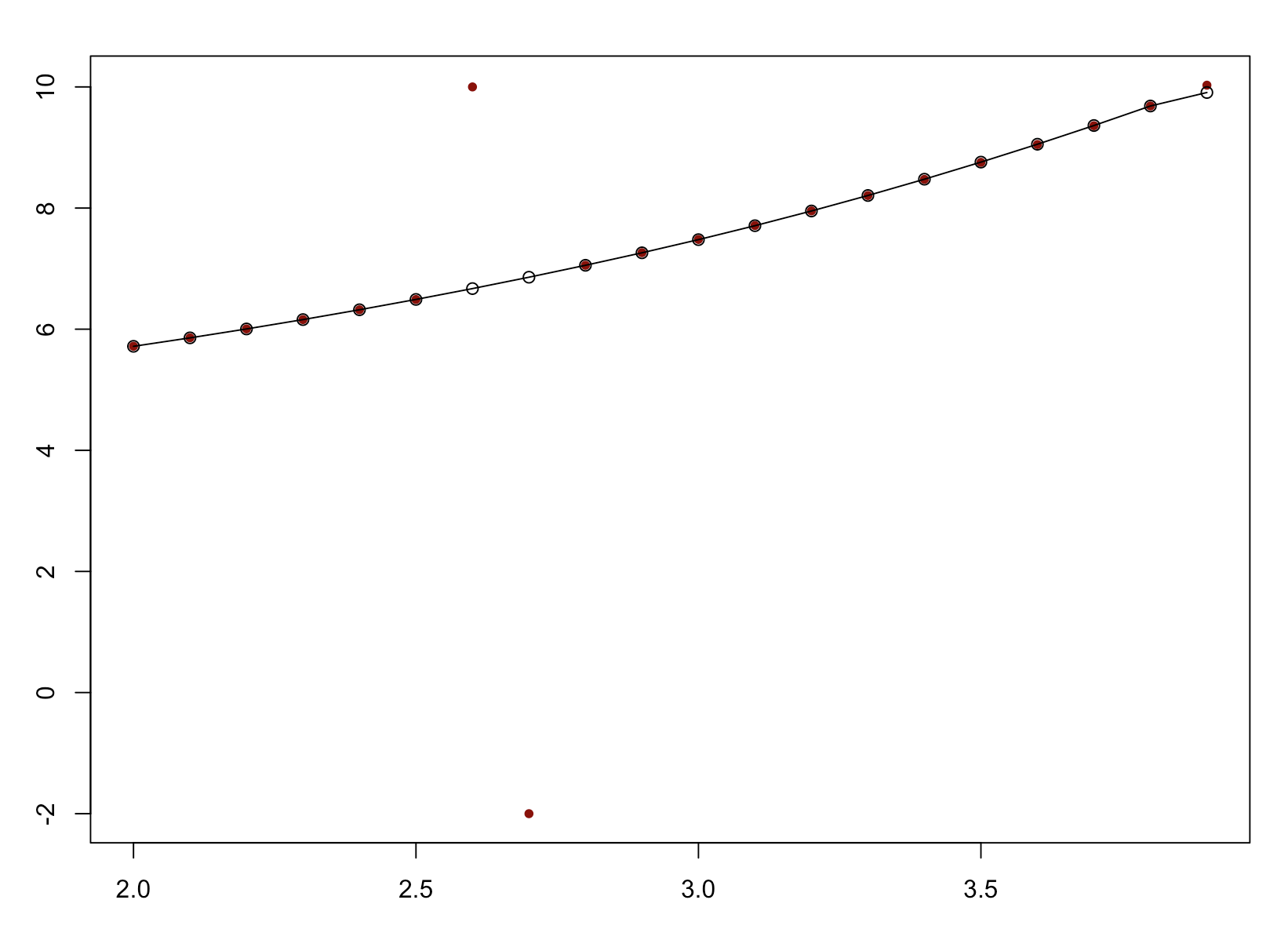
Не представимая в кусочно-кубическом базисе функция () с зашумлением

21 точка –, 2 КЭ, .



Не представимая в кусочно-кубическом базисе функция ( с зашумлением

20 точек –, 2 КЭ, .



