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：实验五： 优化问题的迭代求解 | |
| 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；熟悉求解一维优化问题的区间逼近法（黄金分割搜索、逐次抛物插值搜索）和梯度类方法（牛顿法）；熟悉多维优化问题中无约束优化问题的一般解法（最速下降法、共轭梯度法、牛顿法）；了解以上方法的稳定性与收敛速度特点；了解这些传统优化算法在多极值问题中的局限性。  【实验内容】  实验1.1：（优化问题1）  用黄金分割搜索，求函数 在区间[0,1]上的极小值。  实验1.2\*：（优化问题2）  用黄金分割搜索，求函数在区间 [0,1.2]上的极小值。  实验2.1：（优化问题1）  用逐次抛物插值搜索，求函数 在区间[0,1]上的极小值。  实验2.2：（优化问题2）  用逐次抛物插值搜索求函数在区间 [0,1.2]上的极小值。  实验3.1：（优化问题1）  用牛顿法，求函数 在区间[0,1]上的极小值。  实验3.2：（优化问题2）  用牛顿法求函数在区间 [0,2]上的极小值。  实验4.1：（优化问题3）  用牛顿法求解函数 的最小值。  ；;  实验4.2：（优化问题3）  用最速下降法求解函数 的最小值。  ；;  实验4.3：（优化问题4）  用共轭梯度法求解函数的最小值。  ；  实验5.1：（优化问题5）  求函数 *在(-3,4)的最小值*。  *取x0 = -3 ,或 x0 = 4，或 (x0,x1) = (-3,4)*  实验5.2\*：（优化问题6）  求函数  在(-3,3)\* (-3,3)区间上的最小值， 取(x0,y0)=(3.0,0.0) | | | |
| 实验过程与结果：  实验1.1：（优化问题1）  #include <stdio.h>  #include <math.h>  double f(double x) {  return x\*x -sin(x);  }  double df(double x) {  return 2\*x - cos(x);  }  double d2f(double x) {  return 2 + sin(x);  }  int main() {  double a = 0;  double b = 1;  double eps\_x = 1e-5;  double eps\_dy = 1e-4;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  double min\_x = a;  double max\_x = b;  double r = 0.6180340025899865;  double r0 = 1 - r;  double right\_x = r\* (max\_x - min\_x) + min\_x;  double left\_x = r0\* (max\_x - min\_x) + min\_x;  double result\_x = right\_x;    printf("\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tdf(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, result\_x, f(result\_x), df(result\_x));  while ((fabs(max\_x - min\_x) >= eps\_x) && (fabs(df(max\_x) - df(result\_x)) + fabs(df(min\_x) - df(result\_x)) >= eps\_y) && (count <= MAX\_COUNT )) {  if (f(right\_x) > f(left\_x)) {  max\_x = right\_x;  right\_x = left\_x;  left\_x = r0\* (max\_x - min\_x) + min\_x;  result\_x = left\_x;  } else {  min\_x = left\_x;  left\_x = right\_x;  right\_x = r\* (max\_x - min\_x) + min\_x;  result\_x = right\_x;  }    printf("%d\t%lf\t%lf\t%lf\n", count+1, result\_x, f(result\_x), df(result\_x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(df(result\_x)) >= eps\_y) && (fabs(max\_x - min\_x) >= eps\_x) && (count <= MAX\_COUNT )) {  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("The result is %lf\n", result\_x);  }  return 0;  }    实验1.2\*：（优化问题2）  #include <stdio.h>  #include <math.h>  double f(double x) {  return 0.5 - x\*exp(-x\*x);  }  double df(double x) {  return (2\*x\*x - 1) \* exp(-x\*x);  }  double d2f(double x) {  return (-4\*x\*x\*x + 6\*x) \* exp(-x\*x);  }  int main() {  double a = 0;  double b = 1.2;  double eps\_x = 1e-5;  double eps\_dy = 1e-4;  double eps\_y = 1e-4;  double min\_x = a;  double max\_x = b;  int count = 1;  int MAX\_COUNT = 100;  double r = 0.6180340025899865;  double r0 = 1 - r;  double right\_x = r\* (max\_x - min\_x) + min\_x;  double left\_x = r0\* (max\_x - min\_x) + min\_x;  double result\_x = right\_x;    printf("\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tdf(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, result\_x, f(result\_x), df(result\_x));  while ((fabs(max\_x - min\_x) >= eps\_x) && (fabs(df(max\_x) - df(result\_x)) + fabs(df(min\_x) - df(result\_x)) >= eps\_y) && (count <= MAX\_COUNT )) {  if (f(right\_x) > f(left\_x)) {  max\_x = right\_x;  right\_x = left\_x;  left\_x = r0\* (max\_x - min\_x) + min\_x;  result\_x = left\_x;  } else {  min\_x = left\_x;  left\_x = right\_x;  right\_x = r\* (max\_x - min\_x) + min\_x;  result\_x = right\_x;  }    printf("%d\t%lf\t%lf\t%lf\n", count+1, result\_x, f(result\_x), df(result\_x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(df(result\_x)) >= eps\_y) && (fabs(max\_x - min\_x) >= eps\_x) && (count <= MAX\_COUNT )) {  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("The result is %lf\n", result\_x);  }  return 0;  }    实验2.1：（优化问题1）  #include <stdio.h>  #include <math.h>  double f(double x) {  return x\*x -sin(x);  }  double df(double x) {  return 2\*x - cos(x);  }  double d2f(double x) {  return 2 + sin(x);  }    double func(double x1, double x2, double x3) {  //生成三点生成二次函数的极值点  double y1 = f(x1);  double y2 = f(x2);  double y3 = f(x3);  return 0.5\*((x2\*x2 - x3\*x3)\*y1 + (x3\*x3 - x1\*x1)\*y2 + (x1\*x1 - x2\*x2)\*y3) / ((x2 - x3)\*y1 + (x3 - x1)\*y2 + (x1 - x2)\*y3);  }  int main() {  double left = 0;  double right = 1;  double eps\_x = 1e-5;  double eps\_dy = 1e-4;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  double min\_x = left;  double max\_x = right;  double last\_x = 0.5\*(max\_x - min\_x) + min\_x;  double new\_x = func(min\_x,last\_x,max\_x);  double result\_x = new\_x;    printf("\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tdf(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, result\_x, f(result\_x), df(result\_x));  while ((fabs(max\_x - min\_x) >= eps\_x) && (fabs(df(max\_x) - df(result\_x)) + fabs(df(min\_x) - df(result\_x)) >= eps\_y) && (count <= MAX\_COUNT ) &&(fabs(new\_x - last\_x) >= eps\_x)) {  if (new\_x > last\_x) {  if (f(new\_x) > f(last\_x)) {  max\_x = new\_x;  last\_x = last\_x;  new\_x = func(min\_x,last\_x,max\_x);  result\_x = new\_x;  } else {  min\_x = last\_x;  last\_x = new\_x;  new\_x = func(min\_x,last\_x,max\_x);  result\_x = new\_x;  }  printf("%d\t%lf\t%lf\t%lf\n", count+1, result\_x, f(result\_x), df(result\_x));  count++;  } else {  if (f(last\_x) > f(new\_x)) {  max\_x = last\_x;  last\_x = new\_x;  new\_x = func(min\_x,last\_x,max\_x);  result\_x = new\_x;  } else {  min\_x = new\_x;  last\_x = last\_x;  new\_x = func(min\_x,last\_x,max\_x);  result\_x = new\_x;  }  printf("%d\t%lf\t%lf\t%lf\n", count+1, result\_x, f(result\_x), df(result\_x));  count++;  }  }      printf("%d\t%lf\t%lf\t%lf\n", count+1, result\_x, f(result\_x), df(result\_x));  count++;    printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(df(result\_x)) >= eps\_y) && (fabs(max\_x - min\_x) >= eps\_x) && (count <= MAX\_COUNT )&&(fabs(new\_x - last\_x) >= eps\_x)) {  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("The result is %lf\n", result\_x);  }  return 0;  }    实验2.2：（优化问题2）  #include <stdio.h>  #include <math.h>  double f(double x) {  return 0.5 - x\*exp(-x\*x);  }  double df(double x) {  return (2\*x\*x - 1) \* exp(-x\*x);  }  double d2f(double x) {  return (-4\*x\*x\*x + 6\*x) \* exp(-x\*x);  }    double func(double x1, double x2, double x3) {  //生成三点生成二次函数的极值点  double y1 = f(x1);  double y2 = f(x2);  double y3 = f(x3);  return 0.5\*((x2\*x2 - x3\*x3)\*y1 + (x3\*x3 - x1\*x1)\*y2 + (x1\*x1 - x2\*x2)\*y3) / ((x2 - x3)\*y1 + (x3 - x1)\*y2 + (x1 - x2)\*y3);  }  int main() {  double left = 0;  double right = 1.2;  double eps\_x = 1e-5;  double eps\_dy = 1e-4;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  double min\_x = left;  double