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：实验四： 非线性方程的迭代求解 | |
| 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： 2024/7/7 | 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；熟悉求解非线性方程的区间逼近法（二分法、试值法），不动点迭代法（简单迭代法、加速迭代法），和牛顿类迭代法（牛顿迭代法、割线法）；了解以上方法的算法的稳定性与收敛速度特点；熟悉高阶迭代法在处理特殊病态问题时的收敛性问题，体会二分法作为外部嵌套迭代算法的必要性。  【实验内容】  实验1.1：（分别用二分法与试值法求解非线性方程1）  用二分法与试值法求方程 在 区间的根.  （）  实验1.2：（分别用二分法与试值法求解非线性方程2）  用二分法与试值法求方程 在 区间的根.  （）  实验1.3：（分别用二分法与试值法求解非线性方程5）  用二分法与试值法求方程在 区间的根.  （）  实验2.1：（用简单迭代法求解非线性方程3）  用简单迭代法求方程在 区间的根.  （）  其中迭代公式分别取：  迭代公式：；  迭代公式：；  迭代公式：；  比较其收敛性差别。  实验2.2：（用简单迭代法与加速迭代法求解非线性方程5）  用加速迭代法求方程在 区间的根.  （）  其中简单迭代公式取：,比较其收敛性差别。  实验3.1：（用牛顿迭代法与割线法求解非线性方程3）  用牛顿迭代法与割线法求方程在 区间的根.    （）。  实验3.2：（分别用牛顿迭代法与割线法求解非线性方程5）  用牛顿迭代法与割线法求方程在 区间的根.    （）。  实验3.3：（分别用牛顿迭代法与割线法求解非线性方程2）  用牛顿迭代法与割线法求方程 在 区间的根.  （）。  实验4.1：（获取下面方程的一个解）  求方程 在 区间的一个根.  取x0 = -3 ,或 x0 = 4，或 (x0,x1) = (-3,4) | | | |
| 实验过程与结果：  实验1.1：（分别用二分法与试值法求解非线性方程1）  #include <stdio.h>  #include <math.h>  double f(double x) {  return x\*sin(x) - 1;  }  double x\_step(double a, double b){  return b - f(b)\*(b-a)/(f(b)-f(a));  }  int main(void) {  double a = 0;  double b = 2;  double min\_x = a;  double max\_x = b;  double eps\_x = 1e-5;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  // dichotomy method  double x = (min\_x + max\_x) / 2;  double y = f(x);  printf("dichotomy\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(x-min\_x) >= eps\_x) && (count <= MAX\_COUNT )) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  x = (min\_x + max\_x) / 2;  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, y, x-f(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(y) >= eps\_y) && (fabs(x-min\_x) >= eps\_x)) {  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 <=a || x >= b){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is %lf\n", x);  }  printf("\n");  //trial value method  printf("trial value method\n");  min\_x = a;  max\_x = b;  count = 1;  x = x\_step(min\_x, max\_x);  y = f(x);  printf("dichotomy\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, x, y, x-f(x));  int last\_x = min\_x - max\_x;//保证初始时，上一个x不在区间内  while ((fabs(y) >= eps\_y) && (fabs(max\_x-min\_x) >= eps\_x) && (count <= MAX\_COUNT ) && (fabs((x\_step(min\_x, max\_x)-min\_x)\*(x\_step(min\_x, max\_x)-max\_x))>=eps\_x)) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  last\_x = x;  x = x\_step(min\_x, max\_x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, y, x-f(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(y) >= eps\_y) && (fabs(max\_x-min\_x) >= eps\_x) && (fabs((x\_step(min\_x, max\_x)-min\_x)\*(x\_step(min\_x, max\_x)-max\_x))>=eps\_x)) {  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 <=a || x >= b){  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1.2：（分别用二分法与试值法求解非线性方程2）  #include <stdio.h>  #include <math.h>  double f(double x) {  return exp(-100\*x) - 1;  }  double x\_step(double a, double b){  return b - f(b)\*((b-a)/(f(b)-f(a)));  }  int main(void) {  double a = -0.51;  double b = 0.49;  double min\_x = a;  double max\_x = b;  double eps\_x = 1e-5;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  // dichotomy method  double x = (min\_x + max\_x) / 2;  double y = f(x);  printf("dichotomy\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(x-min\_x) >= eps\_x) && (count <= MAX\_COUNT )) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  x = (min\_x + max\_x) / 2;  