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：实验六： 插值与最小二乘逼近 | |
| 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： 2024/7/9 | 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；熟悉函数的多项式差值方法（线性和二次的Lagrange差值，牛顿差值），和最小二乘算法（超定线性方程组求解、线性最小二乘逼近、非线性最小二乘逼近）。  【实验内容】  实验1.1：（分别用线性和二次的lagrange差值求解对应点处函数值）    （）  实验1.2\*：（分别用线性和二次的lagrange差值求解x=1.5处函数值）      实验2.1：（分别线性和二次的牛顿差值求解对应点处函数值）    （）  实验2.2：（用二次的牛顿差值求解x=1.5处函数值）    实验3.1：（最小二乘求解超定线性方程组1）    实验3.2：（用线性最小二乘逼近获得下列点值对应函数）    实验4.1：（用非线性最小二乘逼近找出点P(x, y)和常数K，使得P点到四个圆的距离均为K。四个圆的圆心分别为，半径分别为）    实验4.2：（用非线性最小二乘逼近，找出GPS接收器在近地球的位置P(x, y, z)和时间矫正d，此时已知的同步卫星的位置（千米）和度量时间间隔（秒）如下：        ， 。  DPS定位方程  设置初始的向量为。  用做检查：近似位置为  s | | | |
| 实验1.1：（分别用线性和二次的lagrange差值求解对应点处函数值）  #include <stdio.h>  #include <stdlib.h>  double lagrange\_1(double x, double \* x\_now, double \* y\_now, int n) {  //假设x是递增的,并且x不与已知的x\_now重复,且x在x\_now的范围内  int count = 0;  for (int i = 0; i < n; i++) {  if (x>x\_now[i]) {  count++;  }  }  double left\_x = x\_now[count-1];  double right\_x = x\_now[count];  double left\_y = y\_now[count-1];  double right\_y = y\_now[count];  double result = (x - left\_x) \* right\_y / (right\_x - left\_x) + (right\_x - x) \* left\_y / (right\_x - left\_x);  return result;  }  double lagrange\_2(double x, double \* x\_now, double \* y\_now, int n) {  //假设x是递增的,并且x不与已知的x\_now重复,且x在x\_now的范围内  int count = 0;  for (int i = 0; i < n; i++) {  if (x>x\_now[i]) {  count++;  }  }  double x\_left = x\_now[count-1];  double x\_mid = x\_now[count];  double x\_right = x\_now[count+1];  double y\_left = y\_now[count-1];  double y\_mid = y\_now[count];  double y\_right = y\_now[count+1];  double result = y\_left \* ((x - x\_mid)\* (x - x\_right)) / ((x\_left - x\_mid) \* (x\_left - x\_right)) + y\_mid \* ((x - x\_left) \* (x - x\_right)) / ((x\_mid - x\_left) \* (x\_mid - x\_right)) + y\_right \* ((x - x\_left) \* (x - x\_mid)) / ((x\_right - x\_left) \* (x\_right - x\_mid));  return result;  }  int main() {  int n = 3;  double \* x\_now;  double \* y\_now;  x\_now = (double\*)malloc(n \* sizeof(double));  y\_now = (double\*)malloc(n \* sizeof(double));  x\_now[0] = 100;  x\_now[1] = 121;  x\_now[2] = 144;  y\_now[0] = 10;  y\_now[1] = 11;  y\_now[2] = 12;  double x = 115;  double result\_1 = lagrange\_1(x, x\_now, y\_now, n);  double result\_2 = lagrange\_2(x, x\_now, y\_now, n);  printf("linear\t : %lf\n", result\_1);  printf("quadratic: %lf\n", result\_2);  return 0;  }    实验1.2\*：（分别用线性和二次的lagrange差值求解x=1.5处函数值）  #include <stdio.h>  #include <stdlib.h>  double lagrange\_1(double x, double \* x\_now, double \* y\_now, int n) {  //假设x是递增的,并且x不与已知的x\_now重复,且x在x\_now的范围内  int count = 0;  for (int i = 0; i < n; i++) {  if (x>x\_now[i]) {  count++;  }  }  double left\_x = x\_now[count-1];  double right\_x = x\_now[count];  double left\_y = y\_now[count-1];  double right\_y = y\_now[count];  double result = (x - left\_x) \* right\_y / (right\_x - left\_x) + (right\_x - x) \* left\_y / (right\_x - left\_x);  return result;  }  double lagrange\_2(double x, double \* x\_now, double \* y\_now, int n) {  //假设x是递增的,并且x不与已知的x\_now重复,且x在x\_now的范围内  int count = 0;  for (int i = 0; i < n; i++) {  if (x>x\_now[i]) {  count++;  }  }  double x\_left = x\_now[count-1];  double x\_mid = x\_now[count];  double x\_right = x\_now[count+1];  double y\_left = y\_now[count-1];  double y\_mid = y\_now[count];  double y\_right = y\_now[count+1];  double result = y\_left \* ((x - x\_mid)\* (x - x\_right)) / ((x\_left - x\_mid) \* (x\_left - x\_right)) + y\_mid \* ((x - x\_left) \* (x - x\_right)) / ((x\_mid - x\_left) \* (x\_mid - x\_right)) + y\_right \* ((x - x\_left) \* (x - x\_mid)) / ((x\_right - x\_left) \* (x\_right - x\_mid));  return result;  }  int main() {  int n = 3;  double \* x\_now;  double \* y\_now;  x\_now = (double\*)malloc(n \* sizeof(double));  y\_now = (double\*)malloc(n \* sizeof(double));  x\_now[0] = 1;  x\_now[1] = 2;  x\_now[2] = 3;  y\_now[0] = 2;  y\_now[1] = 4;  y\_now[2] = 12;  double x = 1.5;  double result\_1 = lagrange\_1(x, x\_now, y\_now, n);  double result\_2 = lagrange\_2(x, x\_now, y\_now, n);  printf("linear\t : %lf\n", result\_1);  printf("quadratic: %lf\n", result\_2);  return 0;  }    实验2.1：（分别线性和二次的牛顿差值求解对应点处函数值）  #include <stdio.h>  #include <stdlib.h>  double newton\_1(double x, double \* x\_now, double \* y\_now, int n) {  //假设x是递增的,并且x不与已知的x\_now重复,且x在x\_now的范围内    double \*\* A = (double\*\*)malloc((n+1) \* sizeof(double\*));  if (A == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  for(int i=0;i<=n;i++) {  A[i] = (double\*)malloc((n+2) \* 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] + A[1][2] \*(x - x\_now[0]);  //释放内存  for(int i=0;i<=n;i++) {  free(A[i]);  }  free(A);  return result;  }  double newton\_2(double x, double \* x\_now, double \* y\_now, int n) {  //假设x是递增的,并且x不与已知的x\_now重复,且x在x\_now的范围内  double \*\* A = (double\*\*)malloc((n+1) \* sizeof(double\*));  if (A == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  for(int i=0;i<=n;i++) {  A[i] = (double\*)malloc((n+2) \* 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] + A[1][2] \*(x - x\_now[0]) + A[2][3] \*(x - x\_now[0])\*(x - x\_now[1]);  //释放内存  for(int i=0;i<=n;i++) {  free(A[i]);  }  free(A);    return result;  }  int main() {  int n = 3;  double \* x\_now;  double \* y\_now;  x\_now = (double\*)malloc(n \* sizeof(double));  y\_now = (double\*)malloc(n \* sizeof(double));  x\_now[0] = 100;  x\_now[1] = 121;  x\_now[2] = 144;  y\_now[0] = 10;  y\_now[1] = 11;  y\_now[2] = 12;  double x = 115;  double result\_1 = newton\_1(x, x\_now, y\_now, n);  double result\_2 = newton\_2(x, x\_now, y\_now, n);  printf("linear\t : %lf\n", result\_1);  printf("quadratic: %lf\n", result\_2);  return 0;  }    实验2.2：（用二次的牛顿差值求解x=1.5处函数值）  #include <stdio.h>  #include <stdlib.h>  double newton\_1(double x, double \* x\_now, double \* y\_now, int n) {  //假设x是递增的,并且x不与已知的x\_now重复,且x在x\_now的范围内    double \*\* A = (double\*\*)malloc((n+1) \* sizeof(double\*));  if (A == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  for(int i=0;i<=n;i++) {  A[i] = (double\*)malloc((n+2) \* 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] + A[1][2] \*(x - x\_now[0]);  //释放内存  for(int i=0;i<=n;i++) {  free(A[i]);  }  free(A);  return result;  }  double newton\_2(double x, double \* x\_now, double \* y\_now, int n) {  //假设x是递增的,并且x不与已知的x\_now重复,且x在x\_now的范围内  double \*\* A = (double\*\*)malloc((n+1) \* sizeof(double\*));  if (A == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  for(int i=0;i<=n;i++) {  A[i] = (double\*)malloc((n+2) \* 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] + A[1][2] \*(x - x\_now[0]) + A[2][3] \*(x - x\_now[0])\*(x - x\_now[1]);  //释放内存  for(int i=0;i<=n;i++) {  free(A[i]);  }  free(A);    return result;  }  int main() {  int n = 4;  double \* x\_now;  double \* y\_now;  x\_now = (double\*)malloc(n \* sizeof(double));  y\_now = (double\*)malloc(n \* sizeof(double));  x\_now[0] = -2;  x\_now[1] = -1;  x\_now[2] = 1;  x\_now[3] = 2;  y\_now[0] = 5;  y\_now[1] = 3;  y\_now[2] = 17;  y\_now[3] = 21;  double x = 1.5;  double result\_1 = newton\_1(x, x\_now, y\_now, n);  double result\_2 = newton\_2(x, x\_now, y\_now, n);  printf("linear\t : %lf\n", result\_1);  printf("quadratic: %lf\n", result\_2);  return 0;  }    实验3.1：（最小二乘求解超定线性方程组1）  //求解超定方程组Ax=b  //其中 计算时用到了列主元高斯消元法  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  void gauss\_elimination(int n, double \*\*A\_tA, double \*A\_tb, double \*x) {  for (int i = 0; i < n; i++) {  int max\_index = i;  for (int j = i + 1; j < n; j++) {//找列最大值  if (fabs(A\_tA[j][i]) > fabs(A\_tA[max\_index][i])) {  max\_index = j;  }  }  if (A\_tA[max\_index][i] == 0) {//如果最大值为0，则矩阵为奇异矩阵  printf("The matrix is singular.