# 例题的程序

[例题的程序 1](#_Toc418707493)

[第二章 2](#_Toc418707497)

[Egch2\_sec1\_01.c 2](#_Toc418707498)

[Egch2\_sec1\_02.c 3](#_Toc418707499)

[Egch2\_sec1\_03.c 5](#_Toc418707500)

[Egch2\_sec3\_02.c 6](#_Toc418707501)

[Egch2\_sec4\_03.c 7](#_Toc418707502)

[Egch2\_sec5\_01.c 9](#_Toc418707503)

[Egch2\_sec5\_02.c 11](#_Toc418707504)

[Egch2\_sec6\_01.c 12](#_Toc418707505)

[Egch2\_sec6\_02.c 15](#_Toc418707506)

[Egch2\_sec6\_03.c 18](#_Toc418707507)

[Egch2\_sec6\_04.c 20](#_Toc418707508)

[Egch2\_sec6\_05.c 23](#_Toc418707509)

[Egch2\_sec7\_01.c 25](#_Toc418707510)

[Egch2\_sec7\_02.c 26](#_Toc418707511)

[第三章 29](#_Toc418707512)

[Egch3\_sec1\_01.c 29](#_Toc418707513)

[Egch3\_sec1\_02.c 31](#_Toc418707514)

[Egch3\_sec2\_01.c 33](#_Toc418707515)

[Egch3\_sec2\_02.c 37](#_Toc418707516)

[Egch3\_sec3\_01.c 40](#_Toc418707517)

[Egch3\_sec3\_02.c 42](#_Toc418707518)

[Egch3\_sec3\_03.c 45](#_Toc418707519)

[Egch3\_sec4\_01.c 49](#_Toc418707520)

[Egch3\_sec4\_02.c 49](#_Toc418707521)

[Egch3\_sec4\_03.c 50](#_Toc418707522)

[Egch3\_sec4\_04.c 54](#_Toc418707523)

[Egch3\_sec5\_01.c 57](#_Toc418707524)

[Egch3\_sec5\_02.c 59](#_Toc418707525)

[Egch3\_sec5\_03.c 61](#_Toc418707526)

[Egch3\_sec6\_01.c 65](#_Toc418707527)

[Egch3\_sec6\_02.c 70](#_Toc418707528)

[Egch3\_sec6\_03.c 76](#_Toc418707529)

[Egch3\_sec7\_01.c 81](#_Toc418707530)

[第四章 87](#_Toc418707531)

[Egch4\_sec1\_01.c 87](#_Toc418707532)

[Egch4\_sec1\_02.c 90](#_Toc418707533)

[Egch4\_sec1\_03.c 92](#_Toc418707534)

[Egch4\_sec1\_04.c 94](#_Toc418707535)

[Egch4\_sec2\_01.c 96](#_Toc418707536)

[Egch4\_sec2\_02.c 99](#_Toc418707537)

[Egch4\_sec3\_01.c 101](#_Toc418707538)

[Egch4\_sec4\_01.c 105](#_Toc418707539)

[Egch4\_sec5\_01.c 108](#_Toc418707540)

[Egch4\_sec6\_01.c 114](#_Toc418707541)

[第五章 121](#_Toc418707542)

[Egch5\_sec1\_01.c 121](#_Toc418707543)

[Egch5\_sec2\_01.c 124](#_Toc418707544)

[Egch5\_sec3\_01.c 128](#_Toc418707545)

[第六章 132](#_Toc418707546)

[Egch6\_sec1\_01.c 132](#_Toc418707547)

[Egch6\_sec4\_01.c 137](#_Toc418707548)

[Egch6\_sec5\_01.c 145](#_Toc418707549)

## 第二章

Egch2\_sec1\_01.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

int N,i;

double a,b,h,\*x,\*y,y0,err,maxerr;

double f(double x, double y);

double yexact(double x);

a=0.0; //求解区域的左端点

b=1.0; //求解区域的右端点

N=10; //总的剖分数

h=(b-a)/N; //步长

x=(double \*)malloc(sizeof(double)\*(N+1)); //动态分配长度为(N+1)的数组，存放节点

坐标

y=(double \*)malloc(sizeof(double)\*(N+1)); //动态分配长度为(N+1)的数组，存放对应

节点的数值解

for(i=0;i<=N;i++)

x[i]=a+i\*h; //节点坐标

y0=1.0; //初值

y[0]=y0; //初值

maxerr=0.0;

for(i=0;i<N;i++)

{

y[i+1]=y[i]+h\*f(x[i],y[i]); //欧拉方法

err=fabs(y[i+1]-yexact(x[i+1])); //计算节点处的误差

printf("x[%d]=%.4f, y[%d]=%f, exact=%f, err=%f\n",i+1,x[i+1],i+1,y[i

+1],yexact(x[i+1]),err); //打印节点及在这个节点上的数值解、精确解和误差

if(err>maxerr)

maxerr=err;

}

printf("The max error is %f\n", maxerr); //打印最大误差

}

double f(double x, double y) //右端项函数

{

return y-2\*x/y;

}

double yexact(double x) //精确解y

{

return sqrt(1.0+2\*x);

}

Egch2\_sec1\_02.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

int N,i,k;

double a,b,h,\*x,\*y,y0,ytemp1,ytemp2,epsilon, err,maxerr;

double f(double x, double y);

double yexact(double x);

a=0.0; //求解区域的左端点

b=1.0; //求解区域的右端点

N=10; //总的剖分数

h=(b-a)/N; //步长

x=(double \*)malloc(sizeof(double)\*(N+1)); //动态分配长度为(N+1)的数组，存放节点坐标

y=(double \*)malloc(sizeof(double)\*(N+1)); //动态分配长度为(N+1)的数组，存放对应节点的数值解

for(i=0;i<=N;i++)

x[i]=a+i\*h; //节点坐标

y0=1.0; //初值

y[0]=y0; //初值

maxerr=0.0;

epsilon=1e-4;

for(i=0;i<N;i++)

{

ytemp1=y[i]+h\*f(x[i],y[i]);

k=0;

do

{

k=k+1;

if(k!=1)

ytemp1=ytemp2;

ytemp2=y[i]+h\*(f(x[i],y[i])+f(x[i+1],ytemp1))/2.0;

}

while(fabs(ytemp1-ytemp2)>epsilon);

y[i+1]=ytemp2;

err=fabs(y[i+1]-yexact(x[i+1])); //计算节点处的误差

printf("k=%d, x[%d]=%.4f, y[%d]=%f, exact=%f, err=%f\n",k, i+1,x[i+1],i+1,y[i+1],yexact(x[i+1]),err); //打印节点及在这个节点上的数值解、精确解和误差

if(err>maxerr)

maxerr=err;

}

printf("The max error is %f\n", maxerr); //打印最大误差

}

double f(double x, double y) //右端项函数

{

return y-2\*x/y;

}

double yexact(double x) //精确解y

{

return sqrt(1.0+2\*x);

}

Egch2\_sec1\_03.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

int N,i,k;

double a,b,h,\*x,\*y,y0,y\_predict,err,maxerr;

double f(double x, double y);

double yexact(double x);

a=0.0; //求解区域的左端点

b=1.0; //求解区域的右端点

N=10; //总的剖分数

h=(b-a)/N; //步长

x=(double \*)malloc(sizeof(double)\*(N+1)); //动态分配长度为(N+1)的数组，存放节点坐标

y=(double \*)malloc(sizeof(double)\*(N+1)); //动态分配长度为(N+1)的数组，存放对应节点的数值解

for(i=0;i<=N;i++)

x[i]=a+i\*h; //节点坐标

y0=1.0; //初值

y[0]=y0; //初值

maxerr=0.0;

for(i=0;i<N;i++)

{

y\_predict=y[i]+h\*f(x[i],y[i]);

y[i+1]=y[i]+h\*(f(x[i],y[i])+f(x[i+1],y\_predict))/2.0;

err=fabs(y[i+1]-yexact(x[i+1])); //计算节点处的误差

printf("x[%d]=%.4f, y[%d]=%f, exact=%f, err=%f\n",i+1,x[i+1],i+1,y[i+1],yexact(x[i+1]),err); //打印节点及在这个节点上的数值解、精确解和误差

if(err>maxerr)

maxerr=err;

}

printf("The max error is %f\n", maxerr); //打印最大误差

}

double f(double x, double y) //右端项函数

{

return y-2\*x/y;

}

double yexact(double x) //精确解y

{

return sqrt(1.0+2\*x);

}

Egch2\_sec3\_02.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

double a, b, x0, x1, y0, y1, yexact, h, err, k1, k2, k3, k4;

int i, N;

double exact(double x);

double f(double x, double y);

a = 0.0;

b = 1.0;

N = 5;

h = (b - a) / N;

x0 = a;

y0 = 1.0;

i = 1;

do

{

x1 = x0 + h;

k1 = h\*f(x0, y0);

k2 = h\*f(x0 + 0.5\*h, y0 + 0.5\*k1);

k3 = h\*f(x0 + 0.5\*h, y0 + 0.5\*k2);

k4 = h\*f(x1, y0 + k3);

y1 = y0 + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6.0;

yexact = exact(x1);

err = fabs(yexact - y1);

printf("x=%.2f, num\_solution=%f, exact=%f, error=%f\n", x1, y1, yexact, err);

i++;

x0 = x1;

y0 = y1;

} while (i <= N);

}

double exact(double x)

{

double z;

z = sqrt(1 + 2 \* x);

return z;

}

double f(double x, double y)

{

double z;

z = y - 2 \* x / y;

return z;

}

Egch2\_sec4\_03.c

#include "stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main()

{

double a,b,h,\*x,\*y,\*u, k1,k2,k3,k4,u\_predict;

int i, N;

double exact(double x);

double f(double x, double y);

a = 0.0;

b= 1.0;

N= 10;

h = (b - a) / N;

x = (double \*)malloc(sizeof(double)\*(N + 1));

for (i = 0; i <= N; i++)

x[i] = a + i\*h;

y = (double \*)malloc(sizeof(double)\*(N + 1));

u = (double \*)malloc(sizeof(double)\*(N + 1));

y[0] = 1.0;

u[0] = 1.0;

for (i = 0; i <= 2; i++) //用龙格－库塔方法求y1,y2,y3作为初始值

{

k1 = h\*f(x[i], y[i]);

k2 = h\*f(x[i] + 0.5\*h, y[i] + 0.5\*k1);

k3 = h\*f(x[i] + 0.5\*h, y[i] + 0.5\*k2);

k4 = h\*f(x[i] + h, y[i] + k3);

y[i + 1] = y[i] + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6.0;

u[i + 1] = y[i + 1];

}

for (i = 3; i < N; i++)

{

y[i + 1] = y[i] + (55 \* f(x[i], y[i]) - 59 \* f(x[i - 1], y[i - 1]) + 37 \* f(x[i - 2], y[i - 2]) - 9 \* f(x[i - 3], y[i - 3]))\*h / 24.0; //四阶显格式

u\_predict = u[i] + (55 \* f(x[i], u[i]) - 59 \* f(x[i - 1], u[i - 1]) + 37 \* f(x[i - 2], u[i - 2]) - 9 \* f(x[i - 3], u[i - 3]))\*h / 24.0;

u[i + 1] = u[i] + (9 \* f(x[i + 1], u\_predict) + 19 \* f(x[i], u[i]) - 5 \* f(x[i - 1], u[i - 1]) + f(x[i - 2], u[i - 2]))\*h / 24; //预估－校正格式

}

for (i = 0; i <= N; i++)

{

printf(" x=%.2f, 显式解=%f, 预估－校正解=%f, exact=%f\n", x[i], y[i], u[i],exact(x[i]));

}

free(x); free(y); free(u);

}

double exact(double x)

{

double z;

z = sqrt(1+2\*x);

return z;

}

double f(double x, double y)

{

double z;

z =y-2\*x/y;

return z;

}

Egch2\_sec5\_01.c

#include "stdafx.h"

#include "math.h"

#include "stdio.h"

#include "stdlib.h"

void main()

{

int m, i;

double a,b,h, \*x, \*y1, \*y2,temp1,temp2,y1\_predict, y2\_predict;

double f1(double x, double y1, double y2);

double f2(double x, double y1, double y2);

double exact1(double x);

double exact2(double x);

a = 0.0;

b = 1.0;

m = 20;

h = (b - a) / m;

x = (double \*)malloc(sizeof(double)\*(m + 1));

y1 = (double \*)malloc(sizeof(double)\*(m + 1));

y2 = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

{

x[i] = a + i\*h;