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, y, x-f(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(y) >= eps\_y) && (fabs(x-min\_x) >= eps\_x)) {  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 <=a || x >=b){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is %lf\n", x);  }  printf("\n");  //trial value method  printf("trial value method\n");  min\_x = a;  max\_x = b;  count = 1;  printf("%lf, %lf\n", min\_x, max\_x);  printf("%lf %lf\n", f(min\_x), f(max\_x));  x = x\_step(min\_x, max\_x);  printf("%lf\n", x);  y = f(x);  printf("%lf\n", y);  printf("dichotomy\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, x, y, x-f(x));  int last\_x = min\_x - max\_x;//保证初始时，上一个x不在区间内  /\* while ((fabs(f(x\_step(min\_x, max\_x))) >= eps\_y) && (fabs(max\_x-min\_x) >= eps\_x) && (count <= MAX\_COUNT ) && (fabs((x\_step(min\_x, max\_x)-min\_x)\*(x\_step(min\_x, max\_x)-max\_x))>=eps\_x)) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  last\_x = x;  x = x\_step(min\_x, max\_x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, y, x-f(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(f(x\_step(min\_x, max\_x))) >= eps\_y) && (fabs(max\_x-min\_x) >= eps\_x) && (fabs((x\_step(min\_x, max\_x)-min\_x)\*(x\_step(min\_x, max\_x)-max\_x))>=eps\_x)) { \*/    while ((fabs(y) >= eps\_y) && (fabs(max\_x-min\_x) >= eps\_x) && (count < MAX\_COUNT )) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  x = x\_step(min\_x, max\_x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, y, x-f(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(y) >= eps\_y) && (fabs(max\_x-min\_x) >= eps\_x)) {  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 <=a || x >=b){  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1.3：（分别用二分法与试值法求解非线性方程5）  #include <stdio.h>  #include <math.h>  double f(double x) {  return x\*x\*x - x - 1;  }  double x\_step(double a, double b){  return b - f(b)\*(b-a)/(f(b)-f(a));  }  int main(void) {  double a = 1;  double b = 2;  double min\_x = a;  double max\_x = b;  double eps\_x = 1e-5;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  // dichotomy method  double x = (min\_x + max\_x) / 2;  double y = f(x);  printf("dichotomy\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(x-min\_x) >= eps\_x) && (count <= MAX\_COUNT )) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  x = (min\_x + max\_x) / 2;  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, y, x-f(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(y) >= eps\_y) && (fabs(x-min\_x) >= eps\_x)) {  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 <=a || x >= b){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is %lf\n", x);  }  printf("\n");  printf("\n");  //trial value method  printf("trial value method\n");  min\_x = a;  max\_x = b;  count = 1;  x = x\_step(min\_x, max\_x);  y = f(x);  printf("dichotomy\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, x, y, x-f(x));  int last\_x = min\_x - max\_x;//保证初始时，上一个x不在区间内  while ((fabs(y) >= eps\_y) && (fabs(max\_x-min\_x) >= eps\_x) && (count <= MAX\_COUNT ) && (fabs((x\_step(min\_x, max\_x)-min\_x)\*(x\_step(min\_x, max\_x)-max\_x))>=eps\_x)) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  last\_x = x;  x = x\_step(min\_x, max\_x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, y, x-f(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(y) >= eps\_y) && (fabs(max\_x-min\_x) >= eps\_x) && (fabs((x\_step(min\_x, max\_x)-min\_x)\*(x\_step(min\_x, max\_x)-max\_x))>=eps\_x)) {  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 <=a || x >= b){  