\n");  exit(0);  }  if (max\_index != i) {//交换两行  for(int j = i; j < n; j++) {  double temp = A\_tA[i][j];  A\_tA[i][j] = A\_tA[max\_index][j];  A\_tA[max\_index][j] = temp;  }  double temp\_b = A\_tb[i];  A\_tb[i] = A\_tb[max\_index];  A\_tb[max\_index] = temp\_b;  }  for (int j = i + 1; j < n; j++) {//消去法  double factor = A\_tA[j][i] / A\_tA[i][i];  A\_tA[j][i] = 0;  for (int k = i + 1; k < n; k++) {  A\_tA[j][k] -= factor \* A\_tA[i][k];  }  A\_tb[j] -= factor \* A\_tb[i];  }  }        // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  double ad = A\_tb[i];  for (int j = n-1; j >= i+1; j--) {  ad -= A\_tA[i][j] \* x[j];  }  x[i] = ad/A\_tA[i][i];  }  }  void result(int n, int num, double \*\*A, double \*b, double \*x) {  // 输出矩阵A和b  printf("A = \n");  for (int i = 0; i < num; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A[i][j]);  }  printf("\n");  }  printf("b = \n");  for (int i = 0; i < num; i++) {  printf("%lf\n", b[i]);  }  printf("\n");  //分配内存  double \*\*A\_tA = (double \*\*)malloc(n \* sizeof(double \*));  if (A\_tA == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  for (int i = 0; i < n; i++) {  A\_tA[i] = (double \*)malloc(n \* sizeof(double));  if (A\_tA[i] == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  }  double \*A\_tb = (double \*)malloc(n \* sizeof(double));  if (A\_tb == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  //计算A\_tA和A\_tb  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  A\_tA[i][j] = 0;  }  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  for (int k = 0; k < num; k++) {  A\_tA[i][j] += A[k][i] \* A[k][j];  }  }  }  for (int i = 0; i < n; i++) {  A\_tb[i] = 0;  }  for (int i = 0; i < n; i++) {  for (int k = 0; k < num; k++) {  A\_tb[i] += A[k][i] \* b[k];  }  }  //输出A\_tA和A\_tb  printf("A\_tA = \n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A\_tA[i][j]);  }  printf("\n");  }  printf("\n");    printf("A\_tb = \n");  for (int i = 0; i < n; i++) {  printf("%lf\n", A\_tb[i]);  }  printf("\n");  //调用高斯消元法求解线性方程组Ax=b  gauss\_elimination(n, A\_tA, A\_tb, x);  //释放内存  for (int i = 0; i < n; i++) {  free(A\_tA[i]);  }  free(A\_tA);  free(A\_tb);  }  int main(void) {    int n = 3;//列数  int num = 4; //行数  //分配内存  double \*\* A = (double \*\*)malloc(num \* sizeof(double \*));  if (A == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  for (int i = 0; i < num; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  if (A[i] == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  }  double \* b = (double \*)malloc(num \* sizeof(double));  if (b == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  A[0][0] = 1; A[0][1] = 2; A[0][2] = 4;  A[1][0] = 2; A[1][1] = 1; A[1][2] = 1;  A[2][0] = 1; A[2][1] = 1; A[2][2] = 2;  A[3][0] = 1; A[3][1] = -1; A[3][2] = -2;    b[0] = -1;  b[1] = 4;  b[2] = 2;  b[3] = 1;  double\* x = (double\*)malloc(n \* sizeof(double));  if (x == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  result(n, num, A, b, x);  // 输出解  printf("The solution is:\n");  for(int i = 0; i < n; i++) {  printf("%lf\n", x[i]);  }  printf("\n");  // 释放内存  for (int i = 0; i < num; i++) {  free(A[i]);  }  free(A);  free(b);  free(x);  return 0;  }    实验3.2：（用线性最小二乘逼近获得下列点值对应函数）  // 线性拟合五个点 输出y = ax + b中的a,b  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  void result(double \*x\_i, double \*y\_i, int num, int n, double \*a, double \*b) {  double \*\* A = (double \*\*)malloc(num \* sizeof(double \*));  //分配内存  if (A == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  for (int i = 0; i < num; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  if (A[i] == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  }  double \* B = (double \*)malloc(num \* sizeof(double));  if (B == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  // 构造矩阵A和b  for (int i = 0; i < num; i++) {  for (int j = 0; j < n; j++) {  if (j == 0) {  A[i][j] = 1;  } else if (j == 1) {  A[i][j] = x\_i[i];  } else {  A[i][j] = 0;  }  }  B[i] = y\_i[i];  }  printf("A = \n");  for (int i = 0; i < num; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A[i][j]);  }  printf("\n");  }  printf("b = \n");  for (int i = 0; i < num; i++) {  printf("%lf\n", B[i]);  }  printf("\n");  double \*\*A\_tA = (double \*\*)malloc(n \* sizeof(double \*));  if (A\_tA == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  for (int i = 0; i < n; i++) {  A\_tA[i] = (double \*)malloc(n \* sizeof(double));  if (A\_tA[i] == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {    A\_tA[i][j] = 0;  }  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {    for (int k = 0; k < num; k++) {  A\_tA[i][j] += A[k][i] \* A[k][j];  }  }  }  printf("A\_tA = \n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A\_tA[i][j]);  }  printf("\n");  }  double \*A\_tb = (double \*)malloc(n \* sizeof(double));  if (A\_tb == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  for (int i = 0; i < n; i++) {  A\_tb[i] = 0;    }  for (int i = 0; i < n; i++) {    for (int k = 0; k < num; k++) {  A\_tb[i] += A[k][i] \* B[k];  }  }    printf("A\_tb = \n");  for (int i = 0; i < n; i++) {  printf("%lf\n", A\_tb[i]);  }  printf("\n");  for (int i = 0; i < n; i++) {  int max\_index = i;  for (int j = i + 1; j < n; j++) {//找列最大值  if (fabs(A\_tA[j][i]) > fabs(A\_tA[max\_index][i])) {  max\_index = j;  }  }  if (A\_tA[max\_index][i] == 0) {//如果最大值为0，则矩阵为奇异矩阵  printf("The matrix is singular.\n");  exit(0);  }  if (max\_index != i) {//交换两行  for(int j = i; j < n; j++) {  double temp = A\_tA[i][j];  A\_tA[i][j] = A\_tA[max\_index][j];  A\_tA[max\_index][j] = temp;  }  double temp\_b = A\_tb[i];  A\_tb[i] = A\_tb[max\_index];  A\_tb[max\_index] = temp\_b;  }  for (int j = i + 1; j < n; j++) {//消去法  double factor = A\_tA[j][i] / A\_tA[i][i];  A\_tA[j][i] = 0;  for (int k = i + 1; k < n; k++) {  A\_tA[j][k] -= factor \* A\_tA[i][k];  }  A\_tb[j] -= factor \* A\_tb[i];  }  }      double\* x = (double\*)malloc(n \* sizeof(double));  // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  double ad = A\_tb[i];  for (int j = n-1; j >= i+1; j--) {  ad -= A\_tA[i][j] \* x[j];  }  x[i] = ad/A\_tA[i][i];  }  // 输出解  printf("The solution is:\n");  for(int i = 0; i < n; i++) {  printf("%lf\n", x[i]);  }  printf("\n");    \*a = x[1];  \*b = x[0];      }  // 线性拟合五个点 输出y = ax + b中的a,b  int main(void) {  //列数  int n = 2;  //点数  int num = 5;  double \* x\_i = (double \*)malloc(num \* sizeof(double));  if (x\_i == NULL) {  printf("Memory allocation failed.\n");  return -1;  }    double \* y\_i = (double \*)malloc(num \* sizeof(double));  if (y\_i == NULL) {  printf("Memory allocation failed.\n");  return -1;  }  x\_i[0] = 25; x\_i[1] = 27; x\_i[2] = 31; x\_i[3] = 33; x\_i[4] = 35;  y\_i[0] = 110; y\_i[1] = 115; y\_i[2] = 155; y\_i[3] = 160; y\_i[4] = 180;  double a = 0, b = 0;  result(x\_i, y\_i, num, n, &a, &b);  printf("a = %lf\n", a);  printf("b = %lf\n", b);  // free memory  free(x\_i);  free(y\_i);  return 0;  }    实验4.1：（用非线性最小二乘逼近找出点P(x, y)和常数K，使得P点到四个圆的距离均为K。