}

y1[0] = 1.0;

y2[0] = 0.0;

for (i = 0; i <= m; i++)

{

temp1 = f1(x[i], y1[i], y2[i]);

temp2 = f2(x[i], y1[i], y2[i]);

y1\_predict = y1[i] + h\*temp1;

y2\_predict = y2[i] + h\*temp2;

y1[i + 1] = y1[i] + 0.5\*h\*(temp1 + f1(x[i + 1], y1\_predict, y2\_predict));

y2[i + 1] = y2[i] + 0.5\*h\*(temp2 + f2(x[i + 1], y1\_predict, y2\_predict));

}

for (i = 0; i <= m; i++)

{

printf("x[%d]=%.2f, y1=%f, exacty1=%f，err1=%f\n", i, x[i], y1[i],exact1(x[i]),fabs(y1[i]-exact1(x[i])));

printf(" y2=%f, exacty2=%f, err2=%f\n", y2[i], exact2(x[i]), fabs(y2[i] - exact2(x[i])));

}

}

double f1(double x, double y1, double y2)

{

return y2;

}

double f2(double x, double y1, double y2)

{

return -y1;

}

double exact1(double x)

{

return cos(x);

}

double exact2(double x)

{

return -sin(x);

}

Egch2\_sec5\_02.c

#include "stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main()

{

double a,b,h,\*x, y1, y2, \*uex, \*k1, \*k2, \*k3, \*k4,err1,err2;

int i, m;

double \* exact(double x);

double \* rhsf(double x, double y1, double y2);

a = 0.0;

b = 1.0;

m = 10;

h = (b - a) / m;

x = (double \*)malloc(sizeof(double)\* (m+1));

for (i = 0; i <= m; i++)

x[i] = a + i\*h;

y1 = 1.0;

y2 = 0.0;

for (i = 0; i<m; i++)

{

k1 = rhsf(x[i], y1, y2);

k2 = rhsf(x[i] + 0.5\*h, y1 + 0.5\*h\*k1[0], y2 + 0.5\*h\*k1[1]);

k3 = rhsf(x[i] + 0.5\*h, y1 + 0.5\*h\*k2[0], y2 + 0.5\*h\*k2[1]);

k4 = rhsf(x[i + 1], y1 + h\*k3[0], y2 + h\*k3[1]);

y1 = y1 + h\*(k1[0] + 2 \* k2[0] + 2 \* k3[0] + k4[0]) / 6.0;

y2 = y2 + h\*(k1[1] + 2 \* k2[1] + 2 \* k3[1] + k4[1]) / 6.0;

uex = exact(x[i + 1]);

err1 = fabs(uex[0] - y1);

err2 = fabs(uex[1] - y2);

free(k1); free(k2); free(k3); free(k4);

printf("x[%d]=%.2f, y1=%f, exacty1=%f, err1=%e\n", i+1, x[i+1],y1, uex[0],err1);

printf(" y2=%f, exacty2=%f, err2=%e\n", y2, uex[1], err2);

free(uex);

}

}

double \* exact(double x)

{

double \*z;

z = (double \*)malloc(sizeof(double)\* 2);

z[0] = cos(x);

z[1] = -sin(x);

return z;

}

double \* rhsf(double x, double y1, double y2)

{

double \*z;

z = (double \*)malloc(sizeof(double)\* 2);

z[0] = y2;

z[1] =-y1;

return z;

}

Egch2\_sec6\_01.c

//Egch2\_sec6\_01.c : 定义控制台应用程序的入口点。

// 这个一个打靶法的程序,这个程序用来求解常微分方程两点Dirichlet边值问题

//y''+p(x)y'+q(x)y=f(x), a<x<b,

//y(a)=alpha, y(b)=beta 。

//其本质是将边值问题化为等价的初值问题，初值问题用龙格库塔方法降阶为一阶方程组求解。

#include "stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

int m;

double h;

void main()

{

double a, b, alpha, beta, gamma;

int i;

double \*x, \*y1, \*y2, \*y3, \*y;

double p(double x); //原始方程中的函数p(x)

double q(double x); //原始方程中的函数q(x)

double f(double x); //原始方程中的右端函数f(x)

double \*rhsf(int k, double x, double y1, double y2); //初值问题降阶以后得到的右端项函数组

double \*RKhigher(int k, double \*x, double c, double d);

//三个参数k,c,d分别用来描述高阶初值问题的右端项函数，初值点值及初值导数值。

double exact(double x);

a = 0.0; //a,b分别是两点边值问题中的两个边界端点。

b = 1.0;

m = 10; //将区间[a,b]等分成 m 份

h = (b - a) / m; //m等分后，步长为h

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = a + i\*h;

alpha = 0.0; //alpha,beta为两点边界条件

beta = 0.0;

y1 = RKhigher(0, x, 1, 0); //解初值问题(2-72)

y2 = RKhigher(0, x, 0, 1); //解初值问题(2-73)

y3 = RKhigher(1, x, 0, 0); //解初值问题(2-74)

y = (double \*)malloc(sizeof(double)\*(m + 1));

gamma = (beta - alpha\*y1[m] - y3[m]) / y2[m]; //求出待定的gamma

for (i = 0; i <= m; i++)

{

y[i] = alpha\*y1[i] + gamma\*y2[i] + y3[i];

printf("x[%d]=%.2f, ynumerical=%f, exact=%f, error=%.4e\n", i, x[i], y[i], exact(x[i]), fabs(exact(x[i]) - y[i]));

}

}

double p(double x)

{

return 0.0;

}

double q(double x)

{

return 1.0;

}

double f(double x)

{

return -1.0;

}

//对y''+p(x)y'+q(x)y=f(x)进行降阶后，成为如下一阶方程组

//y1’=y2, y2’=f(x)-p(x)y2-q(x)y1

//所以在对（2－72）（2－73）（2－74）降阶后得到的右端项分别取参数k=0,0,1.

double \* rhsf(int k, double x, double y1, double y2)

{

double \*z;

z = (double \*)malloc(sizeof(double)\* 2);

z[0] = y2;

z[1] = k\*f(x) - p(x)\*y2 - q(x)\*y1;

return z;

}

//参数k用来确定三个初值问题中的哪个，参数c,d分别确定初值和初始导数值

double \*RKhigher(int k, double \*x, double c, double d)

{

int i;

double \*y, y1, y2,\*k1,\*k2,\*k3,\*k4;

y = (double \*)malloc(sizeof(double)\*(m + 1));

y[0] = c;

y1 = c;

y2 = d;

for (i = 0; i<m; i++)

{

k1 = rhsf(k, x[i], y1, y2);

k2 = rhsf(k, x[i] + 0.5\*h, y1 + 0.5\*h\*k1[0], y2 + 0.5\*h\*k1[1]);

k3 = rhsf(k, x[i] + 0.5\*h, y1 + 0.5\*h\*k2[0], y2 + 0.5\*h\*k2[1]);

k4 = rhsf(k, x[i + 1], y1 + h\*k3[0], y2 + h\*k3[1]);

y1 = y1 + h\*(k1[0] + 2 \* k2[0] + 2 \* k3[0] + k4[0]) / 6.0;

y2 = y2 + h\*(k1[1] + 2 \* k2[1] + 2 \* k3[1] + k4[1]) / 6.0;

y[i + 1] = y1;

free(k1); free(k2); free(k3); free(k4);

}

return y;

}

double exact(double x)

{

return cos(x) + (1 - cos(1.0))\*sin(x) / sin(1.0) - 1;

}

Egch2\_sec6\_02.c

// Egch2\_sec6\_02.cpp : 定义控制台应用程序的入口点。

// 这个一个打靶法的程序，用来求解常微分方程两点混合边值问题

// y''+p(x)y'+q(x)y=rhs(x), a<x<b;

// y’(a)+lambda\*y(a)=alpha, y’(b)+mu\*y(b)=beta。

//本质是将边值问题化为等价的初值问题，初值问题用龙格库塔方法降阶为一阶方程组求解。

#include "stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

int m;

double h;

#define pi 3.14159265359

void main()

{

double a, b, alpha, beta, lambda, mu, s, t, eta1, eta2, eta3, temp;

int i;

double \*x;

double \*y1, \*y2, \*y3, \*y;

double p(double x); //原始方程中的函数p(x)

double q(double x); //原始方程中的函数q(x)

double f(double x); //原始方程中的右端函数f(x)

double \*rhsf(int k, double x, double y1, double y2);

double \*RKhigher(int k, double \*x, double c, double d);

double exact(double x);

a = 0.0; //a,b分别是两点边值问题中的两个边界端点。

b = pi;

m = 10; //将区间[a,b]等分成 m 份

h = (b - a) / m; //m等分后，步长为h

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = a + i\*h;

lambda = -2.0;

mu = 1.0;

alpha = 1.0;

beta = -exp(pi);

y1 = RKhigher(0, x, 1, 0);

y2 = RKhigher(0, x, 0, 1);

y3 = RKhigher(1, x, 0, 0);

eta1 = y1[2 \* m + 1] + mu\*y1[m]; //即z^1\_m+mu\*y^1\_m

eta2 = y2[2 \* m + 1] + mu\*y2[m];

eta3 = y3[2 \* m + 1] + mu\*y3[m];

temp = lambda\*eta2 - eta1;

s = (alpha\*eta2 - beta + eta3) / temp;

t = (lambda\*beta - alpha\*eta1 - lambda\*eta3) / temp;

y = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

{

y[i] = s\*y1[i] + t\*y2[i] + y3[i];

printf("x[%d]=%.2f, ynumerical=%f, exact=%f, error=%.4e\n", i, x[i], y[i], exact(x[i]), fabs(exact(x[i]) - y[i]));

}

}

double p(double x)

{

return x;

}

double q(double x)

{

return -x;

}

double f(double x)

{

return exp(x)\*cos(x)\*(2 + x);

}

double \* rhsf(int k, double x, double y1, double y2)

{

double \*z;

z = (double \*)malloc(sizeof(double)\* 2);

z[0] = y2;

z[1] = k\*f(x) - p(x)\*y2 - q(x)\*y1;

return z;

}

//参数k用来确定三个初值问题中的哪个，参数c,d分别确定初值和初始导数值

double \*RKhigher(int k, double \*x, double c, double d)

{

int i;

double \*y, \*z, \*w, y1, y2, \*k1, \*k2, \*k3, \*k4;

y = (double \*)malloc(sizeof(double)\*(m + 1)); //存放y值

z = (double \*)malloc(sizeof(double)\*(m + 1)); //存放y的导数值

y[0] = c;

z[0] = d;

y1 = c;

y2 = d;

for (i = 0; i<m; i++)

{

k1 = rhsf(k, x[i], y1, y2);

k2 = rhsf(k, x[i] + 0.5\*h, y1 + 0.5\*h\*k1[0], y2 + 0.5\*h\*k1[1]);

k3 = rhsf(k, x[i] + 0.5\*h, y1 + 0.5\*h\*k2[0], y2 + 0.5\*h\*k2[1]);

k4 = rhsf(k, x[i + 1], y1 + h\*k3[0], y2 + h\*k3[1]);

y1 = y1 + h\*(k1[0] + 2 \* k2[0] + 2 \* k3[0] + k4[0]) / 6.0;

y2 = y2 + h\*(k1[1] + 2 \* k2[1] + 2 \* k3[1] + k4[1]) / 6.0;

y[i + 1] = y1;

z[i + 1] = y2;

free(k1); free(k2); free(k3); free(k4);

}

w = (double \*)malloc(sizeof(double)\*(2 \* m + 2));

for (i = 0; i <= m; i++)

{

w[i] = y[i];

w[m + 1 + i] = z[i];

}

free(y); free(z);

return w;

}

double exact(double x)

{

return exp(x)\*sin(x);

}

Egch2\_sec6\_03.c

//Egch2\_sec6\_03.cpp : 定义控制台应用程序的入口点。

//本程序是求解二阶线性常微分方程两点Dirichlet边值问题的：

//y’’+q(x)y=f(x), a<x<b, y(a)=alpha, y(b)=beta

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

double \*matrix\_a\_array, \*matrix\_b\_array, \*x, \*rhs, \*ans, \*y;

double a, b, h, Pi, alpha, beta;

int i, j, m;

double f(double x); //原方程右端项函数f(x)

double q(double x); //原方程中的函数q(x)

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

//追赶法子程序，a,b,c分别为系数矩阵的下次、主、上次对角线元素组，n为方程组阶数，d为矩阵右端项向量

double exact(double x);

m = 8;