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2.1：（用简单迭代法求解非线性方程3）  #include <stdio.h>  #include <math.h>  double f(double x) {  return x\*x\*x + 4 \* x\*x - 10;  }  double diedai\_1(double x0){  return 0.5 \* sqrt(10 - x0\*x0\*x0);  }  double diedai\_2(double x0){  return sqrt(10/x0 - 4\*x0);  }  double diedai\_3(double x0){  return x0 - x0\*x0\*x0 - 4\*x0\*x0 + 10;  }  int main(void) {  double a = 1;  double b = 2;  double x0 = 1.5;  double eps\_x = 1e-5;  double eps\_y = 1e-4;  int MAX\_ITER = 100;    //迭代公式一  double last\_x = 999;  int iter = 0;  double x = x0;  double y = f(x);  printf("simple iteration method\_1:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(diedai\_1(x)-x) >= eps\_y) && (iter <= MAX\_ITER ) && (fabs(diedai\_1(x)-x) >= eps\_x)){  last\_x = x;  x = diedai\_1(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > MAX\_ITER) && (fabs(y) >= eps\_y) && (fabs(diedai\_1(x)-x) >= eps\_y) && (fabs(diedai\_1(x)-x) >= eps\_x)) {  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else if(x <a || x > b){  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else {  printf("root is %lf\n", x);  }  printf("\n");  //迭代公式二  last\_x = 999;  iter = 0;  x = x0;  y = f(x);  printf("simple iteration method\_2:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(diedai\_2(x)-x) >= eps\_y) && (iter <= MAX\_ITER ) && (fabs(diedai\_2(x)-x) >= eps\_x)){  last\_x = x;  x = diedai\_2(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > MAX\_ITER) && (fabs(y) >= eps\_y) && (fabs(diedai\_2(x)-x) >= eps\_y) && (fabs(diedai\_2(x)-x) >= eps\_x)) {  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else {  printf("root is %lf\n", x);  }  printf("\n");  //迭代公式三  last\_x = 999;  iter = 0;  x = x0;  y = f(x);  printf("simple iteration method\_3:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(diedai\_3(x)-x) >= eps\_y) && (iter <= MAX\_ITER ) && (fabs(diedai\_3(x)-x) >= eps\_x)){  last\_x = x;  x = diedai\_3(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > MAX\_ITER) && (fabs(y) >= eps\_y) && (fabs(diedai\_3(x)-x) >= eps\_y) && (fabs(diedai\_3(x)-x) >= eps\_x)) {  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else if(fabs(f(x))> 0){  printf("The result was not found within %d iterations.\n", MAX\_ITER);  } else {  printf("root is %lf\n", x);  }  printf("\n");    return 0;  }      实验2.2：（用简单迭代法与加速迭代法求解非线性方程5）  #include <stdio.h>  #include <math.h>  double f(double x) {  return x\*x\*x - x - 1;  }  double diedai\_1(double x0){  return x0\*x0\*x0 - 1;  }  double diedai\_2(double x0){  return diedai\_1(x0)\*diedai\_1(x0)\*diedai\_1(x0) - 1;  }  double jiasu(double x0 ){  return x0 - (diedai\_1(x0) - x0)\*(diedai\_1(x0) - x0)/(x0 - 2\*diedai\_1(x0) +diedai\_2(x0));  }  int main(void) {  double a = 1;  double b = 2;  double x0 = 1.5;  double eps\_x = 1e-5;  double eps\_y = 1e-4;  int max\_iter = 1000;    //简单迭代法  double last\_x = 999;  int iter = 0;  double x = x0;  double y = f(x);  printf("\n");  printf("simple iteration method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(diedai\_1(x)-x) >= eps\_y) && (iter <= max\_iter ) && (fabs(diedai\_1(x)-x) >= eps\_x)){  last\_x = x;  x = diedai\_1(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (fabs(y) >= eps\_y) && (fabs(diedai\_1(x)-x) >= eps\_y) && (fabs(diedai\_1(x)-x) >= eps\_x)) {  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  }  printf("\n");  //加速迭代法  last\_x = 999;  iter = 0;  x = x0;  y = f(x);  printf("\n");  printf("accelerated iteration