四个圆的圆心分别为，半径分别为）  //求解超定方程组Ax=b  //其中 计算时用到了列主元高斯消元法  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  void gauss\_elimination(int n, double \*\*A\_tA, double \*A\_tb, double \*x) {  for (int i = 0; i < n; i++) {  int max\_index = i;  for (int j = i + 1; j < n; j++) {//找列最大值  if (fabs(A\_tA[j][i]) > fabs(A\_tA[max\_index][i])) {  max\_index = j;  }  }  if (A\_tA[max\_index][i] == 0) {//如果最大值为0，则矩阵为奇异矩阵  printf("The matrix is singular.\n");  exit(0);  }  if (max\_index != i) {//交换两行  for(int j = i; j < n; j++) {  double temp = A\_tA[i][j];  A\_tA[i][j] = A\_tA[max\_index][j];  A\_tA[max\_index][j] = temp;  }  double temp\_b = A\_tb[i];  A\_tb[i] = A\_tb[max\_index];  A\_tb[max\_index] = temp\_b;  }  for (int j = i + 1; j < n; j++) {//消去法  double factor = A\_tA[j][i] / A\_tA[i][i];  A\_tA[j][i] = 0;  for (int k = i + 1; k < n; k++) {  A\_tA[j][k] -= factor \* A\_tA[i][k];  }  A\_tb[j] -= factor \* A\_tb[i];  }  }    // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  double ad = A\_tb[i];  for (int j = n-1; j >= i+1; j--) {  ad -= A\_tA[i][j] \* x[j];  }    x[i] = ad/A\_tA[i][i];  }  }  void result(int n, int num, double \*\*A, double \*b, double \*x) {  /\* // 输出矩阵A和b  printf("A = \n");  for (int i = 0; i < num; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A[i][j]);  }  printf("\n");  }  printf("b = \n");  for (int i = 0; i < num; i++) {  printf("%lf\n", b[i]);  }  printf("\n"); \*/  //分配内存  double \*\*A\_tA = (double \*\*)malloc(n \* sizeof(double \*));  if (A\_tA == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  for (int i = 0; i < n; i++) {  A\_tA[i] = (double \*)malloc(n \* sizeof(double));  if (A\_tA[i] == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  }  double \*A\_tb = (double \*)malloc(n \* sizeof(double));  if (A\_tb == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  //计算A\_tA和A\_tb  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  A\_tA[i][j] = 0;  }  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  for (int k = 0; k < num; k++) {  A\_tA[i][j] += A[k][i] \* A[k][j];  }  }  }  for (int i = 0; i < n; i++) {  A\_tb[i] = 0;  }  for (int i = 0; i < n; i++) {  for (int k = 0; k < num; k++) {  A\_tb[i] += A[k][i] \* b[k];  }  }  /\* //输出A\_tA和A\_tb  printf("A\_tA = \n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A\_tA[i][j]);  }  printf("\n");  }  printf("\n");    printf("A\_tb = \n");  for (int i = 0; i < n; i++) {  printf("%lf\n", A\_tb[i]);  }  printf("\n"); \*/  //调用高斯消元法求解线性方程组Ax=b  gauss\_elimination(n, A\_tA, A\_tb, x);    //释放内存  for (int i = 0; i < n; i++) {  free(A\_tA[i]);  }  free(A\_tA);  free(A\_tb);  }  double S2(double x,double y,double x1,double y1) {  return sqrt((x-x1)\*(x-x1) + (y-y1)\*(y-y1));  }  int main(void) {    int n = 3;//列数//维数+1  int num = 4; //行数/点数  double eps = 1e-5; //误差  double k = 1;//系数  int max\_iter = 100; //最大迭代次数  int iter = 0; //迭代次数  //分配内存  double \*\* A = (double \*\*)malloc(num \* sizeof(double \*));  if (A == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  for (int i = 0; i < num; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  if (A[i] == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  }  double \* b = (double \*)malloc(num \* sizeof(double));  if (b == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  double\* x = (double\*)malloc(num \* sizeof(double));  if (x == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  double\* y = (double\*)malloc(num \* sizeof(double));  if (y == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  double\* r = (double\*)malloc(num \* sizeof(double));  if (r == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  double\* x\_result = (double\*)malloc(n \* sizeof(double));  if (x\_result == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  