Pi = 3.14159265359;

a = 0.0; //边界左端点

b = Pi / 2.0; //边界右端点

h = (b - a) / m;

alpha = 0.0;

beta = 1.0;

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

{

x[i] = a + i\*h;

}

rhs = (double \*)malloc(sizeof(double)\*(m - 1));

for (i = 1; i<m; i++) //矩阵的右端项，含m-1个元素的数组rhs

{

rhs[i - 1] = h\*h\*f(x[i]);

}

rhs[0] = rhs[0] - alpha; //考虑边界条件

rhs[m - 2] = rhs[m - 2] - beta;

matrix\_a\_array = (double \*)malloc(sizeof(double)\*(m - 1)); //矩阵的下次对角线

matrix\_b\_array = (double \*)malloc(sizeof(double)\*(m - 1)); //矩阵的主对角线

for (i = 0; i<m - 1; i++)

{

matrix\_a\_array[i] = 1.0;

matrix\_b\_array[i] = h\*h\*q(x[i + 1]) - 2;

}

ans = chase\_algorithm(matrix\_a\_array, matrix\_b\_array, matrix\_a\_array, m - 1, rhs);

free(matrix\_a\_array); free(matrix\_b\_array); free(rhs);

y = (double \*)malloc(sizeof(double)\*(m + 1)); //y为数值解

y[0] = alpha;

for (i = 1; i<m; i++)

y[i] = ans[i - 1];

free(ans);

y[m] = beta;

j = m / 4;

for (i = 0; i <= m; i = i + j)

printf("x=%.2f, ynumerical=%f,exact=%f, err=%.4e\n", x[i], y[i],exact(x[i]), fabs(exact(x[i])-y[i]));

}

double f(double x)

{

return -(x\*x - x + 5.0 / 4.0)\*sin(x);

}

double q(double x)

{

double z;

z = x - 0.5;

return -z\*z;

}

//本程序是解决三对角方程组的追赶法程序，其中系数矩阵主对角元为数组b, 下次对角元为数组a, 上次对角元为数组c

// b1 c1 0 0 0---0

// a2 b2 c2 0 0---0

// 0 a3 b3 c3 0---0

// 0 0 a4 b4 c4--0

// 0 0 0 a5 b5 0

// ---------------------

// 0 0 0 0 an-1 bn-1 cn-1

// 0 0 0 0 0 an bn

//右端项为数组d, 原问题是n阶方程组.

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

double exact(double x)

{

return sin(x);

}

Egch2\_sec6\_04.c

// Egch2\_sec6\_04.cpp : 定义控制台应用程序的入口点。

//本程序是求解二阶线性常微分方程两点混合边值问题的：

//y’’+q(x)y=f(x), a<x<b, y’(a)+lambda\*y(a)=alpha, y’(b)+mu\*y(b)=beta

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

double \*matrix\_a\_array, \*matrix\_b\_array, \*x, \*rhs, \*y;

double a, b, h, alpha, beta, lambda, mu;

int i, j, m;

double f(double x); //原方程右端项函数f(x)

double q(double x); //原方程中的函数q(x)

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

//追赶法子程序，a,b,c分别为系数矩阵的下次、主、上次对角线元素组，n为方程组阶数，d为矩阵右端项向量

double exact(double x);

a = 0.0; //边界左端点

b = 1.0; //边界右端点

m = 4;

h = (b - a) / m;

alpha = 0.0;

beta = 3 \* exp(1.0);

lambda = -1.0;

mu = 2.0;

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

{

x[i] = a + i\*h;

}

rhs = (double \*)malloc(sizeof(double)\*(m + 1));

rhs[0] = h\*alpha;

for (i = 1; i<m; i++) //矩阵的右端项，含m+1个元素的数组rhs

{

rhs[i] = h\*h\*f(x[i]);

}

rhs[m] = -h\*beta;

matrix\_a\_array = (double \*)malloc(sizeof(double)\*(m + 1)); //矩阵的下次对角线

matrix\_b\_array = (double \*)malloc(sizeof(double)\*(m + 1)); //矩阵的主对角线

for (i = 0; i <= m; i++)

{

matrix\_a\_array[i] = 1.0;

matrix\_b\_array[i] = h\*h\*q(x[i]) - 2;

}

matrix\_b\_array[0] = h\*lambda - 1.0;

matrix\_b\_array[m] =-( h\*mu + 1.0);

y = chase\_algorithm(matrix\_a\_array, matrix\_b\_array, matrix\_a\_array, m + 1, rhs);

free(matrix\_a\_array); free(matrix\_b\_array); free(rhs);

j = m / 4;

for (i = 0; i <= m; i = i + j)

printf("x=%.2f, ynumerical=%f,exact=%f, err=%f\n", x[i], y[i], exact(x[i]), fabs(exact(x[i]) - y[i]));

free(x); free(y);

}

double f(double x)

{

return -exp(x)\*sin(x);

}

double q(double x)

{

return -(1 + sin(x));

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g); free(w);

return ans;

}

double exact(double x)

{

return exp(x);

}

Egch2\_sec6\_05.c

// Egch2\_sec6\_05.cpp : 定义控制台应用程序的入口点。

//本程序是求解二阶线性常微分方程两点混合边值问题的

//根据新思路改进原来的一阶格式成为二阶格式

//y’’+q(x)y=f(x), a<x<b, y’(a)+lambda\*y(a)=alpha, y’(b)+mu\*y(b)=beta

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

double \*matrix\_a\_array, \*matrix\_b\_array, \*matrix\_c\_array, \*x, \*rhs, \*y;

double a, b, h, alpha, beta, lambda, mu;

int i, j, m;

double f(double x); //原方程右端项函数f(x)

double q(double x); //原方程中的函数q(x)

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

//追赶法子程序，a,b,c分别为系数矩阵的下次、主、上次对角线元素组，n为方程组阶数，d为矩阵右端项向量

double exact(double x);

a = 0.0; //边界左端点

b = 1.0; //边界右端点

m = 4;

h = (b - a) / m;

alpha = 0.0;

beta = 3 \* exp(1.0);

lambda = -1.0;

mu = 2.0;

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

{

x[i] = a + i\*h;

}

rhs = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i<=m; i++) //矩阵的右端项，含m+1个元素的数组rhs

{

rhs[i] = h\*h\*f(x[i]);

}

rhs[0]=rhs[0]+2\*h\*alpha;

rhs[m] =rhs[m] -2\*h\*beta;

matrix\_a\_array = (double \*)malloc(sizeof(double)\*(m + 1)); //矩阵的下次对角线

matrix\_b\_array = (double \*)malloc(sizeof(double)\*(m + 1)); //矩阵的主对角线

matrix\_c\_array = (double \*)malloc(sizeof(double)\*(m + 1)); //矩阵的上次对角线

for (i = 0; i <= m; i++)

{

matrix\_a\_array[i] = 1.0;

matrix\_b\_array[i] = h\*h\*q(x[i]) - 2;

matrix\_c\_array[i] =1.0;

}

matrix\_a\_array[m]=2.0;

matrix\_c\_array[0]=2.0;

matrix\_b\_array[0] = matrix\_b\_array[0] +2\* h\*lambda;

matrix\_b\_array[m] = matrix\_b\_array[m]-2\*h\*mu;

y = chase\_algorithm(matrix\_a\_array, matrix\_b\_array, matrix\_c\_array, m + 1, rhs);

free(matrix\_a\_array); free(matrix\_b\_array); free(matrix\_c\_array); free(rhs);

j = m / 4;

for (i = 0; i <= m; i = i + j)

printf("x=%.2f, ynumerical=%f,exact=%f, err=%.4e\n", x[i], y[i], exact(x[i]), fabs(exact(x[i]) - y[i]));

free(x); free(y);

}

double f(double x)

{

return -exp(x)\*sin(x);

}

double q(double x)

{

return -(1 + sin(x));

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g); free(w);

return ans;

}

double exact(double x)

{

return exp(x);

}

Egch2\_sec7\_01.c

// Egch2\_sec7\_01.cpp : 定义控制台应用程序的入口点。

//Richardson外推法

#include "stdafx.h"

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

void main()

{

double u1[3], u2[3], u3[3], v1[3], v2[3],Pi,c;

int i;

Pi = 3.14159265359;

u1[0] = 0.3830961607;

u1[1] = 0.7077460135;

u1[2] = 0.9244085928;

u2[0] = 0.3827864580;

u2[1] = 0.7072663720;

u2[2] = 0.9240117352;

u3[0] = 0.3827091794;

u3[1] = 0.7071466659;

u3[2] = 0.9239125797;

for (i = 0; i < 3; i++)

{

v1[i] = (4 \* u2[i] - u1[i]) / 3.0;

v2[i] = (4 \* u3[i] - u2[i]) / 3.0;

c = sin((i + 1)\*Pi / 8.0);

printf("i=%d, v1=%.10f, err1=%.4e, v2=%.10f, err2=%.4e\n", i, v1[i], fabs(c - v1[i]), v2[i], fabs(c - v2[i]));

}

}

Egch2\_sec7\_02.c

// Egch2\_sec7\_02.cpp : 定义控制台应用程序的入口点。

//本程序是求解二阶线性常微分方程两点Dirichlet边值问题的紧差分方法：

//y’’+q(x)y=f(x), a<x<b, y(a)=alpha, y(b)=beta

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

int m;

void main()

{

double \*matrix\_a\_array, \*matrix\_b\_array, \*matrix\_c\_array, \*x, \*z, \*rhs, \*ans, \*y;

double a, b, h, Pi, alpha, beta, c;

int i, j;

double f(double x); //原方程右端项函数f(x)

double \*farray(double \*x); //f(x)在各节点的函数值组成的数组

double q(double x); //原方程中的函数q(x)

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

//追赶法子程序，a,b,c分别为系数矩阵的下次、主、上次对角线元素组，n为方程组阶数，d为矩阵右端项向量

double exact(double x);

m = 16;

Pi = 3.14159265359;

a = 0.0; //边界左端点

b = Pi / 2.0; //边界右端点

h = (b - a) / m;

alpha = 0.0;

beta = 1.0;

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

{

x[i] = a + i\*h;

}

z = farray(x);

c = h\*h / 12.0;

rhs = (double \*)malloc(sizeof(double)\*(m - 1));

for (i = 1; i<m; i++) //矩阵的右端项，含m-1个元素的数组rhs

{

rhs[i - 1] = (z[i - 1] + 10 \* z[i] + z[i + 1])\*c;

}

rhs[0] = rhs[0]-alpha\*(1 + c\*q(x[0])); //考虑边界条件

rhs[m - 2] = rhs[m - 2]-beta\*(1 + c\*q(x[m]));

free(z);

matrix\_a\_array = (double \*)malloc(sizeof(double)\*(m - 1)); //矩阵的下次对角线

matrix\_b\_array = (double \*)malloc(sizeof(double)\*(m - 1)); //矩阵的主对角线

matrix\_c\_array = (double \*)malloc(sizeof(double)\*(m - 1)); //矩阵的上次对角线

for (i = 0; i<m - 1; i++)

{

matrix\_a\_array[i] = 1.0 + c\*q(x[i]);

matrix\_b\_array[i] = 10 \* c\*q(x[i + 1]) - 2;

matrix\_c\_array[i] = 1.0 + c\*q(x[i + 2]);

}

ans = chase\_algorithm(matrix\_a\_array, matrix\_b\_array, matrix\_c\_array, m - 1, rhs);

free(matrix\_a\_array); free(matrix\_b\_array); free(rhs);

y = (double \*)malloc(sizeof(double)\*(m + 1)); //y为数值解

y[0] = alpha;

for (i = 1; i<m; i++)

y[i] = ans[i - 1];

free(ans);

y[m] = beta;

j = m / 4;

for (i = 0; i <= m; i = i + j)

printf("x=%.2f, ynumerical=%f,exact=%f, err=%.4e\n", x[i], y[i], exact(x[i]), fabs(exact(x[i]) - y[i]));

}

double f(double x)

{

return -(x\*x - x + 5.0 / 4.0)\*sin(x);

}

double \* farray(double \*x)

{

int i;

double \*z;

z = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

z[i] = f(x[i]);

return z;

}

double q(double x)

{

double z;

z = x - 0.5;

return -z\*z;

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

double exact(double x)

{

return sin(x);

}

## 第三章

Egch3\_sec1\_01.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

int m, n, i, j, k,number;

double a, h, tau, r, \*x, \*t, \*\*u;

double phi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

m = 10;

n = 100;

a = 1.0;

h = 1.0 / m;

tau = 1.0 / n;

r = a\*tau / (h\*h);

printf("r=%.4f\n", r);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

u[i][0] = phi(x[i]); //initial condition

for (k = 1; k <= n; k++)

