北京航空航天大学数学科学学院实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称：科学计算通识实验课 | | 实验名称： 实验七： 数值积分算法 | |
| 实验类型： 演示性实验□ 验证性实验☑ 综合性实验□ 设计性实验□ | | | |
| 班级：暑期班 | 姓名：涂磊 | | 学号：22377211 |
| 实验日期： 2024/7/15 | 指导教师：冯成亮 | | 实验成绩： |
| 实验环境：（所用仪器设备及软件）  Windows + VS-code, Ubuntu 20.04.6 + g++ | | | |
| 实验目的与实验内容：  【目的要求】  通过本实验使学生进一步熟悉个人电脑上C++代码的编写与调试，服务器上的代码编译与运行；熟悉数值积分中闭型牛顿-科特斯面积公式（梯形公式T、辛普森公式S、辛普森3/8公式、布尔公式B），并熟练应用它们的复化/组合形式，熟悉它们的误差评估与积分精度关系；对均匀二次加密下牛顿面积公式（T型、S型与B型）的递归过程做了解；熟悉高斯-勒让德积分的逻辑过程，熟练掌握2点和3点高斯积分公式，掌握利用高斯-勒让德变换对定积分做高精度数值计算。  【实验内容】  【实验内容】  实验1.1：（直接数值积分函数1）  分别编写四种牛顿-科斯特面积公式对应的积分函数，并使用它们计算函数在[0,1]上的积分，比较它们的误差。  实验1.2：（复化数值积分函数2）  对函数,使用复化梯形公式与复化辛普森公式和11个采样点，计算其在区间[1,6]上的积分，比较它们的误差。  实验1.3：（复化数值积分函数3）  对函数,使用复化梯形公式与复化辛普森公式和11,21,41,81,161个采样点，计算其在区间[1,6]上的积分，比较它们的误差。  实验1.4\*：（复化数值积分函数4）  对函数 ,使用复化梯形公式与复化辛普森公式计算其在区间[2,7]上的积分，迭代计算M与对应的h，使得误差小于5.0E-9.  实验2.1：（数值积分函数6）  利用2点高斯积分对定积分  做逼近，比较其与梯形公式（h=2）和辛普森公式(h=1)的误差区别；  实验2.2：（数值积分函数4）  利用3点高斯积分对定积分  做逼近，比较其与布尔公式(h=1)的误差区别；  记录网格加密过程（M=1,2,4,8，16）中2种算法的误差表现。  实验3.1：（数值积分曲线弧长）  给定参数路径，其中    通过均匀间距的参数进行定义，采用弧长积分公式  ，  积分求解该曲线的长度。    实验3.2：（数值积分二维翼型的升力系数和阻力系数）  已知二维翼型形状如下图所示    现有翼型的边缘点坐标x, y和每点处的压力系数Cp组成的数据表格（cp1.plt）,    积分求解该翼型的升力和阻力。  ，  为改点处指向翼型内部的单位法向（） | | | |
| 实验1.1：（直接数值积分函数1）  #include <iostream>  using namespace std;  #include "func.hpp"  double f(double x) {      return 1 + exp(-x) \* sin(4\*x);  }  double f\_x(double x) {      return 0;  }  double f\_y(double x) {      return 0;  }  int main(void) {      double real = 1.3082506046426;      double result\_T = T(0,1,2);      double result\_S = S(0,1,3);      double result\_S\_3\_8 = S\_3\_8(0,1,4);      double result\_B = B(0,1,5);      cout << fixed;      cout.precision(8);      cout << "\t\treal\_result\tT(f)\t\tS(f)\t\tS\_3\_8(f)\tB(f)" << endl;      cout << "----------------" << endl;      cout <<"result\t\t" << real << "\t" << result\_T << "\t" << result\_S << "\t" << result\_S\_3\_8 << "\t" << result\_B << endl;      cout <<"error\t\t"<< 0 << "\t\t" << err(real,result\_T) << "\t" << err(real,result\_S) << "\t" << err(real,result\_S\_3\_8) << "\t" << err(real,result\_B) << endl;      cout << "----------------" << endl;      return 0;  }    实验1.2：（复化数值积分函数2）  #include <iostream>  using namespace std;  #include "func.hpp"  double f(double x) {      return 2 + sin(2\*sqrt(x));  }  double f\_x(double x) {      return 0;  }  double f\_y(double x) {      return 0;  }  int main(void) {      double real = 2\*6 - sqrt(6)\*cos(2\*sqrt(6)) + (1.0/2) \* sin(2\*sqrt(6)) -(2\*1 - sqrt(1)\*cos(2 \* sqrt(1)) + (1.0/2) \* sin(2\*sqrt(1)));      double result\_C\_T = C\_T(1,6,11);      double result\_C\_S = C\_S(1,6,11);      cout << fixed;      cout.precision(8);      cout << "\t\treal\_result\tC\_T(f)\t\tC\_S(f)" << endl;      cout << "----------------" << endl;      cout <<"result\t\t" << real << "\t" << result\_C\_T << "\t" << result\_C\_S << endl;      cout <<"error\t\t"<< 0 << "\t\t" << err(real,result\_C\_T) << "\t" << err(real,result\_C\_S) << endl;      cout << "----------------" << endl;      return 0;  }    实验1.3：（复化数值积分函数3）  #include <iostream>  