x[0] = -1; x[1] = 1; x[2] = 1; x[3] = 0;  y[0] = 0; y[1] = 0.5; y[2] = -0.5; y[3] = 1;  r[0] = 1; r[1] = 0.5; r[2] = 0.5; r[3] = 0.5;  // 计算A和b  double\* s = (double\*)malloc(num \* sizeof(double));  if (s == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  x\_result[0] = 0; x\_result[1] = 0; x\_result[2] = 0;    double result\_1 = sqrt((x\_result[0]-x[0])\*(x\_result[0]-x[0]) + (x\_result[1]-y[0])\*(x\_result[1]-y[0])) + sqrt((x\_result[0]-x[1])\*(x\_result[0]-x[1]) + (x\_result[1]-y[1])\*(x\_result[1]-y[1])) + sqrt((x\_result[0]-x[2])\*(x\_result[0]-x[2]) + (x\_result[1]-y[2])\*(x\_result[1]-y[2])) + sqrt((x\_result[0]-x[3])\*(x\_result[0]-x[3]) + (x\_result[1]-y[3])\*(x\_result[1]-y[3]))-r[0]-r[1]-r[2]-r[3]-4\*x\_result[2];  printf("\n");    printf("----------------------------------------------------\n");  printf("iter\tx\t\ty\t\tK\t\terror\n");  printf("----------------------------------------------------\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\n", iter, x\_result[0], x\_result[1], x\_result[2], result\_1);  iter++;    while (iter < max\_iter && fabs(result\_1) > eps) {  for (int i = 0; i < num; i++) {  //printf("xi = %lf, yi = %lf, ri = %lf\n", x[i], y[i], r[i]);  s[i] = S2(x\_result[0],x\_result[1],x[i],y[i]);  }  //printf("s1 = %lf, s2 = %lf, s3 = %lf, s4 = %lf \n", s[0], s[1], s[2], s[3]);  for (int i = 0; i < num; i++) {  for (int j = 0; j < n; j++) {  if (j == 0) {  A[i][j] = (x\_result[j] - x[i])/s[i];  } else if (j == 1) {  A[i][j] = (x\_result[j] - y[i])/s[i];  } else {  A[i][j] = -1\*k;  }  }  }    for (int i = 0; i < num; i++) {  b[i] = -1\*(s[i] - (r[i] + x\_result[2]));  }  double\* v = (double\*)malloc(n \* sizeof(double));  if (v == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  result(n, num, A, b, v);    for (int i = 0; i < n; i++) {  x\_result[i] += v[i];  }  result\_1 = sqrt((x\_result[0]-x[0])\*(x\_result[0]-x[0]) + (x\_result[1]-y[0])\*(x\_result[1]-y[0])) + sqrt((x\_result[0]-x[1])\*(x\_result[0]-x[1]) + (x\_result[1]-y[1])\*(x\_result[1]-y[1])) + sqrt((x\_result[0]-x[2])\*(x\_result[0]-x[2]) + (x\_result[1]-y[2])\*(x\_result[1]-y[2])) + sqrt((x\_result[0]-x[3])\*(x\_result[0]-x[3]) + (x\_result[1]-y[3])\*(x\_result[1]-y[3]))-r[0]-r[1]-r[2]-r[3]-4\*x\_result[2];  printf("%d\t%lf\t%lf\t%lf\t%lf\n", iter, x\_result[0], x\_result[1], x\_result[2], result\_1);  iter++;  }  // 输出解  printf("\n");  printf("The solution is:\n");  for(int i = 0; i < n-1; i++) {  printf("x%d = %lf\n",i+1, x\_result[i]);  }  printf("K = %lf\n", x\_result[n-1]);  printf("\n");  // 释放内存  for (int i = 0; i < num; i++) {  free(A[i]);  }  free(A);  free(b);  free(x);  free(y);  free(r);  free(x\_result);  free(s);  return 0;  }    实验4.2：（用非线性最小二乘逼近，找出GPS接收器在近地球的位置P(x, y, z)和时间矫正d  //求解超定方程组Ax=b  //其中 计算时用到了列主元高斯消元法  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  void gauss\_elimination(int n, double \*\*A\_tA, double \*A\_tb, double \*x) {  for (int i = 0; i < n; i++) {  int max\_index = i;  for (int j = i + 1; j < n; j++) {//找列最大值  if (fabs(A\_tA[j][i]) > fabs(A\_tA[max\_index][i])) {  max\_index = j;  }  }  if (A\_tA[max\_index][i] == 0) {//如果最大值为0，则矩阵为奇异矩阵  printf("The matrix is singular.