max\_x = right;  double last\_x = 0.5\*(max\_x - min\_x) + min\_x;  double new\_x = func(min\_x,last\_x,max\_x);  double result\_x = new\_x;    printf("\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tdf(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, result\_x, f(result\_x), df(result\_x));  while ((fabs(max\_x - min\_x) >= eps\_x) && (fabs(df(max\_x) - df(result\_x)) + fabs(df(min\_x) - df(result\_x)) >= eps\_y) && (count <= MAX\_COUNT ) &&(fabs(new\_x - last\_x) >= eps\_x)) {  if (new\_x > last\_x) {  if (f(new\_x) > f(last\_x)) {  max\_x = new\_x;  last\_x = last\_x;  new\_x = func(min\_x,last\_x,max\_x);  result\_x = new\_x;  } else {  min\_x = last\_x;  last\_x = new\_x;  new\_x = func(min\_x,last\_x,max\_x);  result\_x = new\_x;  }  printf("%d\t%lf\t%lf\t%lf\n", count+1, result\_x, f(result\_x), df(result\_x));  count++;  } else {  if (f(last\_x) > f(new\_x)) {  max\_x = last\_x;  last\_x = new\_x;  new\_x = func(min\_x,last\_x,max\_x);  result\_x = new\_x;  } else {  min\_x = new\_x;  last\_x = last\_x;  new\_x = func(min\_x,last\_x,max\_x);  result\_x = new\_x;  }  printf("%d\t%lf\t%lf\t%lf\n", count+1, result\_x, f(result\_x), df(result\_x));  count++;  }  }          printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(df(result\_x)) >= eps\_y) && (fabs(max\_x - min\_x) >= eps\_x) && (count <= MAX\_COUNT )&&(fabs(new\_x - last\_x) >= eps\_x)) {  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("The result is %lf\n", result\_x);  }  return 0;  }    实验3.1：（优化问题1）  #include <stdio.h>  #include <math.h>  double f(double x) {  return x\*x -sin(x);  }  double df(double x) {  return 2\*x - cos(x);  }  double d2f(double x) {  return 2 + sin(x);  }  double newton(double x0) {  return x0 - df(x0)/d2f(x0);  }    double func(double x1, double x2, double x3) {  //生成三点生成二次函数的极值点  double y1 = f(x1);  double y2 = f(x2);  double y3 = f(x3);  return 0.5\*((x2\*x2 - x3\*x3)\*y1 + (x3\*x3 - x1\*x1)\*y2 + (x1\*x1 - x2\*x2)\*y3) / ((x2 - x3)\*y1 + (x3 - x1)\*y2 + (x1 - x2)\*y3);  }  int main() {  double left = 0;  double right = 1;  double eps\_x = 1e-5;  double eps\_dy = 1e-4;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  // 牛顿迭代法  double x0 = 1;  double x = x0;  double y = df(x0);    printf("\n");  printf("Newton's iteration method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tdf(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, x, f(x), df(x));  while ((fabs(y) >= eps\_y) && (fabs(newton(x)-x) >= eps\_x) && (fabs(df(x))>=eps\_y) && (count <= MAX\_COUNT ) ){    x = newton(x);  y = df(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, f(x), df(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(y) >= eps\_y) && (fabs(newton(x)-x) >= eps\_x) && (fabs(df(x))>=eps\_y)) {  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else if(x <=left || x >= right){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is %lf\n", x);  }  printf("\n");  return 0;  }    实验3.2：（优化问题2）  #include <stdio.h>  #include <math.h>  double f(double x) {  return 0.5 - x\*exp(-x\*x);  }  double df(double x) {  return (2\*x\*x - 1) \* exp(-x\*x);  }  double d2f(double x) {  return (-4\*x\*x\*x + 6\*x) \* exp(-x\*x);  }  double newton(double x0) {  return x0 - df(x0)/d2f(x0);  }    double func(double x1, double x2, double x3) {  //生成三点生成二次函数的极值点  double y1 = f(x1);  double y2 = f(x2);  double y3 = f(x3);  