using namespace std;  #include "func.hpp"  double f(double x) {      return 2 + sin(2\*sqrt(x));  }  double f\_x(double x) {      return 0;  }  double f\_y(double x) {      return 0;  }  int main(void) {      double real = 2\*6 - sqrt(6)\*cos(2\*sqrt(6)) + (1.0/2) \* sin(2\*sqrt(6)) -(2\*1 - sqrt(1)\*cos(2 \* sqrt(1)) + (1.0/2) \* sin(2\*sqrt(1)));      double result\_C\_T\_11 = C\_T(1,6,11);      double result\_C\_T\_21 = C\_T(1,6,21);      double result\_C\_T\_41 = C\_T(1,6,41);      double result\_C\_T\_81 = C\_T(1,6,81);      double result\_C\_T\_161 = C\_T(1,6,161);      double result\_C\_S\_11 = C\_S(1,6,11);      double result\_C\_S\_21 = C\_S(1,6,21);      double result\_C\_S\_41 = C\_S(1,6,41);      double result\_C\_S\_81 = C\_S(1,6,81);      double result\_C\_S\_161 = C\_S(1,6,161);      cout << fixed;      cout.precision(8);      cout << "C\_T(f)" << endl;      cout << "----------------" << endl;      cout << "n\tC\_T(f)\t\terror" << endl;      cout << "11\t" << result\_C\_T\_11 << "\t" << err(real,result\_C\_T\_11) << endl;      cout << "21\t" << result\_C\_T\_21 << "\t" << err(real,result\_C\_T\_21) << endl;      cout << "41\t" << result\_C\_T\_41 << "\t" << err(real,result\_C\_T\_41) << endl;      cout << "81\t" << result\_C\_T\_81 << "\t" << err(real,result\_C\_T\_81) << endl;      cout << "161\t" << result\_C\_T\_161 << "\t" << err(real,result\_C\_T\_161) << endl;      cout << "----------------" << endl;      cout << endl;      cout << "C\_S(f)" << endl;      cout << "----------------" << endl;      cout << "n\tC\_S(f)\t\terror" << endl;      cout << "11\t" << result\_C\_S\_11 << "\t" << err(real,result\_C\_S\_11) << endl;      cout << "21\t" << result\_C\_S\_21 << "\t" << err(real,result\_C\_S\_21) << endl;      cout << "41\t" << result\_C\_S\_41 << "\t" << err(real,result\_C\_S\_41) << endl;      cout << "81\t" << result\_C\_S\_81 << "\t" << err(real,result\_C\_S\_81) << endl;      cout << "161\t" << result\_C\_S\_161 << "\t" << err(real,result\_C\_S\_161) << endl;      cout << "----------------" << endl;      return 0;  }    实验1.4\*：（复化数值积分函数4）  #include <iostream>  using namespace std;  #include "func.hpp"  double f(double x) {      return 2 + sin(2\*sqrt(x));  }  double f\_x(double x) {      return 0;  }  double f\_y(double x) {      return 0;  }  int main(void) {      double real = 2\*6 - sqrt(6)\*cos(2\*sqrt(6)) + (1.0/2) \* sin(2\*sqrt(6)) -(2\*1 - sqrt(1)\*cos(2 \* sqrt(1)) + (1.0/2) \* sin(2\*sqrt(1)));      double result\_C\_T\_11 = C\_T(1,6,11);      double result\_C\_T\_21 = C\_T(1,6,21);      double result\_C\_T\_41 = C\_T(1,6,41);      double result\_C\_T\_81 = C\_T(1,6,81);      double result\_C\_T\_161 = C\_T(1,6,161);      double result\_C\_S\_11 = C\_S(1,6,11);      double result\_C\_S\_21 = C\_S(1,6,21);      double result\_C\_S\_41 = C\_S(1,6,41);      double result\_C\_S\_81 = C\_S(1,6,81);      double result\_C\_S\_161 = C\_S(1,6,161);      cout << fixed;      cout.precision(8);      cout << "C\_T(f)" << endl;      cout << "----------------" << endl;      cout << "n\tC\_T(f)\t\terror" << endl;      cout << "11\t" << result\_C\_T\_11 << "\t" << err(real,result\_C\_T\_11) << endl;      cout << "21\t" << result\_C\_T\_21 << "\t" << err(real,result\_C\_T\_21) << endl;      cout << "41\t" << result\_C\_T\_41 << "\t" << err(real,result\_C\_T\_41) << endl;      cout << "81\t" << result\_C\_T\_81 << "\t" << err(real,result\_C\_T\_81) << endl;      cout << "161\t" << result\_C\_T\_161 << "\t" << err(real,result\_C\_T\_161) << endl;      cout << "----------------" << endl;      cout << endl;      cout << "C\_S(f)" << endl;      cout << "----------------" << endl;      cout << "n\tC\_S(f)\t\terror" << endl;      cout << "11\t" << result\_C\_S\_11 << "\t" << err(real,result\_C\_S\_11) << endl;      cout << "21\t" << result\_C\_S\_21 << "\t" << err(real,result\_C\_S\_21) << endl;      cout << "41\t" << result\_C\_S\_41 << "\t" << err(real,result\_C\_S\_41) << endl;      cout << "81\t" << result\_C\_S\_81 << "\t" << err(real,result\_C\_S\_81) << endl;      cout << "161\t" << result\_C\_S\_161 << "\t" << err(real,result\_C\_S\_161) << endl;      cout << "----------------" << endl;      return 0;  }          实验2.1：（数值积分函数6）  #include <iostream>  using namespace std;  #include "func.hpp"  double f(double x) {      return 1/(x+2);  }  double f\_x(double x) {      return 0;  }  double f\_y(double x) {      return 0;  }  int main(void) {      double real = 1.09861228866810;      double result\_T = T(-1,1,2);      double result\_S = S(-1,1,3);      double result\_G\_2 = G\_2(-1,1);      cout << fixed;      cout.precision(8);        cout << "\t\treal\_result\tT(f)\t\tS(f)\t\tG\_2(f)" << endl;      cout << "----------------" << endl;      cout <<"result\t\t" << real << "\t" << result\_T << "\t" << result\_S << "\t" << result\_G\_2 << endl;      cout <<"error\t\t"<< 0 << "\t\t" << err(real,result\_T) << "\t" << err(real,result\_S) << "\t" << err(real,result\_G\_2) << endl;      cout << "----------------" << endl;      return 0;  }    实验2.2：（数值积分函数4）  #include <iostream>  using namespace std;  #include "func.hpp"  double f(double x) {      return 1/(x);  }  double f\_x(double x) {      return 0;  }  double f\_y(double x) {      return 0;  }  int main(void) {      double real = 1.609438;      double result\_B = B(1,5,5);      double result\_G\_3 = G\_3(1,5);      cout << fixed;      cout.precision(8);        cout << "\t\treal\_result\tB(f)\t\tG\_3(f)" << endl;      cout << "----------------" << endl;      cout <<"result\t\t" << real << "\t" << result\_B  << "\t" << result\_G\_3 << endl;      cout <<"error\t\t"<< 0 << "\t\t" << err(real,result\_B)  << "\t" << err(real,result\_G\_3) << endl;      cout << "----------------" << endl;      cout << endl;      cout << "G\_3(f)" << endl;      cout << "----------------" << endl;      cout << "M\t\treal\t\tG\_3(f)\t\terror" << endl;      cout << "----------------" << endl;      cout << 1 << "\t\t" << real << "\t" << G\_3(1,5) << "\t" << err(real,G\_3(1,5)) << endl;      cout << 2 << "\t\t" << real << "\t" << G\_3(1,3)+G\_3(3,5) << "\t" << err(real,G\_3(1,3)+G\_3(3,5)) << endl;      cout << 4 << "\t\t" << real << "\t" << G\_3(1,2)+G\_3(2,3)+G\_3(3,4)+G\_3(4,5) << "\t" << err(real,G\_3(1,2)+G\_3(2,3)+G\_3(3,4)+G\_3(4,5)) << endl;      cout << 8 << "\t\t" << real << "\t" << G\_3(1,1.5)+G\_3(1.5,2)+G\_3(2,2.5)+G\_3(2.5,3)+G\_3(3,3.5)+G\_3(3.5,4)+G\_3(4,4.5)+G\_3(4.5,5) << "\t" << err(real,G\_3(1,1.5)+G\_3(1.5,2)+G\_3(2,2.5)+G\_3(2.5,3)+G\_3(3,3.5)+G\_3(3.5,4)+G\_3(4,4.5)+G\_3(4.5,5)) << endl;      cout << 16 << "\t\t" << real << "\t" << G\_3(1,1.25)+G\_3(1.25,1.5)+G\_3(1.5,1.75)+G\_3(1.75,2)+G\_3(2,2.25)+G\_3(2.25,2.5)+G\_3(2.5,2.75)+G\_3(2.75,3)+G\_3(3,3.25)+G\_3(3.25,3.5)+G\_3(3.5,3.75)+G\_3(3.75,4)+G\_3(4,4.25)+G\_3(4.25,4.5)+G\_3(4.5,4.75)+G\_3(4.75,5) << "\t" << err(real,G\_3(1,1.25)+G\_3(1.25,1.5)+G\_3(1.5,1.75)+G\_3(1.75,2)+G\_3(2,2.25)+G\_3(2.25,2.5)+G\_3(2.5,2.75)+G\_3(2.75,3)+G\_3(3,3.25)+G\_3(3.25,3.5)+G\_3(3.5,3.75)+G\_3(3.75,4)+G\_3(4,4.25)+G\_3(4.25,4.5)+G\_3(4.5,4.75)+G\_3(4.75,5)) << endl;      cout << "----------------" << endl;      cout << endl;      cout << "B(f)" << endl;      cout << "----------------" << endl;      cout << "M\t\treal\t\tB(f)\t\terror" << endl;      cout << "----------------" << endl;      cout << 1 << "\t\t" << real << "\t" << B(1,5,5) << "\t" << err(real,B(1,5,5)) << endl;      cout << 2 << "\t\t" << real << "\t" << B(1,3,5)+B(3,5,5) << "\t" << err(real,B(1,3,5)+B(3,5,5)) << endl;      cout << 4 << "\t\t" << real << "\t" << B(1,2,5)+B(2,3,5)+B(3,4,5)+B(4,5,5) << "\t" << err(real,B(1,2,5)+B(2,3,5)+B(3,4,5)+B(4,5,5)) << endl;      cout << 8 << "\t\t" << real << "\t" << B(1,1.5,5)+B(1.5,2,5)+B(2,2.5,5)+B(2.5,3,5)+B(3,3.5,5)+B(3.5,4,5)+B(4,4.5,5)+B(4.5,5,5) << "\t" << err(real,B(1,1.5,5)+B(1.5,2,5)+B(2,2.5,5)+B(2.5,3,5)+B(3,3.5,5)+B(3.5,4,5)+B(4,4.5,5)+B(4.5,5,5)) << endl;      cout << 16 << "\t\t" << real << "\t" << B(1,1.25,5)+B(1.25,1.5,5)+B(1.5,1.75,5)+B(1.75,2,5)+B(2,2.25,5)+B(2.25,2.5,5)+B(2.5,2.75,5)+B(2.75,3,5)+B(3,3.25,5)+B(3.25,3.5,5)+B(3.5,3.75,5)+B(3.75,4,5)+B(4,4.25,5)+B(4.25,4.5,5)+B(4.5,4.75,5)+B(4.75,5,5) << "\t" << err(real,B(1,1.25,5)+B(1.25,1.5,5)+B(1.5,1.75,5)+B(1.75,2,5)+B(2,2.25,5)+B(2.25,2.5,5)+B(2.5,2.75,5)+B(2.75,3,5)+B(3,3.25,5)+B(3.25,3.5,5)+B(3.5,3.75,5)+B(3.75,4,5)+B(4,4.25,5)+B(4.25,4.5,5)+B(4.5,4.75,5)+B(4.75,5,5)) << endl;      cout << "----------------" << endl;      return 0;  }    实验3.1：（数值积分曲线弧长）  #include <iostream>  using namespace std;  #include "func.hpp"  double f(double t) {      double x = 0.5 + 0.3\*t + 3.9\*t\*t - 4.7\*t\*t\*t;      double y = 1.5 + 0.3\*t + 0.9\*t\*t - 2.7\*t\*t\*t;      double dx = 0.3 + 3.9\*2\*t - 4.7\*3\*t\*t;      double dy = 0.3 + 0.9\*2\*t - 2.7\*3\*t\*t;      double result = dx\*dx + dy\*dy;      return sqrt(result);  }  double f\_x(double x) {      return 0;  }  double f\_y(double x) {      return 0;  }  int main() {      double a = 0.0, b = 1.0, n = 5;      double length = B(a, b, n);      cout << "The length of the curve is " << length << endl;      return 0;  }    实验3.2：（数值积分二维翼型的升力系数和阻力系数）  #include <iostream>  #include <fstream>  #include <sstream>  #include <vector>  #include <cstdlib>  using namespace std;  #include "func.hpp"  double f(double x){      return 0;  }  /\* double xx\_1(double t){      ifstream inputFile("cp1.plt"); // 打开文件      if (!inputFile) {          cerr << "无法打开文件。