实验报告

一、实验目的

从下面函数表，分别用

（1） 分段线性插值；
（2） 分段二次插值；
（3） 全区间拉格朗日插值；

计算 $f(0.15)$，$f(0.31)$ 和 $f(0.47)$ 的值。要求：(a) 利用程序自动选择在插值计算中所需要的节点；(b) 把原始数据点以及最后得到的拉格朗日插值多项式函数曲线都绘制在 MATLAB Figure 里，便于观察插值效果。

x	0.0	0.1	0.195	0.3	0.401	0.5
$f(x)$	0.39894	0.39695	0.39142	0.38138	0.36812	0.35206

二、实验方法（要求用自己的语言描述算法）

1）插值多项式的构造方法

我们假设插值的点为 $((x_0, y_0),\dots,(x_n, y_n))$，插值多项式为 $L_n(x)=\sum_{k=0}^{n}\left[\prod_{i=0,i\neq k}^{n}(\frac{x-x_i}{x_k-x_i})\right]\bullet y_k$，再次基础上，我们需要定义多项式的乘法和加法操作，以完成插值多项式的构造。

在本例子中，向量 $(a_0, a_1\cdots a_n)$ 表示多项式 $a_0+a_1x+\cdots+a_nx^n$。并在此基础上定义多项式的加法和乘法。

2）伪代码：

```
# 输入 x, y=f(x), exp 为插值多项式，返回多项式曲线列表[..begin, end, curve_vector]
# + 为多项式加法， * 为多项式乘法
def ploy_ana(x, y, exp):
    count := len(x) - exp  # 计算有几根曲线
    r = {0}  # r 为插值多项式
    for i in [0, count)
        for a in [0, exp]
            m = {1}  # m 为插值多项式
            for b in [0, exp]
                if a != b  # 当a!b 时，需要乘(x-xb/xa-xb)
                    m = m * {x[b+i], 1} / (x[a+i] - x[b+i])
                m = m * y[a+i]  # 乘yi
            r = r + m  # 把每一项加起来，得到插值多项式
        yield return curve(x[i], x[i + exp])  # 返回曲线


# 在一系列曲线组中求g(x)得值
def get_result([..curve], x):
    for p in curve:
        if(p.start <= x and p.end >= x):
            return p(x)
```