return 0.5\*((x2\*x2 - x3\*x3)\*y1 + (x3\*x3 - x1\*x1)\*y2 + (x1\*x1 - x2\*x2)\*y3) / ((x2 - x3)\*y1 + (x3 - x1)\*y2 + (x1 - x2)\*y3);  }  int main() {  double left = 0;  double right = 2;  double eps\_x = 1e-5;  double eps\_dy = 1e-4;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  // 牛顿迭代法  double x0 = 1;  double x = x0;  double y = df(x0);    printf("\n");  printf("Newton's iteration method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tdf(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, x, f(x), df(x));  while ((fabs(y) >= eps\_y) && (fabs(newton(x)-x) >= eps\_x) && (fabs(df(x))>=eps\_y) && (count <= MAX\_COUNT ) ){    x = newton(x);  y = df(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, f(x), df(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(y) >= eps\_y) && (fabs(newton(x)-x) >= eps\_x) && (fabs(df(x))>=eps\_y)) {  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else if(x <=left || x >= right){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is %lf\n", x);  }  printf("\n");  return 0;  }    实验4.1：（优化问题3）  #include <stdio.h>  #include <math.h>  #include <stdlib.h>  double f(double x1,double x2) {  return (x1-x2)/(x1\*x1 + x2\*x2 + 2);  }  double dfx1(double x1,double x2) {  return (-1\*x1\*x1 + x2\*x2 + 2\*x1\*x2 + 2) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 + 2));  }  double dfx2(double x1,double x2) {  return (x2\*x2 - x1\*x1 - 2\*x1\*x2 - 2) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 + 2));    }  double dfx1x1(double x1,double x2) {  return (2\*(x1\*x1\*x1 - 3\*x1\*x1\*x2 - 6\*x1 - 3\*x1\*x2\*x2 + x2\*x2\*x2 + 2\*x2)) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2));  }  double dfx1x2(double x1,double x2) {  return (2\*(x1\*x1\*x1 + 3\*x1\*x1\*x2 + 2\*x1 - 3\*x1\*x2\*x2 - x2\*x2\*x2 - 2\*x2)) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2));  }  double dfx2x1(double x1,double x2) {  return (-2\*(x2\*x2\*x2 + 3\*x2\*x2\*x1 + 2\*x2 - 3\*x2\*x1\*x1 - x1\*x1\*x1 - 2\*x1)) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2));  }  double dfx2x2(double x1,double x2) {  return (2\*(-1\*x2\*x2\*x2 + 3\*x2\*x2\*x1 + 6\*x2 + 3\*x2\*x1\*x1 - x1\*x1\*x1 - 2\*x1)) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2));  }  double newton\_x1(double x1, double x2){  double x1x1 = dfx1x1(x1,x2);  double x1x2 = dfx1x2(x1,x2);  double x2x1 = dfx2x1(x1,x2);  double x2x2 = dfx2x2(x1,x2);  double dx1 = dfx1(x1,x2);  double dx2 = dfx2(x1,x2);  double det = x1x1\*x2x2 - x1x2\*x2x1;  double x1\_new = x1 - (1/det)\*(dx1\*x2x2 - dx2\*x2x1);  return x1\_new;  }  double newton\_x2(double x1, double x2){  double x1x1 = dfx1x1(x1,x2);  double x1x2 = dfx1x2(x1,x2);  double x2x1 = dfx2x1(x1,x2);  double x2x2 = dfx2x2(x1,x2);  double dx1 = dfx1(x1,x2);  double dx2 = dfx2(x1,x2);  double det = x1x1\*x2x2 - x1x2\*x2x1;  double x2\_new = x1 - (1/det)\*(dx2\*x1x1 - dx1\*x1x2 );  return x2\_new;  }  void newton\_solve(double \*x1, double \*x2) {  double x1x1 = dfx1x1(\*x1,\*x2);  double x1x2 = dfx1x2(\*x1,\*x2);  double x2x1 = dfx2x1(\*x1,\*x2);  double x2x2 = dfx2x2(\*x1,\*x2);  double dx1 = dfx1(\*x1,\*x2);  double dx2 = dfx2(\*x1,\*x2);  double \*\*A = NULL;  int n = 2;    A = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  A[i] = (double \*)malloc(n \* sizeof(double));  }  if(A == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }    A[0][0] = x1x1;A[0][1] = x1x2;  A[1][0] = x2x1;A[1][1] = x2x2;        double\* b = (double\*)malloc(n \* sizeof(double));  b[0] = -dx1 + (\*x1)\*x1x1 + (\*x2)\*x1x2;  b[1] = -dx2 + (\*x1)\*x2x1 + (\*x2)\*x2x2;  //b[1]出现了误差，发现其误差由dx2引起  //第一步b[1]应该是 0.56011233393660；  //但计算结果为 0.5380709  //列主元Gauss消去法  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[j][i]) > fabs(A[max\_index][i])) {  max\_index = j;  }  }  if (A[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(1);  }  if (max\_index != i) {//交换两行  for(int j = i; j < n; j++) {  double temp = A[i][j];  A[i][j] = A[max\_index][j];  A[max\_index][j] = temp;  }  double temp\_b = b[i];  b[i] = b[max\_index];  b[max\_index] = temp\_b;  }  for (int j = i + 1; j < n; j++) {//消去法  double factor = A[j][i] / A[i][i];  A[j][i] = 0;  for (int k = i + 1; k < n; k++) {  A[j][k] -= factor \* A[i][k];  }  b[j] -= factor \* b[i];  }  }    double\* x = (double\*)malloc(n \* sizeof(double));  // 解线性方程组  for (int i = n - 1; i >= 0; i--) {  double ad = b[i];  for (int j = n-1; j >= i+1; j--) {  ad -= A[i][j] \* x[j];  }  x[i] = ad/A[i][i];  }  \*x1 = x[0];  \*x2 = x[1];  for (int i = 0; i < n; i++) {  free(A[i]);  }  free(A);  free(b);  free(x);    }  int main() {    double eps\_x = 1e-5;  double eps\_dy = 1e-4;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 10;  // 牛顿迭代法  double x1 = -0.3;  double x2 = 0.2;    double y = f(x1,x2);  double dy\_x1 = dfx1(x1,x2);  double dy\_x2 = dfx2(x1,x2);  double det = dfx1x1(x1,x2)\*dfx2x2(x1,x2) - dfx1x2(x1,x2)\*dfx2x1(x1,x2);  double x1x1 = dfx1x1(x1,x2);  double x1x2 = dfx1x2(x1,x2);  double x2x1 = dfx2x1(x1,x2);  double x2x2 = dfx2x2(x1,x2);    printf("\n");  printf("Newton's iteration method:\n");  printf("--------------------------\n");  printf("iter\tx\t\ty\t\tf(x,y)\t\tdf(x)/dx\tdf(x)/dy\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", count, x1,x2, f(x1,x2), dfx1(x1,x2), dfx2(x1,x2));    while ( ((fabs(dfx1(x1,x2)) >= eps\_y) || (fabs(dfx2(x1,x2)) >= eps\_y)) && (count <= MAX\_COUNT )){  /\* //求逆解方程  x1 =newton\_x1(x1, x2);  x2 = newton\_x2(x1, x2); \*/  //列主元高斯消去法  newton\_solve(&x1, &x2);  y = f(x1,x2);  dy\_x1 = dfx1(x1,x2);  dy\_x2 = dfx2(x1,x2);  double det = dfx1x1(x1,x2)\*dfx2x2(x1,x2) - dfx1x2(x1,x2)\*dfx2x1(x1,x2);  double x1x1 = dfx1x1(x1,x2);  double x1x2 = dfx1x2(x1,x2);  double x2x1 = dfx2x1(x1,x2);  double x2x2 = dfx2x2(x1,x2);  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", count+1, x1,x2, f(x1,x2), dfx1(x1,x2), dfx2(x1,x2));    count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && ((fabs(dfx1(x1,x2)) >= eps\_y) || (fabs(dfx2(x1,x2)) >= eps\_y)) ) {  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else if(isnan(x1) || isnan(x2) || isnan(f(x1,x2))){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is x=%lf, y=%lf, f(x,y)=%lf\n", x1,x2, f(x1,x2));  }  printf("\n");  return 0;  }    实验4.2：（优化问题3）  用最速下降法求解函数 的最小值。  ；;  #include <stdio.h>  #include <math.h>  #include <stdlib.h>  double f(double x1,double x2) {  return (x1-x2)/(x1\*x1 + x2\*x2 + 2);  }  double dfx1(double x1,double x2) {  return (-1\*x1\*x1 + x2\*x2 + 2\*x1\*x2 + 2) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 + 2));  }  double dfx2(double x1,double x2) {  return (x2\*x2 - x1\*x1 - 2\*x1\*x2 - 2) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 + 2));    }  double dfx1x1(double x1,double x2) {  return (2\*(x1\*x1\*x1 - 3\*x1\*x1\*x2 - 6\*x1 - 3\*x1\*x2\*x2 + x2\*x2\*x2 + 2\*x2)) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2));  }  double dfx1x2(double x1,double x2) {  return (2\*(x1\*x1\*x1 + 3\*x1\*x1\*x2 + 2\*x1 - 3\*x1\*x2\*x2 - x2\*x2\*x2 - 2\*x2)) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2));  }  double dfx2x1(double x1,double x2) {  return (-2\*(x2\*x2\*x2 + 3\*x2\*x2\*x1 + 2\*x2 - 3\*x2\*x1\*x1 - x1\*x1\*x1 - 2\*x1)) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2));  }  double dfx2x2(double x1,double x2) {  return (2\*(-1\*x2\*x2\*x2 + 3\*x2\*x2\*x1 + 6\*x2 + 3\*x2\*x1\*x1 - x1\*x1\*x1 - 2\*x1)) / ((x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2)\*(x1\*x1 + x2\*x2 +2));  }  void solve(double \*x1, double \*x2) {  double x1x1 = dfx1x1(\*x1,\*x2);  double x1x2 = dfx1x2(\*x1,\*x2);  double x2x1 = dfx2x1(\*x1,\*x2);  double x2x2 = dfx2x2(\*x1,\*x2);  double dx1 = dfx1(\*x1,\*x2);  double dx2 = dfx2(\*x1,\*x2);  double \*\*Q = NULL;  int n = 2;    Q = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  Q[i] = (double \*)malloc(n \* sizeof(double));  }  if(Q == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }    Q[0][0] = x1x1;Q[0][1] = x1x2;  Q[1][0] = x2x1;Q[1][1] = x2x2;  double \*p = (double \*)malloc(n \* sizeof(double));  if(p == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  p[0] = -dx1;  p[1] = -dx2;  double \* g = (double \*)malloc(n \* sizeof(double));  if(g == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  g[0] = dx1;  g[1] = dx2;  double ak = (g[0]\*g[0] + g[1]\*g[1])/(p[0]\*p[0] + p[1]\*p[1]);  double lambda = -1 \* (p[0]\*g[0] + p[1]\*g[1])/((Q[0][0]\*p[0] + Q[0][1]\*p[1])\*p[0] + (Q[1][0]\*p[0] + Q[1][1]\*p[1]) \* p[1]);    //\*x1 = \*x1 - lambda\*dx1;  //\*x2 = \*x2 - lambda\*dx2;  \*x1 = \*x1 - ak\*dx1;  \*x2 = \*x2 - ak\*dx2;    //释放内存  for (int i = 0; i < n; i++) {  free(Q[i]);  }  free(Q);  free(p);  free(g);    }  int main() {    double eps\_x = 1e-5;  double eps\_dy = 1e-4;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  //最速下降法  double x1 = -3;  double x2 = -2;    double y = f(x1,x2);  double dy\_x1 = dfx1(x1,x2);  double dy\_x2 = dfx2(x1,x2);    double x1x1 = dfx1x1(x1,x2);  double x1x2 = dfx1x2(x1,x2);  double x2x1 = dfx2x1(x1,x2);  double x2x2 = dfx2x2(x1,x2);    printf("\n");  printf("\n");  printf("--------------------------\n");  printf("iter\tx\t\ty\t\tf(x,y)\t\tdf(x)/dx\tdf(x)/dy\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", count, x1,x2, f(x1,x2), dfx1(x1,x2), dfx2(x1,x2));    while ( ((fabs(dfx1(x1,x2)) >= eps\_y) || (fabs(dfx2(x1,x2)) >= eps\_y)) && (count <= MAX\_COUNT )){    //最速下降法  solve(&x1, &x2);  y = f(x1,x2);  dy\_x1 = dfx1(x1,x2);  dy\_x2 = dfx2(x1,x2);  double x1x1 = dfx1x1(x1,x2);  double x1x2 = dfx1x2(x1,x2);  double x2x1 = dfx2x1(x1,x2);  double x2x2 = dfx2x2(x1,x2);  double dx1 = dfx1(x1,x2);  double dx2 = dfx2(x1,x2);  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", count+1, x1,x2, f(x1,x2), dfx1(x1,x2), dfx2(x1,x2));    count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && ((fabs(dfx1(x1,x2)) >= eps\_y) || (fabs(dfx2(x1,x2)) >= eps\_y)) ) {  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else if(isnan(x1) || isnan(x2) || isnan(f(x1,x2))){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is x=%lf, y=%lf, f(x,y)=%lf\n", x1,x2, f(x1,x2));  }  printf("\n");  return 0;  }  /\* int main() {    double eps\_x = 1e-5;  double eps\_dy = 1e-4;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;    double x1 = -3;  double x2 = 2;    double y = f(x1,x2);  double dy\_x1 = dfx1(x1,x2);  double dy\_x2 = dfx2(x1,x2);    double x1x1 = dfx1x1(x1,x2);  double x1x2 = dfx1x2(x1,x2);  double x2x1 = dfx2x1(x1,x2);  double x2x2 = dfx2x2(x1,x2);  double dx1 = dfx1(x1,x2);  double dx2 = dfx2(x1,x2);    printf("\n");  printf("\n");  printf("--------------------------\n");  printf("iter\tx\t\ty\t\tf(x,y)\t\tdf(x)/dx\tdf(x)/dy\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", count, x1,x2, f(x1,x2), dfx1(x1,x2), dfx2(x1,x2));    double \*\*Q = NULL;  int n = 2;    Q = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  Q[i] = (double \*)malloc(n \* sizeof(double));  }  if(Q == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }    Q[0][0] = x1x1;Q[0][1] = x1x2;  Q[1][0] = x2x1;Q[1][1] = x2x2;  double \*d = (double \*)malloc(n \* sizeof(double));  if(d == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }    double \* g = (double \*)malloc(n \* sizeof(double));  if(g == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  g[0] = dx1;  g[1] = dx2;  d[0] = -g[0];  d[1] = -g[1];  int result = 1;  while (((fabs(g[0])>eps\_y) || (fabs(g[1])>eps\_y)) && (count <= MAX\_COUNT )){    result = 0;  double x1x1 = dfx1x1(x1,x2);  double x1x2 = dfx1x2(x1,x2);  double x2x1 = dfx2x1(x1,x2);  double x2x2 = dfx2x2(x1,x2);  double dx1 = dfx1(x1,x2);  double dx2 = dfx2(x1,x2);    //double ak = -1 \* (g[0]\*d[0] + g[1]\*d[1])/((Q[0][0]\*d[0] + Q[0][1]\*d[1])\*d[0] + (Q[1][0]\*d[0] + Q[1][1]\*d[1]) \* d[1]);  double ak = (g[0]\*g[0] + g[1]\*g[1])/(d[0]\*d[0] + d[1]\*d[1]);  x1 = x1 + ak \* d[0];  x2 = x2 + ak \* d[1];  g[0] = dfx1(x1, x2);  g[1] = dfx2(x1, x2);  //if ((fabs(g[0])<eps\_y)&&(fabs(g[1])<eps\_y)) {  // result = 1;  // break;  //} else {  // //double bk = ((Q[0][0]\*d[0] + Q[0][1]\*d[1])\*g[0] + (Q[1][0]\*d[0] + Q[1][1]\*d[1]) \* g[1])/((Q[0][0]\*d[0] + Q[0][1]\*d[1])\*d[0] + (Q[1][0]\*d[0] + Q[1][1]\*d[1]) \* d[1]);  // double bk = (g[0]\*g[0] + g[1]\*g[1])/(d[0]\*d[0] + d[1]\*d[1]);  // d[0] = -g[0] + bk\*d[0];  // d[1] = -g[1] + bk\*d[1];  //}  d[0] = -g[0];  d[1] = -g[1];  y = f(x1,x2);  dy\_x1 = dfx1(x1,x2);  dy\_x2 = dfx2(x1,x2);  x1x1 = dfx1x1(x1,x2);  x1x2 = dfx1x2(x1,x2);  x2x1 = dfx2x1(x1,x2);  x2x2 = dfx2x2(x1,x2);  dx1 = dfx1(x1,x2);  dx2 = dfx2(x1,x2);  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", count+1, x1,x2, f(x1,x2), dfx1(x1,x2), dfx2(x1,x2));    count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && ((fabs(dfx1(x1,x2)) >= eps\_y) || (fabs(dfx2(x1,x2)) >= eps\_y)) ) {  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else if(isnan(x1) || isnan(x2) || isnan(f(x1,x2))){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is x=%lf, y=%lf, f(x,y)=%lf\n", x1,x2, f(x1,x2));  }  printf("\n");  //释放内存  for (int i = 0; i < n; i++) {  free(Q[i]);  }  free(Q);  free(d);  free(g);  return 0;  } \*/    实验4.3：（优化问题4）  用共轭梯度法求解函数的最小值。  ；  #include <stdio.h>  #include <math.h>  #include <stdlib.h>  double f(double x1,double x2) {  return 0.5\*x1\*x1 + 2.5\*x2\*x2;  }  double dfx1(double x1,double x2) {  return x1;  }  double dfx2(double x1,double x2) {  return 5\*x2;    }  double dfx1x1(double x1,double x2) {  return 1;  }  double dfx1x2(double x1,double x2) {  return 0;  }  double dfx2x1(double x1,double x2) {  return 0;  }  double dfx2x2(double x1,double x2) {  return 5;  }            int main() {    double eps\_x = 1e-5;  double eps\_dy = 1e-4;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  //最速下降法  double x1 = 5;  double x2 = 1;    double y = f(x1,x2);  double dy\_x1 = dfx1(x1,x2);  double dy\_x2 = dfx2(x1,x2);    double x1x1 = dfx1x1(x1,x2);  double x1x2 = dfx1x2(x1,x2);  double x2x1 = dfx2x1(x1,x2);  double x2x2 = dfx2x2(x1,x2);  double dx1 = dfx1(x1,x2);  double dx2 = dfx2(x1,x2);    printf("\n");  printf("\n");  printf("--------------------------\n");  printf("iter\tx\t\ty\t\tf(x,y)\t\tdf(x)/dx\tdf(x)/dy\n");  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", count, x1,x2, f(x1,x2), dfx1(x1,x2), dfx2(x1,x2));    double \*\*Q = NULL;  int n = 2;    Q = (double \*\*)malloc(n \* sizeof(double \*));  for (int i = 0; i < n; i++) {  Q[i] = (double \*)malloc(n \* sizeof(double));  }  if(Q == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }    Q[0][0] = x1x1;Q[0][1] = x1x2;  Q[1][0] = x2x1;Q[1][1] = x2x2;  double \*d = (double \*)malloc(n \* sizeof(double));  if(d == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }    double \* g = (double \*)malloc(n \* sizeof(double));  if(g == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  g[0] = dx1;  g[1] = dx2;  d[0] = -g[0];  d[1] = -g[1];  int result = 1;  while (((fabs(g[0])>eps\_y) || (fabs(g[1])>eps\_y)) && (count <= MAX\_COUNT )){    result = 0;  double x1x1 = dfx1x1(x1,x2);  double x1x2 = dfx1x2(x1,x2);  double x2x1 = dfx2x1(x1,x2);  double x2x2 = dfx2x2(x1,x2);  double dx1 = dfx1(x1,x2);  double dx2 = dfx2(x1,x2);    double ak = -1 \* (g[0]\*d[0] + g[1]\*d[1])/((Q[0][0]\*d[0] + Q[0][1]\*d[1])\*d[0] + (Q[1][0]\*d[0] + Q[1][1]\*d[1]) \* d[1]);    x1 = x1 + ak \* d[0];  x2 = x2 + ak \* d[1];  g[0] = dfx1(x1, x2);  g[1] = dfx2(x1, x2);  if ((fabs(g[0])<eps\_y)&&(fabs(g[1])<eps\_y)) {  result = 1;  break;  } else {  //double bk = ((Q[0][0]\*d[0] + Q[0][1]\*d[1])\*g[0] + (Q[1][0]\*d[0] + Q[1][1]\*d[1]) \* g[1])/((Q[0][0]\*d[0] + Q[0][1]\*d[1])\*d[0] + (Q[1][0]\*d[0] + Q[1][1]\*d[1]) \* d[1]);  double bk = (g[0]\*g[0] + g[1]\*g[1])/(d[0]\*d[0] + d[1]\*d[1]);  d[0] = -g[0] + bk\*d[0];  d[1] = -g[1] + bk\*d[1];  }  y = f(x1,x2);  dy\_x1 = dfx1(x1,x2);  dy\_x2 = dfx2(x1,x2);  x1x1 = dfx1x1(x1,x2);  x1x2 = dfx1x2(x1,x2);  x2x1 = dfx2x1(x1,x2);  x2x2 = dfx2x2(x1,x2);  dx1 = dfx1(x1,x2);  dx2 = dfx2(x1,x2);  printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", count+1, x1,x2, f(x1,x2), dfx1(x1,x2), dfx2(x1,x2));    count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && ((fabs(dfx1(x1,x2)) >= eps\_y) || (fabs(dfx2(x1,x2)) >= eps\_y)) ) {  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else if(isnan(x1) || isnan(x2) || isnan(f(x1,x2))){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is x=%lf, y=%lf, f(x,y)=%lf\n", x1,x2, f(x1,x2));  }  printf("\n");  //释放内存  for (int i = 0; i < n; i++) {  free(Q[i]);  }  free(Q);  free(d);  free(g);  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1.1，2.1，3.1；1.2，2.2，3.2之间的对比，我们发现  黄金分隔搜索与二分法类似，其求解具有稳定性，不会出现有解却找不到的情况，但其无法加速  逐次抛物插值搜索在函数具有类似二次函数的形状时可以极大提高迭代过程，但可能导致有解找不到  牛顿法与逐次抛物法类似，在函数满足其假设条件：导数近似单调时可以有效提速，但可能找不到解  实验4.1，4.2，4.3比较了牛顿法，最速下降法，共轭梯度法  牛顿法求解速度快，但计算复杂度高，在求解小规模问题时适用，  最速下降法的计算复杂度第，但迭代步数多，适用于大规模问题  共轭梯度法适用于大规模问题，且迭代速度快 | | | |

注：若填写内容较多，可在背面继续填写。