{

u[0][k] = alpha(t[k]);

u[m][k] = beta(t[k]);

}

for (k = 0; k<n; k++)

{

for (i = 1; i<m; i++)

u[i][k + 1] = r\*u[i - 1][k] + (1 - 2 \* r)\*u[i][k] + r\*u[i + 1][k] + tau\*f(x[i], t[k]);

}

j =int( 0.2 / tau);

number = int(0.4 / h); //x=0.4时对应数组x中节点的编号

for (k = j; k<=n; k=k+j)

{

printf("(x,t)=(%.1f,%.1f), numerical=%f, exact=%f, err=%.4e\n", x[number], t[k], u[number][k], exact(x[number], t[k]), fabs(u[number][k] - exact(x[number], t[k])));

}

free(x); free(t);

for (i = 0; i <= m; i++)

free(u[i]);

free(u);

}

double phi(double x)

{

return x\*(x\*x + 1);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 1.0 + exp(t);

}

double f(double x, double t)

{

return x\*exp(t)-6\*x;

}

double exact(double x, double t)

{

return x\*(x\*x+exp(t));

}

Egch3\_sec1\_02.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

int m, n, i, j, k, number;

double a, h, tau, r, \*x, \*t, \*\*u;

double phi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

m = 10;

n = 100;

a = 1.0;

h = 1.0 / m;

tau = 1.0 / n;

r = a\*tau / (h\*h);

printf("r=%.4f\n", r);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

u[i][0] = phi(x[i]); //initial condition

for (k = 1; k <= n; k++)

{

u[0][k] = alpha(t[k]);

u[m][k] = beta(t[k]);

}

for (k = 0; k<n; k++)

{

for (i = 1; i<m; i++)

u[i][k + 1] = r\*u[i - 1][k] + (1 - 2 \* r)\*u[i][k] + r\*u[i + 1][k] + tau\*f(x[i], t[k]);

}

j = int(0.2 / tau);

number = int(0.4 / h); //x=0.4时对应数组x中节点的编号

for (k = j; k <= n; k = k + j)

{

printf("(x,t)=(%.1f,%.1f), numerical=%f, exact=%f, err=%.4e\n", x[number], t[k], u[number][k], exact(x[number], t[k]), fabs(u[number][k] - exact(x[number], t[k])));

}

free(x); free(t);

for (i = 0; i <= m; i++)

free(u[i]);

free(u);

}

double phi(double x)

{

return pow(x,4);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return exp(t);

}

double f(double x, double t)

{

double z;

z = x\*x;

return z\*(z - 12.0)\*exp(t);

}

double exact(double x, double t)

{

return pow(x,4)\*exp(t);

}

Egch3\_sec2\_01.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

int m, n, i, j, k, number;

double a, h, tau, r, \*x, \*t, \*\*u, \*a1, \*b, \*c, \*d, \*ans;

double phi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m = 5;

n = 100;

a = 1.0;

h = 1.0 / m;

tau = 1.0 / n;

r = a\*tau / (h\*h);

printf("r=%.4f\n", r);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

u[i][0] = phi(x[i]); //初始条件

for (k = 1; k <= n; k++)

{

u[0][k] = alpha(t[k]);//边界条件

u[m][k] = beta(t[k]);

}

a1 = (double \*)malloc(sizeof(double)\*(m - 1));//追赶法中下次对角线上的元素组

b = (double \*)malloc(sizeof(double)\*(m - 1));//追赶法中主对角线上的元素组

c = (double \*)malloc(sizeof(double)\*(m - 1));//追赶法中上次对角线上的元素组

d = (double \*)malloc(sizeof(double)\*(m - 1));//追赶法中右端项数组

ans = (double \*)malloc(sizeof(double)\*(m - 1));

for (k = 1; k <= n; k++)

{

for (i = 0; i<m - 1; i++)

{

d[i] = u[i + 1][k - 1] + tau\*f(x[i + 1], t[k]); //由于C语言从0 计数，所以d[0]存储的实为x[1]处的函数值

a1[i] = -r;

b[i] = 1.0 + 2 \* r;

c[i] = a1[i];

}

d[0] = d[0] + r\*u[0][k];

d[m - 2] = d[m - 2] + r\*u[m][k];

ans = chase\_algorithm(a1, b, c, m - 1, d); //追赶法求解

for (i = 0; i<m - 1; i++)

u[i + 1][k] = ans[i];

}

j = int(0.2 / tau);

number = int(0.4 / h); //x=0.4时对应数组x中节点的编号

for (k = j; k <= n; k = k + j)

{

printf("(x,t)=(%.1f,%.1f), numerical=%f, exact=%f, err=%.4e\n", x[number], t[k], u[number][k], exact(x[number], t[k]), fabs(u[number][k] - exact(x[number], t[k])));

}

free(a1); free(b); free(c); free(d); free(ans); free(x); free(t);

for (i = 0; i <= m; i++)

free(u[i]);

free(u);

}

double phi(double x)

{

return x\*(x\*x+1);

}

double alpha(double t)

{

return 0;

}

double beta(double t)

{

return 1+exp(t);

}

double f(double x, double t)

{

return x\*exp(t)-6\*x;

}

double exact(double x, double t)

{

return x\*(x\*x+exp(t));

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

Egch3\_sec2\_02.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

int m, n, i, j, k, number;

double a, h, tau, r, \*x, \*t, \*\*u, \*a1, \*b, \*c, \*d, \*ans;

double phi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m = 5;

n = 100;

a = 1.0;

h = 1.0 / m;

tau = 1.0 / n;

r = a\*tau / (h\*h);

printf("r=%.4f\n", r);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

u[i][0] = phi(x[i]); //初始条件

for (k = 1; k <= n; k++)

{

u[0][k] = alpha(t[k]);//边界条件

u[m][k] = beta(t[k]);

}

a1 = (double \*)malloc(sizeof(double)\*(m - 1));//追赶法中下次对角线上的元素组

b = (double \*)malloc(sizeof(double)\*(m - 1));//追赶法中主对角线上的元素组

c = (double \*)malloc(sizeof(double)\*(m - 1));//追赶法中上次对角线上的元素组

d = (double \*)malloc(sizeof(double)\*(m - 1));//追赶法中右端项数组

ans = (double \*)malloc(sizeof(double)\*(m - 1));

for (k = 1; k <= n; k++)

{

for (i = 0; i<m - 1; i++)

{

d[i] = u[i + 1][k - 1] + tau\*f(x[i + 1], t[k]); //由于C语言从0 计数，所以d[0]存储的实为x[1]处的函数值

a1[i] = -r;

b[i] = 1.0 + 2 \* r;

c[i] = a1[i];

}

d[0] = d[0] + r\*u[0][k];

d[m - 2] = d[m - 2] + r\*u[m][k];

ans = chase\_algorithm(a1, b, c, m - 1, d); //追赶法求解

for (i = 0; i<m - 1; i++)

u[i + 1][k] = ans[i];

}

j = int(0.2 / tau);

number = int(0.4 / h); //x=0.4时对应数组x中节点的编号

for (k = j; k <= n; k = k + j)

{

printf("(x,t)=(%.1f,%.1f), numerical=%f, exact=%f, err=%.4e\n", x[number], t[k], u[number][k], exact(x[number], t[k]), fabs(u[number][k] - exact(x[number], t[k])));

}

free(a1); free(b); free(c); free(d); free(ans); free(x); free(t);

for (i = 0; i <= m; i++)

free(u[i]);

free(u);

}

double phi(double x)

{

double z;

z = x\*x;

return z\*z;

}

double alpha(double t)

{

return 0;

}

double beta(double t)

{

return exp(t);

}

double f(double x, double t)

{

double z;

z = x\*x;

return z\*(z-12)\*exp(t);

}

double exact(double x, double t)

{

return pow(x,4)\*exp(t);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

Egch3\_sec3\_01.c

//这是个Richardson格式，处处不稳定

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

int m, n, i, j, k,number;

double a, h, tau, r, \*x, \*t, \*\*u;

double phi(double x);

double ddphi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

m = 5;

n = 100;

a = 1.0;

h = 1.0 / m;

tau = 1.0 / n;

r = a\*tau / (h\*h);

printf("r=%.4f\n", r);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

u[i][0] = phi(x[i]); //初始条件

for (i = 1; i < m; i++)

u[i][1] = phi(x[i]) + tau\*(a\*ddphi(x[i]) + f(x[i], t[0])); // 第1层内节点信息

for (k = 1; k <= n; k++) //边界条件

{

u[0][k] = alpha(t[k]);

u[m][k] = beta(t[k]);

}

for (k = 1; k<n; k++)

{

for (i = 1; i<m; i++)

u[i][k + 1] = 2\*r\*(u[i - 1][k] -2\*u[i][k] + u[i + 1][k]) +2\*tau\*f(x[i], t[k])+u[i][k-1];

}

j =int( 0.02/ tau);

number = int(0.4 / h); //x=0.4时对应数组x中节点的编号

for (k = j; k<=n; k=k+j)

{

printf("(x,t)=(%.2f,%.2f), numerical=%f, exact=%f, err=%.4e\n", x[number], t[k], u[number][k], exact(x[number], t[k]), fabs(u[number][k] - exact(x[number], t[k])));

}

free(x); free(t);

for (i = 0; i <= m; i++)

free(u[i]);

free(u);

}

double phi(double x)

{

return x\*(x\*x + 1);

}

double ddphi(double x)

{

return 6 \* x;

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 1.0 + exp(t);

}

double f(double x, double t)

{

return x\*exp(t) - 6 \* x;

}

double exact(double x, double t)

{

return x\*(x\*x + exp(t));

}

Egch3\_sec3\_02.c

//Crank-Nicolson格式（二阶）

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

int m, n, i, j, k, number;

double a, h, tau, r, \*x, \*t, \*\*u, \*a1, \*b, \*c, \*d, \*ans, tkmid;

double phi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m = 5;

n = 10;

a = 1.0;

h = 1.0 / m;

tau = 1.0 / n;

r = a\*tau / (h\*h);

printf("r=%.4f\n", r);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

u[i][0] = phi(x[i]); //初始条件

for (k = 1; k <= n; k++)

{

u[0][k] = alpha(t[k]); //边界条件

u[m][k] = beta(t[k]);

}

a1 = (double \*)malloc(sizeof(double)\*(m - 1));

b = (double \*)malloc(sizeof(double)\*(m - 1));

c = (double \*)malloc(sizeof(double)\*(m - 1));

d = (double \*)malloc(sizeof(double)\*(m - 1));

ans = (double \*)malloc(sizeof(double)\*(m - 1));

for (k = 0; k<n; k++)

{

tkmid = (t[k] + t[k + 1]) / 2.0;

for (i = 0; i<m - 1; i++)

{

d[i] = r\*u[i][k] / 2.0 + (1.0 - r)\*u[i + 1][k] + r\*u[i + 2][k] / 2.0 + tau\*f(x[i + 1], tkmid);

a1[i] = -r / 2.0;

b[i] = 1.0 + r;

c[i] = a1[i];

}

d[0] = d[0]+r\*u[0][k+1]/2.0;

d[m - 2] = d[m - 2] + r\*u[m][k + 1] / 2.0;

ans = chase\_algorithm(a1, b, c, m - 1, d);

for (i = 0; i<m - 1; i++)

u[i + 1][k + 1] = ans[i];

}

j = int(0.2 / tau);

number = int(0.4 / h); //x=0.4时对应数组x中节点的编号

for (k = j; k <= n; k = k + j)

{

printf("(x,t)=(%.1f,%.1f), numerical=%f, exact=%f, err=%.4e\n", x[number], t[k], u[number][k], exact(x[number], t[k]), fabs(u[number][k] - exact(x[number], t[k])));

}

free(a1);free(b);free(c);free(d);

free(x); free(t); free(ans);

for (i = 0; i <= m; i++)

free(u[i]);

free(u);

}

double phi(double x)

{

return x\*(x\*x+1);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 1.0+exp(t);

}

double f(double x, double t)

{

return x\*exp(t)-6\*x;

}

double exact(double x, double t)

{

return x\*(x\*x+exp(t));