method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(diedai\_1(x)-x) >= eps\_x) && (fabs(jiasu(x)-x) >= eps\_y) && (iter <= max\_iter ) && (fabs(diedai\_2(x)-2\*diedai\_1(x)+x) >= eps\_x)){  last\_x = x;  x = jiasu(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (fabs(diedai\_1(x)-x) >= eps\_x) && (fabs(jiasu(x)-x) >= eps\_y) && (fabs(diedai\_2(x)-2\*diedai\_1(x)+x) >= eps\_x) ) {  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  }  printf("\n");  return 0;  }      实验3.1：（用牛顿迭代法与割线法求解非线性方程3）  #include <stdio.h>  #include <math.h>  double f(double x) {  return x\*x\*x + 4\*x\*x -10;  }  double df(double x) {  return 3\*x\*x + 8\*x;  }  double newton(double x0) {  return x0 - f(x0)/df(x0);  }  double gexian(double x0, double x1) {  return x1 - f(x1)\*(x1-x0)/(f(x1)-f(x0));  }  int main() {  double a = 1;  double b = 2;    double eps\_x = 1e-5;  double eps\_y = 1e-4;  int iter = 0;  int max\_iter = 100;    // 牛顿迭代法  double x0 = 1.5;  double x = x0;  double y = f(x0);    printf("\n");  printf("Newton's iteration method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(newton(x)-x) >= eps\_x) && (fabs(df(x))>=eps\_y) && (iter <= max\_iter ) ){    x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (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\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  }  printf("\n");  //割线法  x0 = 1;  double x1 = 2;  x = x1;  y = f(x1);  iter = 0;    printf("\n");  printf("secant method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(gexian(x0,x1)-x) >= eps\_x) && (fabs((gexian(x0,x1)-x1)\*(gexian(x0,x1)-x0))>=eps\_x) && (iter <= max\_iter ) ){    x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (fabs(y) >= eps\_y) && (fabs(gexian(x0,x1)-x) >= eps\_x) && (fabs((gexian(x0,x1)-x1)\*(gexian(x0,x1)-x0))>=eps\_x)) {  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  }  printf("\n");    return 0;  }    实验3.2：（分别用牛顿迭代法与割线法求解非线性方程5）  #include <stdio.h>  #include <math.h>  double f(double x) {  return x\*x\*x - x - 1;  }  double df(double x) {  return 3\*x\*x - 1;  }  double newton(double x0) {  return x0 - f(x0)/df(x0);  }  double gexian(double x0, double x1) {  return x1 - f(x1)\*(x1-x0)/(f(x1)-f(x0));  }  int main() {  double a = 1;  double b = 2;    double eps\_x = 1e-5;  double eps\_y = 1e-4;  int iter = 0;  int max\_iter = 100;    // 牛顿迭代法  double x0 = 1.5;  double x = x0;  double y = f(x0);    printf("\n");  printf("Newton's iteration method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(newton(x)-x) >= eps\_x) && (fabs(df(x))>=eps\_y) && (iter <= max\_iter ) ){    x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (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\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  }  printf("\n");  //割线法  x0 = 1;  double x1 = 2;  x = x1;  y = f(x1);  iter = 0;    printf("\n");  printf("secant method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(gexian(x0,x1)-x) >= eps\_x) && (fabs((gexian(x0,x1)-x1)\*(gexian(x0,x1)-x0))>=eps\_x) && (iter <= max\_iter ) ){    x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (fabs(y) >= eps\_y) && (fabs(gexian(x0,x1)-x) >= eps\_x) && (fabs((gexian(x0,x1)-x1)\*(gexian(x0,x1)-x0))>=eps\_x)) {  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  }  printf("\n");    return 0;  }    实验3.3：（分别用牛顿迭代法与割线法求解非线性方程2）  #include <stdio.h>  #include <math.h>  double f(double x) {  return exp(-100\*x) - 1;  }  double df(double x) {  return -100\*exp(-100\*x);  }  double newton(double x0) {  return x0 - f(x0)/df(x0);  }  double gexian(double