\n");  exit(0);  }  if (max\_index != i) {//交换两行  for(int j = i; j < n; j++) {  double temp = A\_tA[i][j];  A\_tA[i][j] = A\_tA[max\_index][j];  A\_tA[max\_index][j] = temp;  }  double temp\_b = A\_tb[i];  A\_tb[i] = A\_tb[max\_index];  A\_tb[max\_index] = temp\_b;  }  for (int j = i + 1; j < n; j++) {//消去法  double factor = A\_tA[j][i] / A\_tA[i][i];  A\_tA[j][i] = 0;  for (int k = i + 1; k < n; k++) {  A\_tA[j][k] -= factor \* A\_tA[i][k];  }  A\_tb[j] -= factor \* A\_tb[i];  }  }    // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  double ad = A\_tb[i];  for (int j = n-1; j >= i+1; j--) {  ad -= A\_tA[i][j] \* x[j];  }    x[i] = ad/A\_tA[i][i];  }  }  void result(int n, int num, double \*\*A, double \*b, double \*x) {  /\* // 输出矩阵A和b  printf("A = \n");  for (int i = 0; i < num; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A[i][j]);  }  printf("\n");  }  printf("b = \n");  for (int i = 0; i < num; i++) {  printf("%lf\n", b[i]);  }  printf("\n"); \*/  //分配内存  double \*\*A\_tA = (double \*\*)malloc(n \* sizeof(double \*));  if (A\_tA == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  for (int i = 0; i < n; i++) {  A\_tA[i] = (double \*)malloc(n \* sizeof(double));  if (A\_tA[i] == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  }  double \*A\_tb = (double \*)malloc(n \* sizeof(double));  if (A\_tb == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  //计算A\_tA和A\_tb  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  A\_tA[i][j] = 0;  }  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  for (int k = 0; k < num; k++) {  A\_tA[i][j] += A[k][i] \* A[k][j];  }  }  }  for (int i = 0; i < n; i++) {  A\_tb[i] = 0;  }  for (int i = 0; i < n; i++) {  for (int k = 0; k < num; k++) {  A\_tb[i] += A[k][i] \* b[k];  }  }  /\* //输出A\_tA和A\_tb  printf("A\_tA = \n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%lf ", A\_tA[i][j]);  }  printf("\n");  }  printf("\n");    printf("A\_tb = \n");  for (int i = 0; i < n; i++) {  printf("%lf\n", A\_tb[i]);  }  printf("\n"); \*/  //调用高斯消元法求解线性方程组Ax=b  gauss\_elimination(n, A\_tA, A\_tb, x);    //释放内存  for (int i = 0; i < n; i++) {  free(A\_tA[i]);  }  free(A\_tA);  free(A\_tb);  }  double S2(double x,double y,double x1,double y1) {  return sqrt((x-x1)\*(x-x1) + (y-y1)\*(y-y1));  }  double S3(double x, double y, double z, double x1, double y1, double z1) {  return sqrt((x-x1)\*(x-x1) + (y-y1)\*(y-y1) + (z-z1)\*(z-z1));  }  int main(void) {    int n = 4;//列数//问题维数+1  int num = 4; //行数/点数  double eps = 1e-5; //误差  double k = 299792.50;//系数  int max\_iter = 100; //最大迭代次数  int iter = 0; //迭代次数  //分配内存  double \*\* A = (double \*\*)malloc(num \* sizeof(double \*));  if (A == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  for (int i = 0; i < num; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  if (A[i] == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  }  double \* b = (double \*)malloc(num \* sizeof(double));  if (b == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  double\* x = (double\*)malloc(num \* sizeof(double));  if (x == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  double\* y = (double\*)malloc(num \* sizeof(double));  if (y == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  double\* z = (double\*)malloc(num \* sizeof(double));  if (z == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  double\* r = (double\*)malloc(num \* sizeof(double));  if (r == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  double\* x\_result = (double\*)malloc(n \* sizeof(double));  if (x\_result == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  x[0] = 15600; x[1] = 18760; x[2] = 17610; x[3] = 19170;  y[0] = 7540; y[1] = 2750; y[2] = 14630; y[3] = 610;  z[0] = 20140; z[1] = 18610; z[2] = 13480; z[3] = 18390;  r[0] = 0.07074; r[1] = 0.07220; r[2] = 0.07690; r[3] = 0.07242;  // 计算A和b  double\* s = (double\*)malloc(num \* sizeof(double));  if (s == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  x\_result[0] = 0; x\_result[1] = 0; x\_result[2] = 6370; x\_result[3] = 0;  for (int i = 0; i < num; i++) {  //printf("xi = %lf, yi = %lf,zi = %lf, ri = %lf\n", x[i], y[i],z[i], r[i]);  s[i] = S3(x\_result[0], x\_result[1], x\_result[2], x[i], y[i], z[i]);  }    double result\_3 =s[0]+ s[1] +s[2] +s[3] +num\*k\*x\_result[3] - k\*r[0] - k\*r[1] - k\*r[2] - k\*r[3];  printf("\n");    printf("----------------------------------------------------\n");  printf("iter\tx\t\ty\t\tz\t\tK\t\terror\n");  printf("----------------------------------------------------\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", iter, x\_result[0], x\_result[1], x\_result[2], x\_result[3], result\_3);  iter++;    while (iter < max\_iter && fabs(result\_3) >= eps) {  for (int i = 0; i < num; i++) {  //printf("xi = %lf, yi = %lf,zi = %lf, ri = %lf\n", x[i], y[i],z[i], r[i]);  s[i] = S3(x\_result[0], x\_result[1], x\_result[2], x[i], y[i], z[i]);  }  //printf("s1 = %lf, s2 = %lf, s3 = %lf, s4 = %lf \n", s[0], s[1], s[2], s[3]);  for (int i = 0; i < num; i++) {  for (int j = 0; j < n; j++) {  if (j == 0) {  A[i][j] = (x\_result[j] - x[i])/s[i];  } else if (j == 1) {  A[i][j] = (x\_result[j] - y[i])/s[i];  } else if (j == 2) {  A[i][j] = (x\_result[j] - z[i])/s[i];  } else {  A[i][j] = k;  }  }  }    for (int i = 0; i < num; i++) {  b[i] = -1\*(s[i] - k\*(r[i] - x\_result[3]));  }  double\* v = (double\*)malloc(n \* sizeof(double));  if (v == NULL) {  printf("Memory allocation failed.\n");  exit(0);  }  result(n, num, A, b, v);    for (int i = 0; i < n; i++) {  x\_result[i] += v[i];  }  result\_3 =s[0]+ s[1] +s[2] +s[3] +num\*k\*x\_result[3] - k\*r[0] - k\*r[1] - k\*r[2] - k\*r[3];  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", iter, x\_result[0], x\_result[1], x\_result[2], x\_result[3], result\_3);  iter++;  }  // 输出解  printf("\n");  printf("The solution is:\n");  for(int i = 0; i < n-1; i++) {  printf("x%d = %lf\n",i+1, x\_result[i]);  }  printf("K = %lf\n", x\_result[n-1]);  printf("\n");  // 释放内存  for (int i = 0; i < num; i++) {  free(A[i]);  }  free(A);  free(b);  free(x);  free(y);  free(r);  free(x\_result);  free(s);  return 0;  }    插值法求样例：  #include <stdio.h>  #include <stdlib.h>  double f(double x) {  int n = 7;  double \* x\_now;  double \* y\_now;  x\_now = (double\*)malloc(n \* sizeof(double));  y\_now = (double\*)malloc(n \* sizeof(double));  x\_now[0] = -0.1;  x\_now[1] = 0;  x\_now[2] = 1;  x\_now[3] = 4;  x\_now[4] = 5;  x\_now[5] = 5;  x\_now[6] = 6;  y\_now[0] = 0;  y\_now[1] = -8;  y\_now[2] = 0;  y\_now[3] = 6;  y\_now[4] = 1;  y\_now[5] = 1;  y\_now[6] = 4;  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++) {  if (i == 5 && j == 2){  A[i][j] = 0;  } else{  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 \* (x - 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;  }  double df(double x) {  double eps = 0.000001;  return (f(x+eps) - f(x-eps))/(2\*eps);  }  double ddf(double x) {  double eps = 0.000001;  return (df(x+eps) - df(x-eps))/(2\*eps);  }  int main() {  double x1 = -0.1;  double x2 = 0;  double x3 = 1;  double x4 = 4;  double x5 = 5;  double x6 = 5;  double x7 = 6;    double y1 = f(x1);  double y2 = f(x2);  double y3 = f(x3);  double y4 = f(x4);  double y5 = f(x5);  double y6 = f(x6);  double y7 = f(x7);  printf("x1 = %lf\ty1 = %lf\n", x1, y1);  printf("x2 = %lf\ty2 = %lf\n", x2, y2);  printf("x3 = %lf\ty3 = %lf\n", x3, y3);  printf("x4 = %lf\ty4 = %lf\n", x4, y4);  printf("x5 = %lf\ty5 = %lf\n", x5, y5);  printf("x6 = %lf\ty6 = %lf\n", x6, y6);  printf("x7 = %lf\ty7 = %lf\n", x7, y7);  printf("\n");  double eps = 0.000001;  double x = 5;  double dy = (f(x+eps) - f(x-eps))/(2\*eps);  printf("dy = %lf\n", dy);  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。  通过实验对比；二次插值比一次插值在预测位置函数在某点的函数值时表现较好  牛顿插值比拉格朗日插值好用在：当在给出更多新的关于未知函数的数据时，不需要再重新进行计算，直接将计算结果加在原结果之后即可，且再给定某点导数信息时，牛顿法也可以进行有效处理  超定线性方程组的求解可以用左乘系数矩阵A的转置A'的方法将其转换为非超定问题，后迭代求解 | | | |