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

Egch3\_sec3\_03.c

//Crank-Nicolson格式（二阶）

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

int m, n, i, j, k, number;

double a, h, tau, r, \*x, \*t, \*\*u, \*a1, \*b, \*c, \*d, \*ans, tkmid;

double phi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m = 5;

n = 10;

a = 1.0;

h = 1.0 / m;

tau = 1.0 / n;

r = a\*tau / (h\*h);

printf("r=%.4f\n", r);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

u[i][0] = phi(x[i]); //初始条件

for (k = 1; k <= n; k++)

{

u[0][k] = alpha(t[k]); //边界条件

u[m][k] = beta(t[k]);

}

a1 = (double \*)malloc(sizeof(double)\*(m - 1));

b = (double \*)malloc(sizeof(double)\*(m - 1));

c = (double \*)malloc(sizeof(double)\*(m - 1));

d = (double \*)malloc(sizeof(double)\*(m - 1));

ans = (double \*)malloc(sizeof(double)\*(m - 1));

for (k = 0; k<n; k++)

{

tkmid = (t[k] + t[k + 1]) / 2.0;

for (i = 0; i<m - 1; i++)

{

d[i] = r\*u[i][k] / 2.0 + (1.0 - r)\*u[i + 1][k] + r\*u[i + 2][k] / 2.0 + tau\*f(x[i + 1], tkmid);

a1[i] = -r / 2.0;

b[i] = 1.0 + r;

c[i] = a1[i];

}

d[0] = d[0] + r\*u[0][k + 1] / 2.0;

d[m - 2] = d[m - 2] + r\*u[m][k + 1] / 2.0;

ans = chase\_algorithm(a1, b, c, m - 1, d);

for (i = 0; i<m - 1; i++)

u[i + 1][k + 1] = ans[i];

}

j = int(0.2 / tau);

number = int(0.4 / h); //x=0.4时对应数组x中节点的编号

for (k = j; k <= n; k = k + j)

{

printf("(x,t)=(%.1f,%.1f), numerical=%f, exact=%f, err=%.4e\n", x[number], t[k], u[number][k], exact(x[number], t[k]), fabs(u[number][k] - exact(x[number], t[k])));

}

free(a1); free(b); free(c); free(d);

free(x); free(t); free(ans);

for (i = 0; i <= m; i++)

free(u[i]);

free(u);

}

double phi(double x)

{

return pow(x,4);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return exp(t);

}

double f(double x, double t)

{

double z;

z = x\*x;

return z\*(z-12)\*exp(t);

}

double exact(double x, double t)

{

return pow(x,4)\*exp(t);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

Egch3\_sec4\_01.c

//Richardson外推法

#include "stdafx.h"

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

void main()

{

double u1[5], u2[5], u3[5], v1[5], v2[5],c;

int i;

u1[0] = 0.5525382324;

u1[1] = 0.6606989808;

u1[2] = 0.7928093979;

u1[3] = 0.9541697560;

u1[4] = 1.1512557860;

u2[0] = 0.5525554756;

u2[1] =0.6607221484;

u2[2] = 0.7928379622;

u2[3] = 0.9542046817;

u2[4] = 1.1512984515;

u3[0] =0.5525596981;

u3[1] = 0.6607279459;

u3[2] =0.7928451291;

u3[3] = 0.9542134468;

u3[4] =1.1513091587;

for (i = 0; i < 5; i++)

{

v1[i] = (4 \* u2[i] - u1[i]) / 3.0;

v2[i] = (4 \* u3[i] - u2[i]) / 3.0;

c =0.4\*(0.16+exp(0.2\*(i+1)));

printf("i=%d, v1=%.10f, err1=%.4e, v2=%.10f, err2=%.4e\n", i, v1[i], fabs(c - v1[i]), v2[i], fabs(c - v2[i]));

}

}

Egch3\_sec4\_02.c

//Richardson外推法

#include "stdafx.h"

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

void main()

{

double u1[5], u2[5], u3[5], v1[5], v2[5],c;

int i;

u1[0] =0.0412476914;

u1[1] = 0.0516017382;

u1[2] = 0.0631711853;

u1[3] =0.0771733069;

u1[4] = 0.0942610774;

u2[0] =0.0337297132;

u2[1] =0.0415430399;

u2[2] =0.0507878226;

u2[3] =0.0620387880;

u2[4] = 0.0757752186;

u3[0] =0.0318823557;

u3[1] = 0.0390290850;

u3[2] =0.0476823660;

u3[3] = 0.0582410491;

u3[4] =0.0711360095;

for (i = 0; i < 5; i++)

{

v1[i] = (4 \* u2[i] - u1[i]) / 3.0;

v2[i] = (4 \* u3[i] - u2[i]) / 3.0;

c =pow(0.4,4)\*exp(0.2\*(i+1));

printf("i=%d, v1=%.10f, err1=%.4e, v2=%.10f, err2=%.4e\n", i, v1[i], fabs(c - v1[i]), v2[i], fabs(c - v2[i]));

}

}

Egch3\_sec4\_03.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

int m,n,i,j,k,xnumber;

double a,h,tau,r,\*x,\*t,\* \*u, \*a1,\*b,\*c,\*d,\*ans,tkmid;

double phi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m=5;

n=10;

a=1.0;

h=1.0/m;

tau=1.0/n;

r=a\*tau/(h\*h);

printf("r=%.4f\n",r);

x=(double \*)malloc(sizeof(double)\*(m+1));

for(i=0;i<=m;i++)

x[i]=i\*h;

t=(double \*)malloc(sizeof(double)\*(n+1));

for(k=0;k<=n;k++)

t[k]=k\*tau;

u=(double \* \*)malloc(sizeof(double \*)\*(m+1));

for(i=0;i<=m;i++)

u[i]=(double \*)malloc(sizeof(double)\*(n+1));

for(i=0;i<=m;i++)

u[i][0]=phi(x[i]); //initial condition

for(k=1;k<=n;k++)

{

u[0][k]=alpha(t[k]); //boundary condition

u[m][k]=beta(t[k]);

}

a1=(double \*)malloc(sizeof(double)\*(m-1));

b=(double \*)malloc(sizeof(double)\*(m-1));

c=(double \*)malloc(sizeof(double)\*(m-1));

d=(double \*)malloc(sizeof(double)\*(m-1));

ans=(double \*)malloc(sizeof(double)\*(m-1));

for(k=0;k<n;k++)

{

tkmid=(t[k]+t[k+1])/2.0;

for(i=0;i<m-1;i++)

{

d[i]=(1.0/12+0.5\*r)\*(u[i][k]+u[i+2][k])+(5.0/6-r)\*u[i+1][k]+tau\*(f(x[i],tkmid)+10\*f(x[i+1],tkmid)+f(x[i+2],tkmid))/12.0;

a1[i]=1.0/12-r/2.0;

b[i]=5.0/6+r;

c[i]=a1[i];

}

d[0]=d[0]-(1.0/12-0.5\*r)\*u[0][k+1];

d[m-2]=d[m-2]-(1.0/12-0.5\*r)\*u[m][k+1];

ans=chase\_algorithm(a1,b,c,m-1,d);

for(i=0;i<m-1;i++)

u[i+1][k+1]=ans[i];

}

xnumber=int(0.4/h);

j=int(0.2/tau);

for(k=j;k<=n;k=k+j)

{

printf("(x,t)=(0.4,%.1f), numerical=%f, error=%.4e\n",t[k],u[xnumber][k], fabs(u[xnumber][k]-exact(0.4,t[k])));

}

free(a1); free(b);free(c);free(d);free(x);free(t);free(ans);

for(i=0;i<=m;i++)

free(u[i]);

free(u);

}

double phi(double x)

{

return x\*(x\*x+1);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 1.0+exp(t);

}

double f(double x,double t)

{

return x\*exp(t)-6\*x;

}

double exact(double x, double t)

{

return x\*(x\*x+exp(t));

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans,\*g,\*w,p;

int i;

ans=(double \*)malloc(sizeof(double)\*n);

g=(double \*)malloc(sizeof(double)\*n);

w=(double \*)malloc(sizeof(double)\*n);

g[0]=d[0]/b[0];

w[0]=c[0]/b[0];

for(i=1;i<n;i++)

{

p=b[i]-a[i]\*w[i-1];

g[i]=(d[i]-a[i]\*g[i-1])/p;

w[i]=c[i]/p;

}

ans[n-1]=g[n-1];

i=n-2;

do

{

ans[i]=g[i]-w[i]\*ans[i+1];

i=i-1;

}

while(i>=0);

free(g);

free(w);

return ans;

}

Egch3\_sec4\_04.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

int m,n,i,j,k,xnumber;

double a,h,tau,r,\*x,\*t,\* \*u, \*a1,\*b,\*c,\*d,\*ans,tkmid;

double phi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m=5;

n=10;

a=1.0;

h=1.0/m;

tau=1.0/n;

r=a\*tau/(h\*h);

printf("r=%.4f\n",r);

x=(double \*)malloc(sizeof(double)\*(m+1));

for(i=0;i<=m;i++)

x[i]=i\*h;

t=(double \*)malloc(sizeof(double)\*(n+1));

for(k=0;k<=n;k++)

t[k]=k\*tau;

u=(double \* \*)malloc(sizeof(double \*)\*(m+1));

for(i=0;i<=m;i++)

u[i]=(double \*)malloc(sizeof(double)\*(n+1));

for(i=0;i<=m;i++)

u[i][0]=phi(x[i]); //initial condition

for(k=1;k<=n;k++)

{

u[0][k]=alpha(t[k]); //boundary condition

u[m][k]=beta(t[k]);

}

a1=(double \*)malloc(sizeof(double)\*(m-1));

b=(double \*)malloc(sizeof(double)\*(m-1));

c=(double \*)malloc(sizeof(double)\*(m-1));

d=(double \*)malloc(sizeof(double)\*(m-1));

ans=(double \*)malloc(sizeof(double)\*(m-1));

for(k=0;k<n;k++)

{

tkmid=(t[k]+t[k+1])/2.0;

for(i=0;i<m-1;i++)

{

d[i]=(1.0/12+0.5\*r)\*(u[i][k]+u[i+2][k])+(5.0/6-r)\*u[i+1][k]+tau\*(f(x[i],tkmid)+10\*f(x[i+1],tkmid)+f(x[i+2],tkmid))/12.0;

a1[i]=1.0/12-r/2.0;

b[i]=5.0/6+r;

c[i]=a1[i];

}

d[0]=d[0]-(1.0/12-0.5\*r)\*u[0][k+1];

d[m-2]=d[m-2]-(1.0/12-0.5\*r)\*u[m][k+1];

ans=chase\_algorithm(a1,b,c,m-1,d);

for(i=0;i<m-1;i++)

u[i+1][k+1]=ans[i];

}

xnumber=int(0.4/h);

j=int(0.2/tau);

for(k=j;k<=n;k=k+j)

{

printf("(x,t)=(0.4,%.1f), numerical=%f, error=%.4e\n",t[k],u[xnumber][k], fabs(u[xnumber][k]-exact(0.4,t[k])));

}

free(a1); free(b);free(c);free(d);free(x);free(t);free(ans);

for(i=0;i<=m;i++)

free(u[i]);

free(u);

}

double phi(double x)

{

return pow(x,4.0);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return exp(t);

}

double f(double x,double t)

{

double z;

z=x\*x;

return exp(t)\*z\*(z-12.0);

}

double exact(double x, double t)

{

return pow(x,4.0)\*exp(t);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans,\*g,\*w,p;

int i;

ans=(double \*)malloc(sizeof(double)\*n);

g=(double \*)malloc(sizeof(double)\*n);

w=(double \*)malloc(sizeof(double)\*n);

g[0]=d[0]/b[0];

w[0]=c[0]/b[0];

for(i=1;i<n;i++)

{

p=b[i]-a[i]\*w[i-1];

g[i]=(d[i]-a[i]\*g[i-1])/p;

w[i]=c[i]/p;

}

ans[n-1]=g[n-1];

i=n-2;

do

{

ans[i]=g[i]-w[i]\*ans[i+1];

i=i-1;

}

while(i>=0);

free(g);

free(w);

return ans;

}

Egch3\_sec5\_01.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

int m, n, i, k;

double h, tau, a,lambda,mu,r, \*x, \*t,\*\*u;

double f(double x, double t);

double phi(double x);

double alpha(double t);

double beta(double t);

m = 10;

n = 400;

h = 1.0 / m;

tau = 1.0 / n;

a = 1.0;

lambda = 1.0;

mu = 1.0;

r =a\* tau / (h\*h);

printf("r=%.4f\n", r);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

u[i][0] = phi(x[i]); //初始条件

for (k = 0; k<n; k++)