x0, double x1) {  return x1 - f(x1)\*(x1-x0)/(f(x1)-f(x0));  }  int main() {  double a = -0.51;  double b = 0.49;    double eps\_x = 1e-5;  double eps\_y = 1e-4;  int iter = 0;  int max\_iter = 100;    // 牛顿迭代法  double x0 = -0.51;  double x = x0;  double y = f(x0);    printf("\n");  printf("Newton's iteration method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(newton(x)-x) >= eps\_x) && (fabs(df(x))>=eps\_y) && (iter <= max\_iter ) ){    x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (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\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  }  printf("\n");    //割线法  iter = 0;  x0 = -0.51;  double x1 = 0.49;  x = x1;  y = f(x1);    printf("\n");  printf("secant method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));    /\* while ((fabs(y) >= eps\_y) && (fabs(gexian(x0,x1)-x) >= eps\_x) && (fabs((gexian(x0,x1)-x1)\*(gexian(x0,x1)-x0))>=eps\_x) && (iter <= max\_iter ) ){    x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (fabs(y) >= eps\_y) && (fabs(gexian(x0,x1)-x) >= eps\_x) && (fabs((gexian(x0,x1)-x1)\*(gexian(x0,x1)-x0))>=eps\_x)) {  printf("The result was not found within %d iterations.\n", max\_iter);  } \*/    while ((fabs(y) >= eps\_y) && (iter <= max\_iter ) ){    x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (fabs(y) >= eps\_y) ) {  printf("The result was not found within %d iterations.\n", max\_iter);  }    else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >=b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  }  printf("\n");    return 0;  }        实验4.1：（获取下面方程的一个解）  //割线法为 当不知函数的导数 而无法使用牛顿法时的平替  //此方程已知函数形式 且导数易得 故可使用牛顿法求根  //为防止牛顿法在极值点附近无限次徘徊的情况  //在外边包一层二分法  #include <stdio.h>  #include <math.h>  const double pi = 3.14159265358979323846264338327950288419716939937510;  double f(double x) {  return x + 4.0 \* pi \* sin(pi\*x);  }  double df(double x) {  return 1.0 + 4.0 \* pi \* pi \* cos(pi\*x);  }  double newton(double x0) {  return x0 - f(x0)/df(x0);  }  double mid(double x0, double x1) {  return (x0 + x1) / 2.0;  }  void x(double x0) {  printf("When x0 is %lf\n", x0);  double a = -3;  double b = 4;    double eps\_x = 1e-5;  double eps\_y = 1e-5;  int iter = 0;  int max\_iter\_mid = 60;  int max\_iter\_newton = 40;  int max\_iter = max\_iter\_mid + max\_iter\_newton;  // 牛顿迭代法    double x = x0;  double y = f(x0);  double min\_x = a;  double max\_x = b;    printf("\n");  printf("Use Newton's method first, then use the bisection method.:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));    int result = 0;    while ((fabs(y) >= eps\_y) && (fabs(newton(x)-x) >= eps\_x) && (fabs(df(x))>=eps\_y) ){  x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  if (iter+1 > max\_iter\_newton) {  result = 0;  break;  } else if ((fabs(y) < eps\_y) || (fabs(newton(x)-x) < eps\_x) || (fabs(df(x))<eps\_y) ){  result = 1;  if(x <=a || x >= b) result = 0;  break;  }  }  if (result == 0) printf("Use dichotomy from here\n");  // 二分法  if (result == 0) {  max\_x = b;  min\_x = a;  x0 = -3;  double x1 = 4;      x = mid(x0, x1);  y = f(x);    printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(x-min\_x) >= eps\_x) ) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  x = (min\_x + max\_x) / 2;  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  if(iter-1 > max\_iter\_mid) {  result = 0;  break;  } else if ((fabs(y) < eps\_y) || (fabs(x-min\_x) < eps\_x) ){  result = 1;  break;  }  }  }          printf("--------------------------\n");  if((result == 0) ) {  