\n";          return 1;      }      string line;      vector<double> X, Y, Cp;      // 跳过头部信息（TITLE和VARIABLES行）      getline(inputFile, line); // TITLE行      getline(inputFile, line); // VARIABLES行      // 读取数据      while (getline(inputFile, line)) {          istringstream iss(line);          double x, y, cp;          if (!(iss >> x >> y >> cp)) {              cerr << "解析数据行出错：" << line << '\n';              return 1;          }          X.push\_back(x);          Y.push\_back(y);          Cp.push\_back(cp);      }      // 关闭文件      inputFile.close();      // 输出数据来验证读取是否正确       std::cout << "读取到的数据点数量： " << X.size() << '\n';      for (size\_t i = 0; i < X.size(); ++i) {          std::cout << X[i] << " " << Y[i] << " " << Cp[i] << '\n';      }       // 使用gnuplot绘制图形        // 首先，将数据写入一个临时文件      std::ofstream tempFile("temp\_data.txt");      for (size\_t i = 0; i < X.size(); ++i) {          tempFile << X[i] << " " << Y[i] << '\n';      }      tempFile.close();      // 调用gnuplot绘图      std::string command = "gnuplot -p -e \"set terminal wxt size 800,600; plot 'temp\_data.txt' with points\" ";      system(command.c\_str());      // 记得删除临时文件      std::remove("temp\_data.txt");      // 绘制Cp-X图      std::ofstream tempFile2("temp\_data2.txt");      for (size\_t i = 0; i < X.size(); ++i) {          tempFile2 << X[i] << " " << Cp[i] << '\n';      }      tempFile2.close();      // 调用gnuplot绘图      std::string command2 = "gnuplot -p -e \"set terminal wxt size 800,600; plot 'temp\_data2.txt' with points\" ";      system(command2.c\_str());      // 记得删除临时文件      std::remove("temp\_data2.txt");      // 计算拟合曲线      int n =  X.size()+1;      double \*x\_now = (double\*)malloc(n \* sizeof(double));      double \*y\_now = (double\*)malloc(n \* sizeof(double));      for (int i = 0; i < n; i++) {          x\_now[i] = i;      }      for (int i = 0; i < (n-1)/2; i++) {          y\_now[i] = X[i];      }      y\_now[n/2] = 0;      for (int i = (n+1)/2; i < n; i++) {          y\_now[i] = X[i-1];      }      double \*\* A = (double\*\*)malloc(n \* sizeof(double\*));      if (A == NULL) {          printf("Memory allocation failed.\n");          exit(1);      }      for(int i=0;i<n;i++) {          A[i] = (double\*)malloc((n+1) \* sizeof(double));          if (A[i] == NULL) {              printf("Memory allocation failed.\n");              exit(1);          }      }      for(int i=0;i<n;i++) {          for(int j=0;j<n+1;j++) {              A[i][j] = 0;          }      }        for (int i = 0; i < n; i++) {          A[i][0] = x\_now[i];          A[i][1] = y\_now[i];      }      for (int j = 2; j < n+1; j++) {          for (int i = j-1; i < n; i++) {                A[i][j] = (A[i][j-1] - A[i-1][j-1]) / (A[i][0] - A[i-(j-1)][0]);            }      }          double result = A[0][1];      for (int i = 1; i < n; i++) {          double sum = 1;          for (int k = 0; k < i;k++){              sum = sum \* (t - A[k][0]);          }          result = result + A[i][i+1]\*sum;      }      //释放内存      