{

u[0][k + 1] = (1.0-2\*r-2\*r\*lambda\*h)\*u[0][k] + 2 \* r\*u[1][k]-2\*r\*h\*alpha(t[k]) + tau\*f(x[0], t[k]);

for (i = 1; i<m; i++)

u[i][k + 1] = r\*u[i - 1][k] + (1 - 2 \* r)\*u[i][k] + r\*u[i + 1][k] + tau\*f(x[i], t[k]); //一阶向前Euler,边界条件用中心差分

u[m][k + 1] = 2 \* r\*u[m - 1][k] + (1.0 - 2 \* r - 2 \* r\*mu\*h)\*u[m][k] +2\*r\*h\*beta(t[k])+ tau\*f(x[m], t[k]);

}

printf("t/x 0 0.1 0.2 0.3 0.4 0.5\n");

for (k = 1; k <= 8; k++)

{

printf("%.4f ", t[k]);

for (i = 0; i <= m / 2; i++)

printf("%.4f ", u[i][k]);

printf("\n");

}

printf("\n");

printf("……\n");

printf("\n");

printf("0.1000 ");

for (i = 0; i <= m / 2; i++)

printf("%.4f ", u[i][40]);

printf("\n");

for (k = 1; k <= 4; k = 2 \* k)

{

printf("%.4f ", t[k\*100]);

for (i = 0; i <= m / 2; i++)

printf("%.4f ", u[i][k \* 100]);

printf("\n");

}

}

double f(double x, double t)

{

return 0;

}

double phi(double x)

{

return 1.0;

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 0.0;

}

Egch3\_sec5\_02.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

int m, n, i, k;

double h, tau, a, lambda, mu,r, \*x, \*t, \*\*u;

double f(double x, double t);

double phi(double x);

double alpha(double t);

double beta(double t);

m = 10;

n = 400;

h = 1.0 / m;

tau = 1.0 / n;

a = 1.0;

lambda = 1.0;

mu = 1.0;

r = a\*tau / (h\*h);

printf("r=%.4f\n", r);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

u[i][0] = phi(x[i]); //初始条件

for (k = 0; k<n; k++)

{

for (i = 1; i<m; i++)

u[i][k + 1] = r\*u[i - 1][k] + (1 - 2 \* r)\*u[i][k] + r\*u[i + 1][k] + tau\*f(x[i], t[k]);//一阶向前欧拉，左右边界分别用向前、向后差分

u[0][k + 1] = (u[1][k + 1]-h\*alpha(t[k]))/(1.0+lambda\*h);

u[m][k + 1] = (u[m - 1][k + 1]+h\*beta(t[k]))/(1.0+mu\*h);

}

printf("t/x 0 0.1 0.2 0.3 0.4 0.5\n");

for (k = 1; k <= 8; k++)

{

printf("%.4f ", t[k]);

for (i = 0; i <= m / 2; i++)

printf("%.4f ", u[i][k]);

printf("\n");

}

printf("\n");

printf("……\n");

printf("\n");

printf("0.1000 ");

for (i = 0; i <= m / 2; i++)

printf("%.4f ", u[i][40]);

printf("\n");

for (k = 1; k <= 4; k = 2 \* k)

{

printf("%.4f ", t[k \* 100]);

for (i = 0; i <= m / 2; i++)

printf("%.4f ", u[i][k \* 100]);

printf("\n");

}

}

double f(double x, double t)

{

return 0;

}

double phi(double x)

{

return 1.0;

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 0.0;

}

Egch3\_sec5\_03.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

int m, n, i, k;

double h, tau, a, lambda,mu,r, \*x, \*t, \*a1, \*b, \*c, \*d, \*ans, \*\*u, tkmid;

double f(double x, double t);

double phi(double x);

double alpha(double t);

double beta(double t);

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m = 10;

n = 400;

h = 1.0 / m;

tau = 1.0 / n;

a = 1.0;

lambda = 1.0;

mu = 1.0;

r = a\*tau / (h\*h);

printf("r=%.4f\n", r);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

u[i][0] = phi(x[i]); //初始条件

a1 = (double \*)malloc(sizeof(double)\*(m + 1));

b = (double \*)malloc(sizeof(double)\*(m + 1));

c = (double \*)malloc(sizeof(double)\*(m + 1));

d = (double \*)malloc(sizeof(double)\*(m + 1));

ans = (double \*)malloc(sizeof(double)\*(m + 1));

for (k = 0; k<n; k++)

{

tkmid = (t[k] + t[k + 1]) / 2.0;

for (i = 1; i<m; i++)

{

d[i] = r\*u[i - 1][k] / 2.0 + (1.0 - r)\*u[i][k] + r\*u[i + 1][k] / 2.0 + tau\*f(x[i], tkmid);

a1[i] = -r / 2.0;

b[i] = 1.0 + r;

c[i] = a1[i];

}

b[0] = 1.0 + r + r\*lambda\*h;

b[m] = 1.0 + r + r \* mu\*h;

c[0] = -r;

a1[m] = -r;

d[0] = (1.0 - r - r\*lambda\*h)\*u[0][k] + r\*u[1][k]-r\*h\*alpha(t[k])-r\*h\*alpha(t[k+1]) + tau\*f(x[0], tkmid);

d[m] = r\*u[m - 1][k] + (1.0 - r - r\*mu\*h)\*u[m][k] + r\*h\*beta(t[k])+r\*h\*beta(t[k+1])+tau\*f(x[m], tkmid);

ans = chase\_algorithm(a1, b, c, m + 1, d);

for (i = 0; i <= m; i++)

u[i][k + 1] = ans[i];

}

free(a1);free(b);free(c);free(d);

printf("t/x 0 0.1 0.2 0.3 0.4 0.5\n");

for (k = 1; k <= 8; k++)

{

printf("%.4f ", t[k]);

for (i = 0; i <= m / 2; i++)

printf("%.4f ", u[i][k]);

printf("\n");

}

printf("\n");

printf("……\n");

printf("\n");

printf("0.1000 ");

for (i = 0; i <= m / 2; i++)

printf("%.4f ", u[i][40]);

printf("\n");

for (k = 1; k <= 4; k = 2 \* k)

{

printf("%.4f ", t[k \* 100]);

for (i = 0; i <= m / 2; i++)

printf("%.4f ", u[i][k \* 100]);

printf("\n");

}

}

double f(double x, double t)

{

return 0;

}

double phi(double x)

{

return 1.0;

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 0.0;

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

//printf("i=%d, Now it's OK!\n",i);

} while (i >= 0);

free(g);

free(w);

return ans;

}

Egch3\_sec6\_01.c

//求解二维抛物型方程初边值问题： ADI 之 Peaceman-Rachford 格式

#include"stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main()

{

double a, b, T, r1, r2, dx, dy, dt, \*x, \*y, \*t, \*\* \*u, \*\*v;

double \*a1, \*b1, \*c1, \*d1, \*a2, \*b2, \*c2, \*d2, tmid, \*ans, temp;

int i, j, k, m, n, L, gap\_i, gap\_j;

double f(double x, double y, double t);

double phi(double x, double y);

double g1(double y, double t);

double g2(double y, double t);

double g3(double x, double t);

double g4(double x, double t);

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

double exact(double x, double y, double t);

a = 1.0;

b = 1.0;

T = 1.0;

m = 60;

n = 40;

L = 20;

dx = a / m;

dy = b / n;

dt = T / L;

r1 = dt / (dx\*dx);

r2 = dt / (dy\*dy);

printf("m=%d, n=%d, L=%d\n", m, n, L);

printf("r1=%.4f, r2=%.4f\n", r1, r2);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*dx;

y = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= n; j++)

y[j] = j\*dy;

t = (double \*)malloc(sizeof(double)\*(L + 1));

for (k = 0; k <= L; k++)

t[k] = k\*dt;

u = (double \* \* \*)malloc(sizeof(double \*)\*((m + 1)\*(n + 1)\*(L + 1)));

for (i = 0; i <= m; i++)

u[i] = (double \* \*)malloc(sizeof(double \*)\*((n + 1)\*(L + 1)));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

u[i][j] = (double \*)malloc(sizeof(double)\*(L + 1));

}

v = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

v[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

{

u[i][j][0] = phi(x[i], y[j]); //初始条件

}

}

for (k = 1; k <= L; k++)

{

for (j = 0; j <= n; j++)

{

u[0][j][k] = g1(y[j], t[k]); //左边界条件

u[m][j][k] = g2(y[j], t[k]); //右边界条件

}

for (i = 1; i <= m - 1; i++)

{

u[i][0][k] = g3(x[i], t[k]); //下边界条件

u[i][n][k] = g4(x[i], t[k]); //上边界条件

}

}

a1 = (double \*)malloc(sizeof(double)\*(m - 1));

b1 = (double \*)malloc(sizeof(double)\*(m - 1));

c1 = (double \*)malloc(sizeof(double)\*(m - 1));

d1 = (double \*)malloc(sizeof(double)\*(m - 1));

for (i = 0; i< m - 1; i++)

{

a1[i] = -r1/2.0;

b1[i] = 1.0 + r1;

c1[i] = a1[i];

}

a2 = (double \*)malloc(sizeof(double)\*(n - 1));

b2 = (double \*)malloc(sizeof(double)\*(n - 1));

c2 = (double \*)malloc(sizeof(double)\*(n - 1));

d2 = (double \*)malloc(sizeof(double)\*(n - 1));

for (j = 0; j < n - 1; j++)

{

a2[j] = -r2/2.0;

b2[j] = 1.0 + r2;

c2[j] = a2[j];

}

for (k = 0; k < L; k++)

{

tmid = (t[k] + t[k + 1]) / 2.0;

for (j = 1; j <= n - 1; j++)//固定j

{

for (i = 1; i <= m - 1; i++)

d1[i - 1] = r2\*(u[i][j - 1][k] + u[i][j + 1][k])/2.0 + (1.0 - r2)\*u[i][j][k] + f(x[i], y[j], tmid)\*dt/2.0;

v[0][j] = (1 - r2)\*u[0][j][k] / 2.0 + (1 + r2)\*u[0][j][k + 1] / 2.0 + r2\*(u[0][j - 1][k] + u[0][j + 1][k] - u[0][j - 1][k + 1] - u[0][j + 1][k + 1]) / 4.0;

v[m][j] = (1 - r2)\*u[m][j][k] / 2.0 + (1 + r2)\*u[m][j][k + 1] / 2.0 + r2\*(u[m][j - 1][k] + u[m][j + 1][k] - u[m][j - 1][k + 1] - u[m][j + 1][k + 1]) / 4.0;

d1[0] = d1[0] + r1\*v[0][j]/2.0;

d1[m - 2] = d1[m - 2] + r1\*v[m][j]/2.0;

ans = chase\_algorithm(a1, b1, c1, m - 1, d1);

for (i = 1; i <= m - 1; i++)

v[i][j] = ans[i - 1];

free(ans);

}

for (i = 1; i <= m - 1; i++)//固定i

{

for (j = 1; j <= n - 1; j++)

d2[j - 1] = r1\*(v[i - 1][j] + v[i + 1][j])/2.0 + (1.0 - r1)\*v[i][j] + f(x[i], y[j], tmid)\*dt/2.0;

d2[0] = d2[0] + r2\*u[i][0][k + 1]/2.0;

d2[n - 2] = d2[n - 2] + r2\*u[i][n][k + 1]/2.0;

ans = chase\_algorithm(a2, b2, c2, n - 1, d2);

for (j = 1; j <= n - 1; j++)

u[i][j][k + 1] = ans[j - 1];

free(ans);

}

}//end for k

gap\_i = m / 4;//用于确定x方向每隔多少个点打印结果

gap\_j = n / 4;//用于确定y方向每隔多少个点打印结果

for (i = gap\_i; i <= m - 1; i = i + gap\_i)

{

for (j = gap\_j; j <= n - 1; j = j + gap\_j)

{

temp = fabs(exact(x[i], y[j], T / 2.0) - u[i][j][L / 2]);

printf("(%.2f, %.2f, 0.50) numerical=%f, err=%.4e\n", x[i], y[j], u[i][j][L / 2], temp);

}

}

printf("\n");

printf("\n");

for (i = gap\_i; i <= m - 1; i = i + gap\_i)

{

for (j = gap\_j; j <= n - 1; j = j + gap\_j)

{

temp = fabs(exact(x[i], y[j], T) - u[i][j][L]);

printf("(%.2f, %.2f, 1.00) numerical=%f, err=%.4e\n", x[i], y[j], u[i][j][L], temp);