三、实验代码

见附录

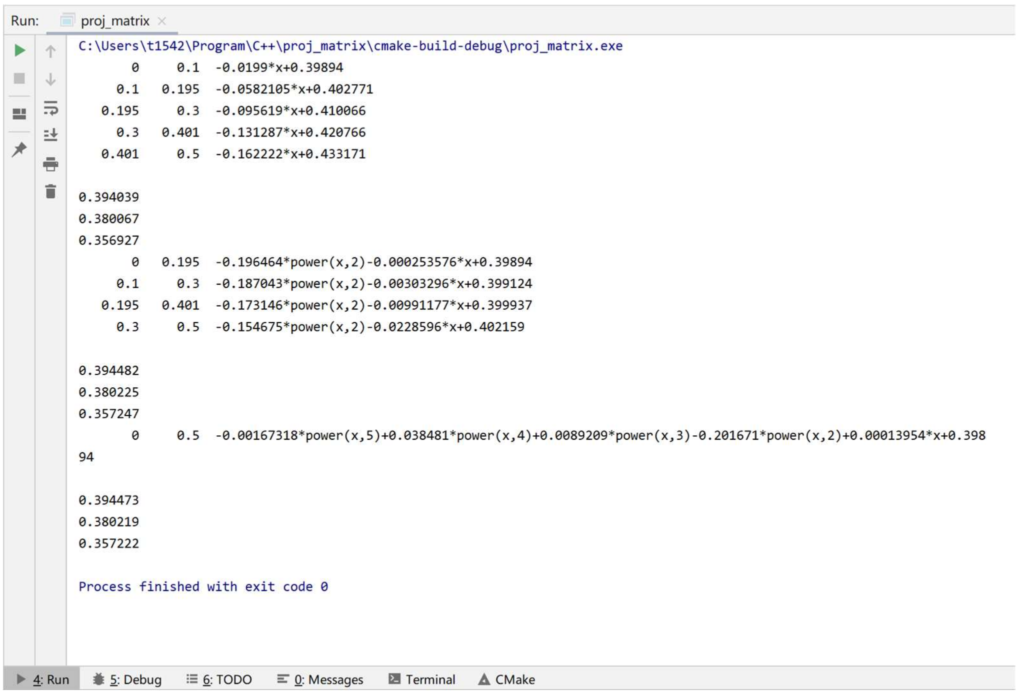
matlab 脚本

```
clear
clc

% points
figure()
x1 = [0 0.1 0.195 0.3 0.401 0.5];
y1 = [0.39894 0.39695 0.39142 0.38138 0.36812 0.35206];
plot(x1,y1,'red*')
% curve
x = 0:0.01:0.5;
f = -0.00167318*power(x,5)+0.038481*power(x,4)+0.0089209*power(x,3)-
0.201671*power(x,2)+0.00013954*x+0.39894;
hold on
plot(x, f, 'red-')
% linear
x = 0:0.01:0.1;
f = -0.0199*x+0.39894;
hold on
plot(x, f, 'black-')
x = 0.1:0.01:0.195;
f = -0.0582105*x+0.402771;
hold on
plot(x, f, 'black-')
x = 0.195:0.01:0.3;
f = -0.095619*x+0.410066;
hold on
plot(x, f, 'black-')
x = 0.3:0.01:0.401;
f = -0.131287*x+0.420766;
```

```
hold on
plot(x, f, 'black-')
x = 0.401:0.01:0.5;
f = -0.162222*x+0.433171;
hold on
plot(x, f, 'black-')
% para
x = 0:0.01:0.195;
f = -0.196464*power(x,2)-0.000253576*x+0.39894;
hold on
plot(x, f, '-.blue')
x = 0.1:0.01:0.3;
f = -0.187043*power(x,2)-0.00303296*x+0.399124;
hold on
plot(x, f, '-.blue')
x = 0.195:0.01:0.401;
f = -0.173146*power(x,2)-0.00991177*x+0.399937;
hold on
plot(x, f, '-.blue')
x = 0.3:0.01:0.5;
f = -0.154675*power(x,2)-0.0228596*x+0.402159;
hold on
plot(x, f, '-.blue')
```

四、实验结果及其讨论



```
C:\Users\t1542\Program\C++\proj_matrix\cmake-build-debug\proj_matrix.exe
        0     0.1  -0.0199*x+0.39894
      0.1   0.195  -0.0582105*x+0.402771
    0.195     0.3  -0.095619*x+0.410066
      0.3   0.401  -0.131287*x+0.420766
    0.401     0.5  -0.162222*x+0.433171

0.394039
0.380067
0.356927
        0   0.195  -0.196464*power(x,2)-0.000253576*x+0.39894
      0.1     0.3  -0.187043*power(x,2)-0.00303296*x+0.399124
    0.195   0.401  -0.173146*power(x,2)-0.00991177*x+0.399937
      0.3     0.5  -0.154675*power(x,2)-0.0228596*x+0.402159

0.394482
0.380225
0.357247
        0     0.5  -0.00167318*power(x,5)+0.038481*power(x,4)+0.0089209*power(x,3)-0.201671*power(x,2)+0.00013954*x+0.398
94

0.394473
0.380219
0.357222

Process finished with exit code 0
```