for(int i=0;i<n;i++) {          free(A[i]);      }      free(A);      free(x\_now);      free(y\_now);      return result;    } \*/  void calcDerivative(const double\* x, const double\* y, int n, double\* dy\_dx, double\* dx\_dy) {      //通过finite difference method计算导数      for (int i = 0; i < n; i++) {          if (i == 0) {              //边界点处理              dy\_dx[i] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]);              dx\_dy[i] = 1.0 / dy\_dx[i];          } else if (i == n - 1) {              //边界点处理              dy\_dx[i] = (y[i] - y[i - 1]) / (x[i] - x[i - 1]);              dx\_dy[i] = 1.0 / dy\_dx[i];          } else {              //内部点处理              dy\_dx[i] = (y[i + 1] - y[i - 1]) / (x[i + 1] - x[i - 1]);              dx\_dy[i] = (x[i + 1] - x[i - 1]) / (y[i + 1] - y[i - 1]);          }      }  }  void normal(const double\* dy\_dx, const double\* dx\_dy, int n, double\* nx, double\* ny) {      //计算单位法向量      for (int i = 0; i < n/2; i++) {          if (dy\_dx[i] > 0) {              nx[i] = -1;              ny[i] = dx\_dy[i];          } else {              nx[i] = 1;              ny[i] = -dx\_dy[i];          }          double len = sqrt(nx[i] \* nx[i] + ny[i] \* ny[i]);          nx[i] /= len;          ny[i] /= len;      }      for (int i = n/2; i < n; i++) {          if (dy\_dx[i] > 0) {              nx[i] = 1;              ny[i] = -dx\_dy[i];          }else {              nx[i] = -1;              ny[i] = dx\_dy[i];          }          double len = sqrt(nx[i] \* nx[i] + ny[i] \* ny[i]);          nx[i] /= len;          ny[i] /= len;      }  }  double f\_y(double x){      ifstream inputFile("cp1.plt"); // 打开文件      if (!inputFile) {          cerr << "无法打开文件。\n";          return 1;      }      string line;      vector<double> X, Y, Cp;      // 跳过头部信息（TITLE和VARIABLES行）      getline(inputFile, line); // TITLE行      getline(inputFile, line); // VARIABLES行      // 读取数据      while (getline(inputFile, line)) {          istringstream iss(line);          double x, y, cp;          if (!(iss >> x >> y >> cp)) {              cerr << "解析数据行出错：" << line << '\n';              return 1;          }          X.push\_back(x);          Y.push\_back(y);          Cp.push\_back(cp);      }      // 关闭文件      inputFile.close();      int n = X.size();      /\* // 输出数据来验证读取是否正确      std::cout << "读取到的数据点数量： " << X.size() << '\n';      for (size\_t i = 0; i < X.size(); ++i) {          std::cout << X[i] << " " << Y[i] << " " << Cp[i] << '\n';      } \*/          double\* dy\_dx = (double\*)malloc(n \* sizeof(double));      double\* dx\_dy = (double\*)malloc(n \* sizeof(double));        calcDerivative(X.data(), Y.data(), n, dy\_dx, dx\_dy);      /\* for (int i = 0; i < n; i++) {          std::cout << X[i] << " " << Y[i] << " " <<  "dy\_dx[" << i << "] = " << dy\_dx[i] << ' ';          std::cout << "dx\_dy[" << i << "] = " << dx\_dy[i] << '\n';      } \*/      /\*   // 使用gnuplot绘制图形          // 首先，将数据写入一个临时文件      std::ofstream tempFile("temp\_data.txt");      for (size\_t i = 0; i < X.size(); ++i) {          tempFile << X[i] << " " << Y[i] << '\n';      }      