}

}

free(x); free(y); free(t);

free(a1); free(b1); free(c1); free(d1);

free(a2); free(b2); free(c2); free(d2);

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

free(u[i][j]);

}

for (i = 0; i <= m; i++)

{

free(u[i]); free(v[i]);

}

free(u); free(v);

}

double f(double x, double y, double t)

{

return -3.0 \* exp((x + y) / 2.0 - t) / 2.0;

}

double phi(double x, double y)

{

return exp((x + y) / 2.0);

}

double g1(double y, double t)

{

return exp(y / 2.0 - t);

}

double g2(double y, double t)

{

return exp((1.0 + y) / 2.0 - t);

}

double g3(double x, double t)

{

return exp(x / 2.0 - t);

}

double g4(double x, double t)

{

return exp((1.0 + x) / 2.0 - t);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

double exact(double x, double y, double t)

{

return exp((x + y) / 2.0 - t);

}

Egch3\_sec6\_02.c

//求解二维抛物型方程初边值问题 : ADI 之 D'Yakonov 格式

#include"stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main()

{

double a, b, T, r1, r2, dx, dy, dt, \*x, \*y, \*t, \*\* \*u, \*\*v;

double \*a1, \*b1, \*c1, \*d1, \*a2, \*b2, \*c2, \*d2, tmid, \*ans, temp;

int i, j, k, m, n, L, gap\_i, gap\_j;

double f(double x, double y, double t);

double phi(double x, double y);

double g1(double y, double t);

double g2(double y, double t);

double g3(double x, double t);

double g4(double x, double t);

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

double exact(double x, double y, double t);

a = 1.0;

b = 1.0;

T = 1.0;

m = 60;

n = 40;

L = 20;

dx = a / m;

dy = b / n;

dt = T / L;

r1 = dt / (dx\*dx);

r2 = dt / (dy\*dy);

printf("m=%d, n=%d, L=%d\n", m, n, L);

printf("r1=%.4f, r2=%.4f\n", r1, r2);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*dx;

y = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= n; j++)

y[j] = j\*dy;

t = (double \*)malloc(sizeof(double)\*(L + 1));

for (k = 0; k <= L; k++)

t[k] = k\*dt;

u = (double \* \* \*)malloc(sizeof(double \*)\*((m + 1)\*(n + 1)\*(L + 1)));

for (i = 0; i <= m; i++)

u[i] = (double \* \*)malloc(sizeof(double \*)\*((n + 1)\*(L + 1)));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

u[i][j] = (double \*)malloc(sizeof(double)\*(L + 1));

}

v = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

v[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

{

u[i][j][0] = phi(x[i], y[j]); //初始条件

}

}

for (k = 1; k <= L; k++)

{

for (j = 0; j <= n; j++)

{

u[0][j][k] = g1(y[j], t[k]); //左边界条件

u[m][j][k] = g2(y[j], t[k]); //右边界条件

}

for (i = 1; i <= m - 1; i++)

{

u[i][0][k] = g3(x[i], t[k]); //下边界条件

u[i][n][k] = g4(x[i], t[k]); //上边界条件

}

}

a1 = (double \*)malloc(sizeof(double)\*(m - 1));

b1 = (double \*)malloc(sizeof(double)\*(m - 1));

c1 = (double \*)malloc(sizeof(double)\*(m - 1));

d1 = (double \*)malloc(sizeof(double)\*(m - 1));

for (i = 0; i< m - 1; i++)

{

a1[i] = -r1/2.0;

b1[i] = 1.0 + r1;

c1[i] = a1[i];

}

a2 = (double \*)malloc(sizeof(double)\*(n - 1));

b2 = (double \*)malloc(sizeof(double)\*(n - 1));

c2 = (double \*)malloc(sizeof(double)\*(n - 1));

d2 = (double \*)malloc(sizeof(double)\*(n - 1));

for (j = 0; j < n - 1; j++)

{

a2[j] = -r2/2.0;

b2[j] = 1.0 + r2;

c2[j] = a2[j];

}

for (k = 0; k < L; k++)

{

tmid = (t[k] + t[k + 1]) / 2.0;

for (j = 1; j <= n - 1; j++)//固定j

{

for (i = 1; i <= m - 1; i++)

{

temp = r2\*(1 - r1)\*(u[i][j - 1][k] + u[i][j + 1][k]) / 2.0 + r1\*(1 - r2)\*(u[i - 1][j][k] + u[i + 1][j][k]) / 2.0 + (1 - r1)\*(1 - r2)\*u[i][j][k];

d1[i-1] = temp + f(x[i], y[j], tmid)\*dt + r1\*r2\*(u[i - 1][j - 1][k] + u[i + 1][j - 1][k] + u[i - 1][j + 1][k] + u[i + 1][j + 1][k]) / 4.0;

}

v[0][j] = (1 + r2)\*u[0][j][k + 1] - r2\*(u[0][j - 1][k + 1] + u[0][j + 1][k + 1]) / 2.0;

v[m][j] = (1 + r2)\*u[m][j][k + 1] - r2\*(u[m][j - 1][k + 1] + u[m][j + 1][k + 1]) / 2.0;

d1[0] = d1[0] + r1\*v[0][j]/2.0;

d1[m - 2] = d1[m - 2] + r1\*v[m][j]/2.0;

ans = chase\_algorithm(a1, b1, c1, m - 1, d1);

for (i = 1; i <= m - 1; i++)

v[i][j] = ans[i - 1];

free(ans);

}

for (i = 1; i <= m - 1; i++)//固定i

{

for (j = 1; j <= n - 1; j++)

d2[j - 1] = v[i][j];

d2[0] = d2[0] + r2\*u[i][0][k + 1]/2.0;

d2[n - 2] = d2[n - 2] + r2\*u[i][n][k + 1]/2.0;

ans = chase\_algorithm(a2, b2, c2, n - 1, d2);

for (j = 1; j <= n - 1; j++)

u[i][j][k + 1] = ans[j - 1];

free(ans);

}

}//end for k

gap\_i = m / 4;//用于确定x方向每隔多少个点打印结果

gap\_j = n / 4;//用于确定y方向每隔多少个点打印结果

for (i = gap\_i; i <= m - 1; i = i + gap\_i)

{

for (j = gap\_j; j <= n - 1; j = j + gap\_j)

{

temp = fabs(exact(x[i], y[j], T / 2.0) - u[i][j][L / 2]);

printf("(%.2f, %.2f, 0.50) numerical=%f, err=%.4e\n", x[i], y[j], u[i][j][L / 2], temp);

}

}

printf("\n");

printf("\n");

for (i = gap\_i; i <= m - 1; i = i + gap\_i)

{

for (j = gap\_j; j <= n - 1; j = j + gap\_j)

{

temp = fabs(exact(x[i], y[j], T) - u[i][j][L]);

printf("(%.2f, %.2f, 1.00) numerical=%f, err=%.4e\n", x[i], y[j], u[i][j][L], temp);

}

}

free(x); free(y); free(t);

free(a1); free(b1); free(c1); free(d1);

free(a2); free(b2); free(c2); free(d2);

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

free(u[i][j]);

}

for (i = 0; i <= m; i++)

{

free(u[i]); free(v[i]);

}

free(u); free(v);

}

double f(double x, double y, double t)

{

return -3.0 \* exp((x + y) / 2.0 - t) / 2.0;

}

double phi(double x, double y)

{

return exp((x + y) / 2.0);

}

double g1(double y, double t)

{

return exp(y / 2.0 - t);

}

double g2(double y, double t)

{

return exp((1.0 + y) / 2.0 - t);

}

double g3(double x, double t)

{

return exp(x / 2.0 - t);

}

double g4(double x, double t)

{

return exp((1.0 + x) / 2.0 - t);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

double exact(double x, double y, double t)

{

return exp((x + y) / 2.0 - t);

}

Egch3\_sec6\_03.c

//求解二维抛物型方程初边值问题：ADI 之 Douglas 格式

#include"stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main()

{

double a, b, T, r1, r2, dx, dy, dt, \*x, \*y, \*t, \*\* \*u, \*\*v;

double \*a1, \*b1, \*c1, \*d1, \*a2, \*b2, \*c2, \*d2, tmid, \*ans, temp;

int i, j, k, m, n, L, gap\_i, gap\_j;

double f(double x, double y, double t);

double phi(double x, double y);

double g1(double y, double t);

double g2(double y, double t);

double g3(double x, double t);

double g4(double x, double t);

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

double exact(double x, double y, double t);

a = 1.0;

b = 1.0;

T = 1.0;

m = 60;

n = 40;

L = 20;

dx = a / m;

dy = b / n;

dt = T / L;

r1 = dt / (dx\*dx);

r2 = dt / (dy\*dy);

printf("m=%d, n=%d, L=%d\n", m, n, L);

printf("r1=%.4f, r2=%.4f\n", r1, r2);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*dx;

y = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= n; j++)

y[j] = j\*dy;

t = (double \*)malloc(sizeof(double)\*(L + 1));

for (k = 0; k <= L; k++)

t[k] = k\*dt;

u = (double \* \* \*)malloc(sizeof(double \*)\*((m + 1)\*(n + 1)\*(L + 1)));

for (i = 0; i <= m; i++)

u[i] = (double \* \*)malloc(sizeof(double \*)\*((n + 1)\*(L + 1)));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

u[i][j] = (double \*)malloc(sizeof(double)\*(L + 1));

}

v = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

v[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

{

u[i][j][0] = phi(x[i], y[j]); //初始条件

}

}

for (k = 1; k <= L; k++)

{

for (j = 0; j <= n; j++)

{

u[0][j][k] = g1(y[j], t[k]); //左边界条件

u[m][j][k] = g2(y[j], t[k]); //右边界条件

}

for (i = 1; i <= m - 1; i++)

{

u[i][0][k] = g3(x[i], t[k]); //下边界条件

u[i][n][k] = g4(x[i], t[k]); //上边界条件

}

}

a1 = (double \*)malloc(sizeof(double)\*(m - 1));

b1 = (double \*)malloc(sizeof(double)\*(m - 1));

c1 = (double \*)malloc(sizeof(double)\*(m - 1));

d1 = (double \*)malloc(sizeof(double)\*(m - 1));

for (i = 0; i< m - 1; i++)

{

a1[i] = -r1/2.0;

b1[i] = 1.0 + r1;

c1[i] = a1[i];

}

a2 = (double \*)malloc(sizeof(double)\*(n - 1));

b2 = (double \*)malloc(sizeof(double)\*(n - 1));

c2 = (double \*)malloc(sizeof(double)\*(n - 1));

d2 = (double \*)malloc(sizeof(double)\*(n - 1));

for (j = 0; j < n - 1; j++)

{

a2[j] = -r2/2.0;

b2[j] = 1.0 + r2;

c2[j] = a2[j];

}

for (k = 0; k < L; k++)

{

tmid = (t[k] + t[k + 1]) / 2.0;

for (j = 1; j <= n - 1; j++)//固定j

{

for (i = 1; i <= m - 1; i++)

{

d1[i - 1] = r1\*(u[i - 1][j][k] - 2 \* u[i][j][k] + u[i + 1][j][k])+ r2\*(u[i][j - 1][k] - 2 \* u[i][j][k] + u[i][j + 1][k]) + f(x[i], y[j], tmid)\*dt;

}

v[0][j] = (1 + r2)\*(u[0][j][k + 1] - u[0][j][k]) - r2\*(u[0][j + 1][k + 1] - u[0][j + 1][k] + u[0][j - 1][k + 1] - u[0][j - 1][k]) / 2.0;

v[m][j] = (1 + r2)\*(u[m][j][k + 1] - u[m][j][k]) - r2\*(u[m][j + 1][k + 1] - u[m][j + 1][k] + u[m][j - 1][k + 1] - u[m][j - 1][k]) / 2.0;

d1[0] = d1[0] + r1\*v[0][j]/2.0;

d1[m - 2] = d1[m - 2] + r1\*v[m][j]/2.0;

ans = chase\_algorithm(a1, b1, c1, m - 1, d1);

for (i = 1; i <= m - 1; i++)

v[i][j] = ans[i - 1];

free(ans);

}

for (i = 1; i <= m - 1; i++)//固定i

{

for (j = 1; j <= n - 1; j++)

d2[j - 1] = (1+r2)\*u[i][j][k]-r2\*(u[i][j-1][k]+u[i][j+1][k])/2.0+v[i][j];

d2[0] = d2[0] + r2\*u[i][0][k + 1]/2.0;

d2[n - 2] = d2[n - 2] + r2\*u[i][n][k + 1]/2.0;

ans = chase\_algorithm(a2, b2, c2, n - 1, d2);

for (j = 1; j <= n - 1; j++)

u[i][j][k + 1] = ans[j - 1];

free(ans);