*5. Текст программы*

*Основной модуль*

#include **"header.h"**#include **<iostream>  
  
using namespace** std;  
  
**int** main( **int** argc, **const char** \*argv[] ) {  
 grid\_type grid;  
  
 grid = read\_grid();  
  
 vector<**double**> b\_vector = calc\_global\_vector(grid);  
 global\_matrix\_type global\_matrix = calc\_global\_matrix(grid);  
 vector<**double**> q = calc\_spline\_params(global\_matrix, b\_vector);  
 diff\_type diff = difference(grid, q);  
 cout<<**"DIFF: "**<<diff.mean<<endl;  
 cout<<**"HAS BIG ERR: "**<<!diff.isSmaller<<endl;  
  
  
 **for** (**int** j = 0; j < max\_iteration && !diff.isSmaller; ++j) {  
 grid = update\_grid(grid, diff, q);  
 b\_vector = calc\_global\_vector(grid);  
 global\_matrix = calc\_global\_matrix(grid);  
 q = calc\_spline\_params(global\_matrix, b\_vector);  
 diff = difference(grid, q);  
 cout<<**"DIFF: "**<<diff.mean<<endl;  
 }  
  
  
  
 cout<<**"Q vector: "**<<endl;  
 **for** (**int** i = 0; i < global\_matrix\_size; ++i) {  
 cout<<q[i]<<**" "**;  
 }  
 cout<<endl;  
  
 **double** x = grid.points[0][0];  
 **double** x\_end = grid.points[grid.points.size() - 1][grid.points[grid.points.size() - 1].size() -1];  
 **double** step = 0.1;  
  
 **while**(x <= x\_end){  
 cout<<**"X: "**<<x<<**" P(x): "**<<spline(x,grid,q)<<endl;  
 x += step;  
 }  
  
  
 **return** 0 ;  
}

*Операции с локальными и глобальными матрицами*

**void** add\_local\_matrix\_to\_global(local\_matrix\_type local\_matrix, global\_matrix\_type\* global\_matrix, **int** elem\_number) {  
 **int** offset = elem\_number \* 2;  
 **for** (**int** i = 0; i < local\_matrix\_size; ++i) {  
 **for** (**int** j = 0; j < local\_matrix\_size; ++j) {  
 (\*global\_matrix)[offset \* elem\_number+i][offset \* elem\_number+j] += local\_matrix[i][j];  
 }  
 }  
}  
  
global\_matrix\_type calc\_global\_matrix(grid\_type grid) {  
 *// Расчет локальных матриц и векторов + добавление в глобальную* global\_matrix\_type global\_matrix = init\_global\_matrix();  
 **for** (**int** i = 0; i < omega\_count; ++i) {  
 local\_matrix\_type beta\_matrix = local\_beta\_matrix(grid.h\_length[i]);  
 local\_matrix\_type a\_matrix = local\_a\_matrix(grid.h\_length[i], i, grid);  
 local\_matrix\_type local\_matrix = sum\_local\_matrix(a\_matrix, beta\_matrix);  
  
 add\_local\_matrix\_to\_global(local\_matrix, &global\_matrix, i);  
 }  
  
 **return** global\_matrix;  
}

*Расчет коэффициентов сплайна и его значений*

#include **"header.h"  
  
using namespace** std;  
  
**double** spline(**double** x, grid\_type grid, vector<**double**> q) {  
 **short** omega\_number;  
 **bool** finded = **false**;  
 **double** result = 0;  
  
 *// определяем КЭ* **for**(**short** i = 0; i < grid.points.size() && !finded; i++){  
 *// ожидаем что элементы в сетке упорядочены* **double** start = i == 0 ?  
 grid.points[0][0] : grid.points[i - 1][grid.points[i - 1].size() - 1];  
 **double** end = grid.points[i][grid.points[i].size() - 1];  
 **if**(x > start && x < end){  
 finded = **true**;  
 omega\_number = i;  
 }  
 }  
  
 **double** x\_left = omega\_number == 0 ?  
 grid.points[omega\_number][0] :  
 grid.points[omega\_number - 1][grid.points[omega\_number - 1].size() - 1];  
  
 **double** h = grid.points[omega\_number][grid.points[omega\_number].size() - 1] - x\_left;  
 **for** (**short** j = 0; j < local\_matrix\_size; ++j) {  
 **double** value = psi(j, h, x\_left, x) ;  
 result += value \* q[j + omega\_number \* 2];  
 }  
  
 **return** result;  
}

vector<**double**> calc\_spline\_params(global\_matrix\_type global\_matrix, vector<**double**> b\_vector){  
 global\_matrix\_type l\_matrix = init\_global\_matrix();  
 global\_matrix\_type u\_matrix = init\_global\_matrix();  
  
 lu\_decomp(global\_matrix, l\_matrix, u\_matrix);  
  
 vector<**double**> x\_1 = forward\_solution(l\_matrix, b\_vector);  
 **return** backward\_solution(u\_matrix, x\_1);  
}