图 1 程序执行结果

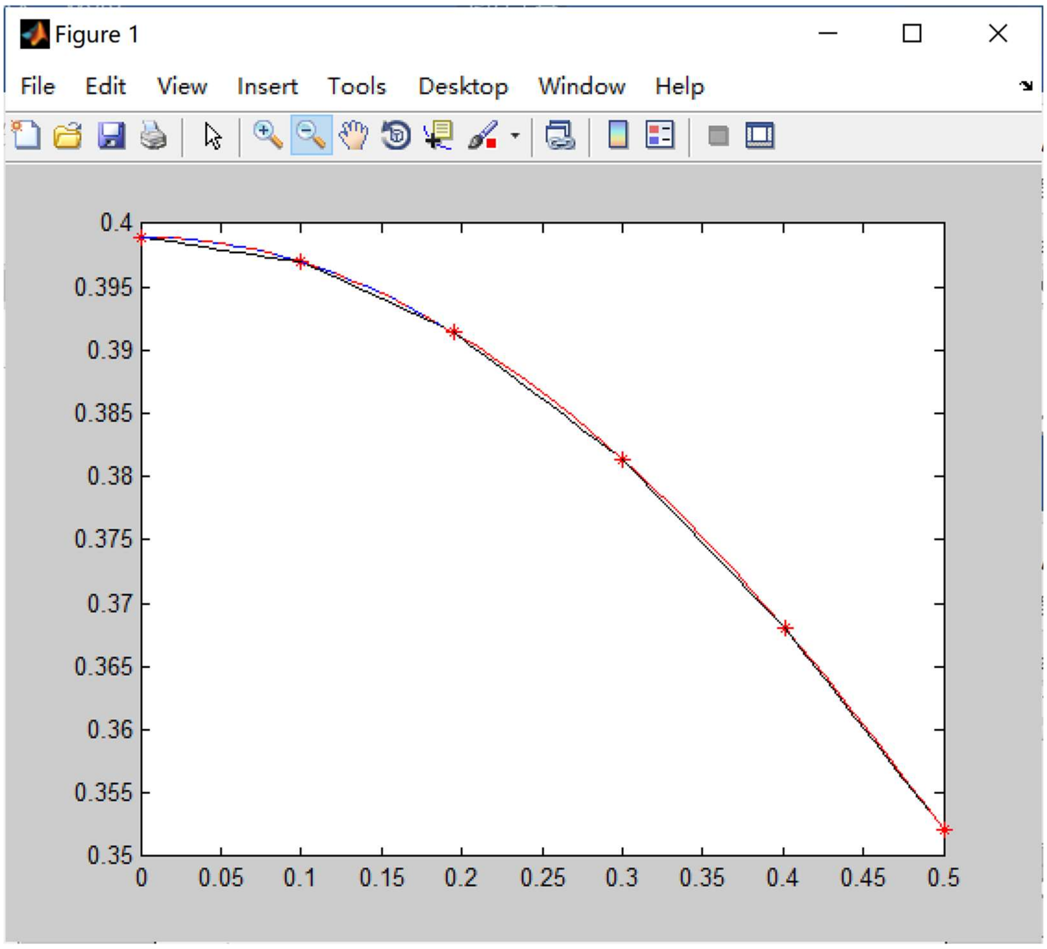


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
0.4
0.395
0.39
0.385
0.38
0.375
0.37
0.365
0.36
0.355
0.35
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5