tempFile.close();      // 调用gnuplot绘图      std::string command = "gnuplot -p -e \"set terminal wxt size 800,600; plot 'temp\_data.txt' with points\" ";      system(command.c\_str());      // 记得删除临时文件      std::remove("temp\_data.txt");        // 首先，将数据写入一个临时文件      std::ofstream tempFile2("temp\_data2.txt");      for (size\_t i = 0; i < X.size(); ++i) {          tempFile2 << X[i] << " " << Cp[i] << '\n';      }      tempFile2.close();      // 调用gnuplot绘图      std::string command2 = "gnuplot -p -e \"set terminal wxt size 800,600; plot 'temp\_data2.txt' with points\" ";      system(command2.c\_str());      // 记得删除临时文件      std::remove("temp\_data2.txt"); \*/      double \* nx = (double\*)malloc(n \* sizeof(double));      double \* ny = (double\*)malloc(n \* sizeof(double));      normal(dy\_dx, dx\_dy, n, nx, ny);     /\*  for (int i = 0; i < n; i++) {          std::cout << "nx[" << i << "] = " << nx[i] << " ny[" << i << "] = " << ny[i] << '\n';      } \*/      int i = (int)x;      double result = Cp[i] \* ny[i] \* sqrt(1 + dy\_dx[i] \* dy\_dx[i]);      //释放内存      free(dy\_dx);      free(dx\_dy);      free(nx);      free(ny);        return result;  }  double f\_x(double x){      ifstream inputFile("cp1.plt"); // 打开文件      if (!inputFile) {          cerr << "无法打开文件。\n";          return 1;      }      string line;      vector<double> X, Y, Cp;      // 跳过头部信息（TITLE和VARIABLES行）      getline(inputFile, line); // TITLE行      getline(inputFile, line); // VARIABLES行      // 读取数据      while (getline(inputFile, line)) {          istringstream iss(line);          double x, y, cp;          if (!(iss >> x >> y >> cp)) {              cerr << "解析数据行出错：" << line << '\n';              return 1;          }          X.push\_back(x);          Y.push\_back(y);          Cp.push\_back(cp);      }      // 关闭文件      inputFile.close();      int n = X.size();      /\* // 输出数据来验证读取是否正确      std::cout << "读取到的数据点数量： " << X.size() << '\n';      for (size\_t i = 0; i < X.size(); ++i) {          std::cout << X[i] << " " << Y[i] << " " << Cp[i] << '\n';      } \*/          double\* dy\_dx = (double\*)malloc(n \* sizeof(double));      double\* dx\_dy = (double\*)malloc(n \* sizeof(double));        calcDerivative(X.data(), Y.data(), n, dy\_dx, dx\_dy);      /\* for (int i = 0; i < n; i++) {          std::cout << X[i] << " " << Y[i] << " " <<  "dy\_dx[" << i << "] = " << dy\_dx[i] << ' ';          std::cout << "dx\_dy[" << i << "] = " << dx\_dy[i] << '\n';      } \*/      /\*   // 使用gnuplot绘制图形          // 首先，将数据写入一个临时文件      std::ofstream tempFile("temp\_data.txt");      for (size\_t i = 0; i < X.size(); ++i) {          tempFile << X[i] << " " << Y[i] << '\n';      }      tempFile.close();      // 调用gnuplot绘图      std::string command = "gnuplot -p -e \"set terminal wxt size 800,600; plot 'temp\_data.txt' with points\" ";      system(command.c\_str());      // 记得删除临时文件      std::remove("temp\_data.txt");        // 首先，将数据写入一个临时文件      std::ofstream tempFile2("temp\_data2.txt");      for (size\_t i = 0; i < X.size(); ++i) {          tempFile2 << X[i] << " " << Cp[i] << '\n';      }      