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  printf("x = %lf f(x) = %lf\n", x, f(x));  printf("\n");  }  }  int main(void) {  x(4.0);  x(-3.0);  return 0;  }  //割线法为 当不知函数的导数 而无法使用牛顿法时的平替  //此方程已知函数形式 且导数易得 故可使用牛顿法求根  //为防止牛顿法在极值点附近无限次徘徊的情况  //在外边包一层二分法  #include <stdio.h>  #include <math.h>  const double pi = 3.14159265358979323846264338327950288419716939937510;  double f(double x) {  return x + 4.0 \* pi \* sin(pi\*x);  }  double df(double x) {  return 1.0 + 4.0 \* pi \* pi \* cos(pi\*x);  }  double newton(double x0) {  return x0 - f(x0)/df(x0);  }  double mid(double x0, double x1) {  return (x0 + x1) / 2.0;  }  void x(double x0) {  printf("When x0 is %lf\n", x0);  double a = -3;  double b = 4;    double eps\_x = 1e-5;  double eps\_y = 1e-5;  int iter = 0;  int max\_iter\_mid = 60;  int max\_iter\_newton = 40;  int max\_iter = max\_iter\_mid + max\_iter\_newton;  // 牛顿迭代法    double x = x0;  double y = f(x0);  double min\_x = a;  double max\_x = b;    printf("\n");  printf("Use Newton's method first, then use the bisection method.:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));    int result = 0;    while ((fabs(y) >= eps\_y) && (fabs(newton(x)-x) >= eps\_x) && (fabs(df(x))>=eps\_y) ){  x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  if (iter+1 > max\_iter\_newton) {  result = 0;  break;  } else if ((fabs(y) < eps\_y) || (fabs(newton(x)-x) < eps\_x) || (fabs(df(x))<eps\_y) ){  result = 1;  if(x <=a || x >= b) result = 0;  break;  }  }  if (result == 0) printf("Use dichotomy from here\n");  // 二分法  if (result == 0) {  max\_x = b;  min\_x = a;  x0 = -3;  double x1 = 4;      x = mid(x0, x1);  y = f(x);    printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(x-min\_x) >= eps\_x) ) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  x = (min\_x + max\_x) / 2;  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  if(iter-1 > max\_iter\_mid) {  result = 0;  break;  } else if ((fabs(y) < eps\_y) || (fabs(x-min\_x) < eps\_x) ){  result = 1;  break;  }  }  }          printf("--------------------------\n");  if((result == 0) ) {  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  printf("x = %lf f(x) = %lf\n", x, f(x));  printf("\n");  }  }  int main(void) {  x(4.0);  x(-3.0);  return 0;  }      接下来是新的实验  老师给了一个具有约束的函数，通过插值法求得其函数表达式后，使用二分法，试值法，牛顿法，割线法分别求其零点  一：  函数生成：  #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;  }    我们发现函数满足我们的约束  二：  二分法与试值法  #include <stdio.h>  #include <stdlib.h>  #include <math.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);  }  double x\_step(double a, double b){  return b - f(b)\*(b-a)/(f(b)-f(a));  }  int main() {  double a = 0;  double b = 6;  double min\_x = a;  double max\_x = b;  double eps\_x = 1e-5;  double eps\_y = 1e-4;  int count = 1;  int MAX\_COUNT = 100;  // dichotomy method  double x = (min\_x + max\_x) / 2;  double y = f(x);  printf("dichotomy\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(x-min\_x) >= eps\_x) && (count <= MAX\_COUNT )) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  x = (min\_x + max\_x) / 2;  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, y, x-f(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(y) >= eps\_y) && (fabs(x-min\_x) >= eps\_x)) {  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 <=a || x >= b){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is %lf\n", x);  }  printf("\n");  //trial value method  printf("trial value method\n");  min\_x = a;  max\_x = b;  count = 1;  x = x\_step(min\_x, max\_x);  y = f(x);  