图 2 插值曲线

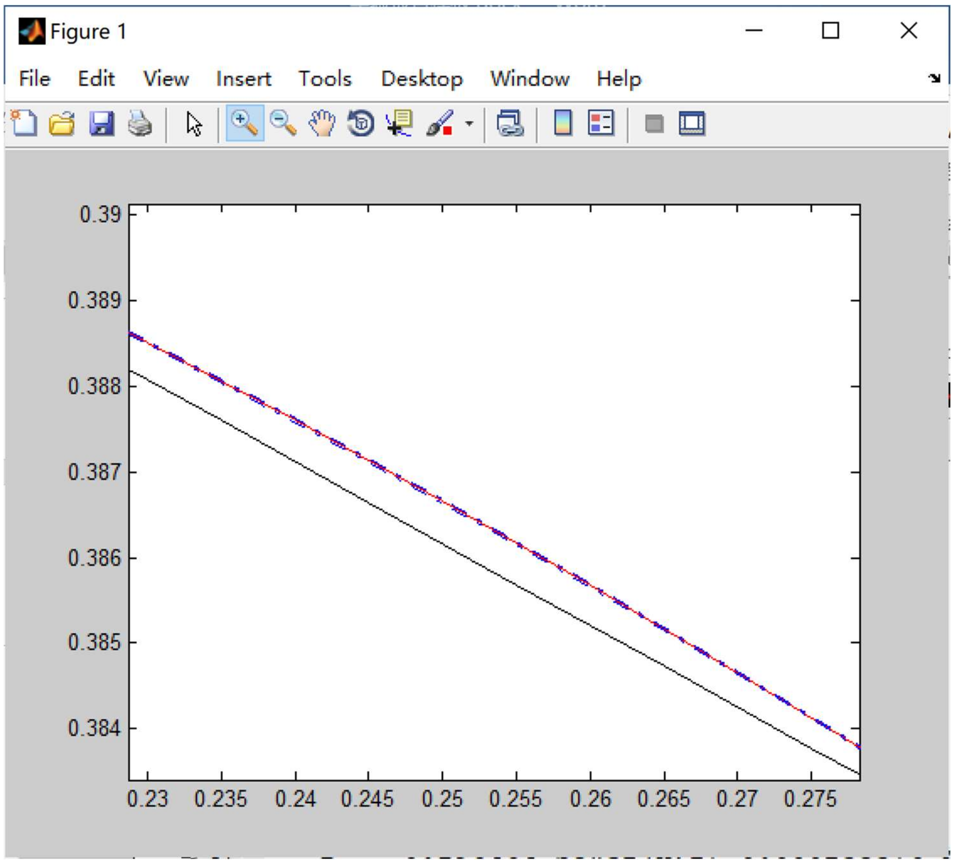


图 3 插值曲线-局部

在本例中，三条曲线中，二次曲线和拉格朗日插值曲线非常接近，因此难以分辨。黑色线代表线性插值曲线，红色线为拉格朗日插值曲线，蓝色为抛物线插值曲线。观察可知，除了线性插值，其他两条曲线都是比较光滑的。

五、总结

通过实验，对拉格朗日插值多项式和多项式的运算有了更加深入的了解，可以通过编程求解插值多项式。为了避免 Runge 现象，一般使用分段抛物线插值及分段四次插值，这样有较好的精确度。

六、附录（代码）

main.cpp

```
#include <iostream>
#include "matrix.h"
#include <vector>
#include <cmath>
#include "points.h"
#include "insertion_result.h"

using namespace std;
int main() {
    points p({{0, 0.39894},
              {0.1, 0.39695},
              {0.195, 0.39142},
              {0.3, 0.38138},
              {0.401, 0.36812},
              {0.5, 0.35206}});
//    points p({{0, 0},
//              {1,1},
//              {2,2}});
    insertion_result r = p.get_ploy_exp(1);
    cout << r << endl;
    r.print_result({0.15, 0.31,  0.47});
    insertion_result r2 = p.get_ploy_exp(2);
    cout << r2 << endl;
    r2.print_result({0.15, 0.31,  0.47});
    insertion_result r3 = p.get_ploy_exp(5);
    cout << r3 << endl;
    r3.print_result({0.15, 0.31,  0.47});
}
```

matrix.cpp（仅部分）

```
matrix matrix::vector_add(const matrix &m1, const matrix &m2) {
    if(m1._row_count == 1 && m2._row_count == 1){
        const matrix* a = &m1;
        const matrix* b = &m2;
        if(a->_column_count < b->_column_count){
            a = &m2;
            b = &m1;
        }
        matrix m(1, a->_column_count);
        m.fills = new double[a->_column_count];
        for(int i = 0; i < b->_column_count; ++i)
            m.fills[i] = a->fills[i] + b->fills[i];
        for(int i = b->_column_count; i<a->_column_count;++i)
            m.fills[i] = a->fills[i];
        //m.vector_adjust();
        return m;
    } else {
        return matrix::error_instance();
    }
}

matrix matrix::vector_multiply(const matrix& m1, const matrix& m2) {
    if(m1._row_count == 1 && m2._row_count == 1){
        int length = m1._column_count + m2._column_count -1;
        matrix m(1, length);
        m.fills = new double[length];
        fill(m.fills, length);
        for(int i = 0; i < m1._column_count; ++i){
            for(int j = 0; j < m2._column_count; ++j){
                m.fills[i+j] += m1.fills[i] * m2.fills[j];
            }
        }
        m.vector_adjust();
        return m;
    } else {
        return matrix::error_instance();
    }
}

matrix matrix::row_at(int row) {
    matrix m(1, _column_count);
    m.fills = new double[_column_count];
    for (int i = 0; i < _column_count; ++i) {
        m.fills[i] = fills[row * _column_count + i];
    }
    return m;
}

matrix matrix::column_at(int column) {
    matrix m(_row_count, 1);
    m.fills = new double[_row_count];
    for (int i = 0; i < _row_count; ++i) {
        m.fills[i] = fills[i * _column_count + column];
    }
    return m;
}

double matrix::at(int row, int column) {
    return fills[row * _column_count + column];
}

matrix operator*(double a, const matrix& m) {
    matrix m1 = matrix(m._row_count, m._column_count);
    int length = m._row_count * m._column_count;
    m1.fills = new double[length];
    for (int i = 0; i < length; ++i) {
        m1.fills[i] = a * m.fills[i];
    }
    return m1;
}

void matrix::vector_adjust() {
    if(_row_count == 1){
        int length = _column_count;
        int i;
        for (i = length - 1 ; i > 0; -- i) {
            if(fills[i] != 0)
                break;
        }
        auto* f = new double[i + 1];
        for (int j = 0; j < i + 1; ++j) {
            f[j] = fills[j];
        }
        delete fills;
        fills = f;
    }
}

string matrix::to_expression() {
    if(_row_count == 1){
        ostringstream s;
        for (int i = _column_count - 1; i >= 0; --i) {
            if(i < _column_count - 1 && fills[i] > 0)
                s << "+";
            s << fills[i];
            if(i > 0){
                s << "*";
                if(i == 1)
                    s << "x";
                else
                    s << "power(x," << i << ")";
            }

        }
        return s.str();
    } else {
        return "";
    }
}

double matrix::expression_invoke(double v) {
    if(_row_count == 1){
        double r = 0;
        for (int i = 0; i < _column_count; ++i) {
            r += fills[i] * pow(v,i);
        }
        return r;
    } else {
        return -1;
    }
}
```