tempFile2.close();      // 调用gnuplot绘图      std::string command2 = "gnuplot -p -e \"set terminal wxt size 800,600; plot 'temp\_data2.txt' with points\" ";      system(command2.c\_str());      // 记得删除临时文件      std::remove("temp\_data2.txt"); \*/      double \* nx = (double\*)malloc(n \* sizeof(double));      double \* ny = (double\*)malloc(n \* sizeof(double));      normal(dy\_dx, dx\_dy, n, nx, ny);      /\* for (int i = 0; i < n; i++) {          std::cout << "nx[" << i << "] = " << nx[i] << " ny[" << i << "] = " << ny[i] << '\n';      } \*/      int i = (int)x;      double result = Cp[i] \* nx[i] \* sqrt(1 + dy\_dx[i] \* dy\_dx[i]);      //释放内存      free(dy\_dx);      free(dx\_dy);      free(nx);      free(ny);        return result;  }  int main() {      ifstream inputFile("cp1.plt"); // 打开文件      if (!inputFile) {          cerr << "无法打开文件。\n";          return 1;      }      string line;      vector<double> X, Y, Cp;      // 跳过头部信息（TITLE和VARIABLES行）      getline(inputFile, line); // TITLE行      getline(inputFile, line); // VARIABLES行      // 读取数据      while (getline(inputFile, line)) {          istringstream iss(line);          double x, y, cp;          if (!(iss >> x >> y >> cp)) {              cerr << "解析数据行出错：" << line << '\n';              return 1;          }          X.push\_back(x);          Y.push\_back(y);          Cp.push\_back(cp);      }      // 关闭文件      inputFile.close();      int n = X.size();      // 输出数据来验证读取是否正确      std::cout << "读取到的数据点数量： " << X.size() << '\n';      /\* for (size\_t i = 0; i < X.size(); ++i) {          std::cout << X[i] << " " << Y[i] << " " << Cp[i] << '\n';      } \*/        // 使用gnuplot绘制图形          // 首先，将数据写入一个临时文件      std::ofstream tempFile("temp\_data.txt");      for (size\_t i = 0; i < X.size(); ++i) {          tempFile << X[i] << " " << Y[i] << '\n';      }      tempFile.close();      // 调用gnuplot绘图      std::string command = "gnuplot -p -e \"set terminal wxt size 800,600; plot 'temp\_data.txt' with points\" ";      system(command.c\_str());      // 记得删除临时文件      std::remove("temp\_data.txt");        // 首先，将数据写入一个临时文件      std::ofstream tempFile2("temp\_data2.txt");      for (size\_t i = 0; i < X.size(); ++i) {          tempFile2 << X[i] << " " << Cp[i] << '\n';      }      tempFile2.close();      // 调用gnuplot绘图      std::string command2 = "gnuplot -p -e \"set terminal wxt size 800,600; plot 'temp\_data2.txt' with points\" ";      system(command2.c\_str());      // 记得删除临时文件      std::remove("temp\_data2.txt");      double a = 0.0;      double b = 63.0;        double F\_x = 0.0;      double F\_y = 0.0;      for (double x = a; x < b; x += 3.0) {          F\_x += S\_3\_8\_x(x,x+3.0,4);          F\_y += S\_3\_8\_y(x,x+3.0,4);      }      std::cout << "F\_x = " << F\_x << '\n';      std::cout << "F\_y = " << F\_y << '\n';      return 0;  } | | | |
| 实验分析与总结：  经过本次实验，了解到了误差产生的原因以及为什么要避免误差，如何避免误差。  强化了编程能力，学会了如何使用远程服务器辅助完成代码的运行。  梯型公式，辛普森公式，辛普森3/8公式，布尔公式的精度越来越高，计算量越来越大  实际应用过程中，辛普森3/8公式较为常用  复化梯形公式的精度小于复化辛普森公式，如果要达到相同的精度，则迭代次数相差很大  如test1\_4中，复化梯形公式迭代九千多次，而复化辛普森公式一百多次  两点高斯勒让德公式和辛普森公式，三点高斯勒让德公式和布尔公式的精度大致在同一量级 | | | |