printf("dichotomy\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", count, x, y, x-f(x));  int last\_x = min\_x - max\_x;//保证初始时，上一个x不在区间内  while ((fabs(y) >= eps\_y) && (fabs(max\_x-min\_x) >= eps\_x) && (count <= MAX\_COUNT ) && (fabs((x\_step(min\_x, max\_x)-min\_x)\*(x\_step(min\_x, max\_x)-max\_x))>=eps\_x)) {  if (f(x) \* f(min\_x) < 0) {  max\_x = x;  } else {  min\_x = x;  }  last\_x = x;  x = x\_step(min\_x, max\_x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", count+1, x, y, x-f(x));  count++;  }  printf("--------------------------\n");  if((count > MAX\_COUNT) && (fabs(y) >= eps\_y) && (fabs(max\_x-min\_x) >= eps\_x) && (fabs((x\_step(min\_x, max\_x)-min\_x)\*(x\_step(min\_x, max\_x)-max\_x))>=eps\_x)) {  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 <=a || x >= b){  printf("The result was not found within %d iterations.\n", MAX\_COUNT);  } else {  printf("root is %lf\n", x);  }  printf("\n");    return 0;  }    三：  牛顿法，割线法  #include <stdio.h>  #include <stdlib.h>  #include <math.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);  }  double newton(double x0) {  return x0 - f(x0)/df(x0);  }  double gexian(double x0, double x1) {  return x1 - f(x1)\*(x1-x0)/(f(x1)-f(x0));  }  int main(void) {  double a = -0.2;  double b = 6.1;    double eps\_x = 1e-5;  double eps\_y = 1e-4;  int iter = 0;  int max\_iter = 100;    // 牛顿迭代法  double x0 = 4;  double x = x0;  double y = f(x0);    printf("\n");  printf("Newton's iteration method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(newton(x)-x) >= eps\_x) && (fabs(df(x))>=eps\_y) && (iter <= max\_iter ) ){    x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (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\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  }  printf("\n");  //割线法  x0 = 4;  double x1 = 4.5;  x = x1;  y = f(x1);  iter = 0;    printf("\n");  printf("secant method:\n");  printf("--------------------------\n");  printf("iter\tx\t\tf(x)\t\tx-f(x)\n");  printf("%d\t%lf\t%lf\t%lf\n", iter, x, y, x-f(x));  while ((fabs(y) >= eps\_y) && (fabs(gexian(x0,x1)-x) >= eps\_x) && (fabs((gexian(x0,x1)-x1)\*(gexian(x0,x1)-x0))>=eps\_x) && (iter <= max\_iter ) ){    x = newton(x);  y = f(x);  printf("%d\t%lf\t%lf\t%lf\n", iter+1, x, y, x-f(x));  iter++;  }  printf("--------------------------\n");  if((iter > max\_iter) && (fabs(y) >= eps\_y) && (fabs(gexian(x0,x1)-x) >= eps\_x) && (fabs((gexian(x0,x1)-x1)\*(gexian(x0,x1)-x0))>=eps\_x)) {  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(x)){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(isnan(f(x))){  printf("The result was not found within %d iterations.\n", max\_iter);  } else if(x <=a || x >= b){  printf("The result was not found within %d iterations.\n", max\_iter);  } else {  printf("root is %lf\n", x);  }  printf("\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。  通过实验1.1，1.2，1.3我比较了二分法与试值法的区别，了解到  二分法具有普遍适用的特性，但他也无法做到加速迭代过程，  试值法在目标函数具有接近线性的情况下，可以有效加速迭代过程，但若不满足接近线性的假设，反而可能不如二分法高效  通过实验2.1，我了解到，不同的迭代公式对迭代的影响有显著的不同，若选取合适的迭代公式，可能很快就可以有结果，但若选取不合适，则可能需要很久，甚至是无解  通过实验2.2，我尝试使用加速迭代法优化问题，发现，加速迭代法在很多情况下可以有效加速迭代的过程，并且有时可以将原来简单迭代法无法求解的问题解决  通过实验3.1，3.2，3.3我比较了牛顿法与割线法的区别，割线法是牛顿法的近似，在很多问题上他们两个的表现相差不大：如3.1与3.2，迭代都在4步内完成，而有时候牛顿法会比割线法好一些，如3.3，牛顿法可以解，但割线法无法求解；但割线法也有其特有的优点，由于牛顿法要求函数可微，若不具备此性质，就只能使用割线法求解  通过实验4.1，我对一个新问题进行了求解，其算法的主要逻辑是在牛顿法外包一层二分法，当牛顿法在所设定步数内没有得到解，则使用二分法求解。  通过对 用插值法求得的函数分别使用上述方法求解 我了解到  牛顿法与割线法及试值法在目标函数有极值点，且该极值点不是最值点的时候，可能会由于初值点选取的不同而造成无法求解的情况，而二分法则不存在此问题 | | | |
|  | | | |

注：若填写内容较多，可在背面继续填写。