points.h points.cpp

```
#pragma once
#include "matrix.h"
#include "ploy.h"
#include <vector>
#include "insertion_result.h"

using namespace std;

enum class insertion_type{
    linear,
    para,
    lagrange
};

class points {
public:
    explicit points(const matrix& m): _data(m){}
    insertion_result get_ploy_exp(int exp);
private:
    matrix _data;
};
```

```
#include "points.h"



insertion_result points::get_ploy_exp(int exp) {
    if(exp >= 1){
        vector<ploy> results;
        int r_count = _data.row_count() - exp;
        for(int i = 0; i < r_count; ++i){
            matrix r = {{0}};
            for(int a = 0; a <= exp; ++a){
                matrix m = {{1}};
                for(int b = 0; b <= exp; ++b){
                    if(a != b){
                        matrix u = {{-_data.at(i + b, 0), 1}};
                        u = 1.0 / (_data.at(i+a, 0) - _data.at(i+b,0)) * u;

                        m = matrix::vector_multiply(m, u);

                    }
                }
                m = _data.at(i+a,1) * m;
                //m.print_with_title("b");
                //cout << _data.at(i+a, 1) << endl;
                //r.print_with_title("t1");
                r = matrix::vector_add(r, m);
                //r.print_with_title("t");
            }
            //cout << r.to_expression() << endl;
            results.emplace_back(ploy(_data.at(i, 0), _data.at(i+exp, 0),
r));
        }
        return results;
    } else {
        return vector<ploy>();
    }
}
```

ploy.h

```
#pragma once
#include "matrix.h"

class ploy {
public:
    ploy(double start, double end,const matrix& ploy_ana): _start(start),
_end(end), _curve(ploy_ana){}
    double start(){
        return _start;
    }
    double end(){
        return _end;
    }
    matrix curve(){
        return _curve;
    }
private:
    double _start;
    double _end;
    matrix _curve;
};
```

insertion_result.h insertion_result.cpp

```
#pragma once
#include "matrix.h"
#include "ploy.h"
#include <utility>
#include <vector>
#include <initializer_list>

using namespace std;

class insertion_result {
public:
    insertion_result(vector<ploy> d): data(std::move(d)){}
    friend ostream& operator << (ostream& o, const insertion_result& r);
    void print_result(initializer_list<double> v);
private:
    vector<ploy> data;
};
```

```
#include "insertion_result.h"

ostream &operator<<(ostream &o, const insertion_result& r) {
    int i = 0;
    for(auto p:r.data){
        o << setw(8) << p.start() << setw(8) << p.end() << "  " <<
p.curve().to_expression() << endl;
    }
    return o;
}

void insertion_result::print_result(initializer_list<double> v) {
    for(auto p:v){
        cout << setw(8);
        for(auto d:data){
            if(p >= d.start() && p<= d.end()){
                cout << setw(8) << d.curve().expression_invoke(p);
                break;
            }
        }
        cout << endl;
    }
}
```