}

}//end for k

gap\_i = m / 4;//用于确定x方向每隔多少个点打印结果

gap\_j = n / 4;//用于确定y方向每隔多少个点打印结果

for (i = gap\_i; i <= m - 1; i = i + gap\_i)

{

for (j = gap\_j; j <= n - 1; j = j + gap\_j)

{

temp = fabs(exact(x[i], y[j], T / 2.0) - u[i][j][L / 2]);

printf("(%.2f, %.2f, 0.50) numerical=%f, err=%.4e\n", x[i], y[j], u[i][j][L / 2], temp);

}

}

printf("\n");

printf("\n");

for (i = gap\_i; i <= m - 1; i = i + gap\_i)

{

for (j = gap\_j; j <= n - 1; j = j + gap\_j)

{

temp = fabs(exact(x[i], y[j], T) - u[i][j][L]);

printf("(%.2f, %.2f, 1.00) numerical=%f, err=%.4e\n", x[i], y[j], u[i][j][L], temp);

}

}

free(x); free(y); free(t);

free(a1); free(b1); free(c1); free(d1);

free(a2); free(b2); free(c2); free(d2);

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

free(u[i][j]);

}

for (i = 0; i <= m; i++)

{

free(u[i]); free(v[i]);

}

free(u); free(v);

}

double f(double x, double y, double t)

{

return -3.0 \* exp((x + y) / 2.0 - t) / 2.0;

}

double phi(double x, double y)

{

return exp((x + y) / 2.0);

}

double g1(double y, double t)

{

return exp(y / 2.0 - t);

}

double g2(double y, double t)

{

return exp((1.0 + y) / 2.0 - t);

}

double g3(double x, double t)

{

return exp(x / 2.0 - t);

}

double g4(double x, double t)

{

return exp((1.0 + x) / 2.0 - t);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

double exact(double x, double y, double t)

{

return exp((x + y) / 2.0 - t);

}

Egch3\_sec7\_01.c

//求解二维抛物型方程初边值问题

//**紧**差分之－交替方向 之 Douglas 格式, 关于时间二阶，关于空间四阶。

#include"stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main()

{

double a, b, T, r1, r2, dx, dy, dt, \*x, \*y, \*t, \*\* \*u, \*\*v,const11,const12,const21,const22;

double \*a1, \*b1, \*c1, \*d1, \*a2, \*b2, \*c2, \*d2, tmid, \*ans, temp,temp1,temp2;

int i, j, k, m, n, L, gap\_i, gap\_j;

double f(double x, double y, double t);

double phi(double x, double y);

double g1(double y, double t);

double g2(double y, double t);

double g3(double x, double t);

double g4(double x, double t);

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

double exact(double x, double y, double t);

a = 1.0;

b = 1.0;

T = 1.0;

m = 120;

n = 80;

L = 80;

dx = a / m;

dy = b / n;

dt = T / L;

r1 = dt / (dx\*dx);

r2 = dt / (dy\*dy);

printf("m=%d, n=%d, L=%d\n", m, n, L);

printf("r1=%.4f, r2=%.4f\n", r1, r2);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*dx;

y = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= n; j++)

y[j] = j\*dy;

t = (double \*)malloc(sizeof(double)\*(L + 1));

for (k = 0; k <= L; k++)

t[k] = k\*dt;

u = (double \* \* \*)malloc(sizeof(double \*)\*((m + 1)\*(n + 1)\*(L + 1)));

for (i = 0; i <= m; i++)

u[i] = (double \* \*)malloc(sizeof(double \*)\*((n + 1)\*(L + 1)));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

u[i][j] = (double \*)malloc(sizeof(double)\*(L + 1));

}

v = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

v[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

{

u[i][j][0] = phi(x[i], y[j]); //初始条件

}

}

for (k = 1; k <= L; k++)

{

for (j = 0; j <= n; j++)

{

u[0][j][k] = g1(y[j], t[k]); //左边界条件

u[m][j][k] = g2(y[j], t[k]); //右边界条件

}

for (i = 1; i <= m - 1; i++)

{

u[i][0][k] = g3(x[i], t[k]); //下边界条件

u[i][n][k] = g4(x[i], t[k]); //上边界条件

}

}

const11 = 1.0 / 12.0 - r1 / 2.0;

const12 = 10.0 / 12.0 + r1;

const21 = 1.0 / 12.0 - r2 / 2.0;

const22 = 10.0 / 12.0 + r2;

a1 = (double \*)malloc(sizeof(double)\*(m - 1));

b1 = (double \*)malloc(sizeof(double)\*(m - 1));

c1 = (double \*)malloc(sizeof(double)\*(m - 1));

d1 = (double \*)malloc(sizeof(double)\*(m - 1));

for (i = 0; i< m - 1; i++)

{

a1[i] =const11;

b1[i] =const12;

c1[i] = a1[i];

}

a2 = (double \*)malloc(sizeof(double)\*(n - 1));

b2 = (double \*)malloc(sizeof(double)\*(n - 1));

c2 = (double \*)malloc(sizeof(double)\*(n - 1));

d2 = (double \*)malloc(sizeof(double)\*(n - 1));

for (j = 0; j < n - 1; j++)

{

a2[j] = const21;

b2[j] = const22;

c2[j] = a2[j];

}

for (k = 0; k < L; k++)

{

tmid = (t[k] + t[k + 1]) / 2.0;

for (j = 1; j <= n - 1; j++)//固定j

{

for (i = 1; i <= m - 1; i++)

{

temp1 = u[i - 1][j - 1][k] - 2 \* u[i][j - 1][k] + u[i + 1][j - 1][k];

temp1 = temp1 + u[i - 1][j + 1][k] - 2 \* u[i][j + 1][k] + u[i + 1][j + 1][k];

temp1 = temp1 + 10 \* (u[i - 1][j][k] - 2 \* u[i][j][k] + u[i + 1][j][k]);

temp1 = r1\*temp1 / 12.0;

temp2 = u[i - 1][j - 1][k] - 2 \* u[i - 1][j][k] + u[i - 1][j + 1][k];

temp2 = temp2 + u[i + 1][j - 1][k] - 2 \* u[i + 1][j][k] + u[i + 1][j + 1][k];

temp2 = temp2 + 10 \* (u[i][j - 1][k] - 2 \* u[i][j][k] + u[i][j + 1][k]);

temp2 = r2\*temp2/12.0;

temp = f(x[i - 1], y[j - 1], tmid) + 10 \* f(x[i - 1], y[j], tmid) + f(x[i - 1], y[j + 1], tmid);

temp = temp + f(x[i + 1], y[j - 1], tmid) + 10 \* f(x[i + 1], y[j], tmid) + f(x[i + 1], y[j + 1], tmid);

temp = temp + 10 \* (f(x[i], y[j - 1], tmid) + 10 \* f(x[i], y[j], tmid) + f(x[i], y[j + 1], tmid));

temp = dt\*temp / 144.0;

d1[i - 1] = temp1+temp2 +temp;

}

v[0][j] = const21\*(u[0][j - 1][k + 1] + u[0][j + 1][k + 1] - u[0][j - 1][k] - u[0][j + 1][k]) + const22\*(u[0][j][k + 1] - u[0][j][k]);

v[m][j] = const21\*(u[m][j - 1][k + 1] + u[m][j + 1][k + 1] - u[m][j - 1][k] - u[m][j + 1][k]) + const22\*(u[m][j][k + 1] - u[m][j][k]);

d1[0] = d1[0] - const11\*v[0][j];

d1[m - 2] = d1[m - 2] - const11\*v[m][j];

ans = chase\_algorithm(a1, b1, c1, m - 1, d1);

for (i = 1; i <= m - 1; i++)

v[i][j] = ans[i - 1];

free(ans);

}

for (i = 1; i <= m - 1; i++)//固定i

{

for (j = 1; j <= n - 1; j++)

d2[j - 1] = const22\*u[i][j][k]+const21\*(u[i][j - 1][k] + u[i][j + 1][k])+ v[i][j];

d2[0] = d2[0] -const21\*u[i][0][k + 1];

d2[n - 2] = d2[n - 2]-const21\*u[i][n][k + 1];

ans = chase\_algorithm(a2, b2, c2, n - 1, d2);

for (j = 1; j <= n - 1; j++)

u[i][j][k + 1] = ans[j - 1];

free(ans);

}

}//end for k

gap\_i = m / 4;//用于确定x方向每隔多少个点打印结果

gap\_j = n / 4;//用于确定y方向每隔多少个点打印结果

for (i = gap\_i; i <= m - 1; i = i + gap\_i)

{

for (j = gap\_j; j <= n - 1; j = j + gap\_j)

{

temp = fabs(exact(x[i], y[j], T / 2.0) - u[i][j][L / 2]);

printf("(%.2f, %.2f, 0.50) numerical=%f, err=%.4e\n", x[i], y[j], u[i][j][L / 2], temp);

}

}

printf("\n");

printf("\n");

for (i = gap\_i; i <= m - 1; i = i + gap\_i)

{

for (j = gap\_j; j <= n - 1; j = j + gap\_j)

{

temp = fabs(exact(x[i], y[j], T) - u[i][j][L]);

printf("(%.2f, %.2f, 1.00) numerical=%f, err=%.4e\n", x[i], y[j], u[i][j][L], temp);

}

}

free(x); free(y); free(t);

free(a1); free(b1); free(c1); free(d1);

free(a2); free(b2); free(c2); free(d2);

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

free(u[i][j]);

}

for (i = 0; i <= m; i++)

{

free(u[i]); free(v[i]);

}

free(u); free(v);

}

double f(double x, double y, double t)

{

return -3.0 \* exp((x + y) / 2.0 - t) / 2.0;

}

double phi(double x, double y)

{

return exp((x + y) / 2.0);

}

double g1(double y, double t)

{

return exp(y / 2.0 - t);

}

double g2(double y, double t)

{

return exp((1.0 + y) / 2.0 - t);

}

double g3(double x, double t)

{

return exp(x / 2.0 - t);

}

double g4(double x, double t)

{

return exp((1.0 + x) / 2.0 - t);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

double exact(double x, double y, double t)

{

return exp((x + y) / 2.0 - t);

}

## 第四章

Egch4\_sec1\_01.c

// 这是用迎风格式求解对流方程，求解区间为x in [-1,2], t in [0,1]

//最后打印t=0.5时，x in [0,1]的数值解

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

double a, h, tau, r, \*x, \*t, \* \*u;

int j, k, m, n;

double phi(double x);

double exact(double x, double t);

FILE \*fp;

m = 300;

n = 200;

h = 3.0 / m;

tau = 1.0 / n;

a = 1.0;

r = a\*tau / h;

printf("r=%.4f\n", r);

if (r > 1.0)

{

printf("stability condition is not satisfied!\n");

return;

}

x = (double \*)malloc(sizeof(double)\*(m + 1));

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= m; j++)

x[j] = -1.0+j\*h;

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (j = 0; j <= m; j++)

u[j] = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= m; j++)

u[j][0] = phi(x[j]);

for (k = 0; k < n; k++)

{

for (j = k+1; j <= m; j++)

u[j][k + 1] = r\*u[j - 1][k] + (1.0 - r)\*u[j][k];

}

if (fopen\_s(&fp, "upwind.dat", "w")!= 0)//写入数据文件，到时将此文件拷贝到Matlab的工作目录下可以直接画图。

{

printf("cannot open the file!\n");

return;

}

for (j = 100; j <= 200; j++)

{

printf("x=%.2f, numerical=%.2f\n", x[j], u[j][n / 2]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][n / 2]); //写到dat文件中，第一列是x坐标，第二列是在相应x坐标处的u值。

//这样用matlab可以直接读取数据文件从而画出直观图来。

}

fclose(fp);

free(x); free(t);

for (j = 0; j <= m; j++)

free(u[j]);

free(u);

}

double phi(double x)

{

if (x <= 0)

return 0.0;

else

return 1.0;

}

double exact(double x, double t)

{

if (x <= t)

return 0.0;

else

return 1.0;

}

程序运行后在当前目录下会有一个数据文件upwind.dat，把数据文件upwind.dat文件拷到matlab软件的工作目录下，然后新建一个m脚本文件，文件内容为：

load upwind.dat

x=upwind(:,1);

u=upwind(:,2);

axis([0 1 -0.5 1.5]);

hold on;

plot(x,u);

xlabel('x');

ylabel('u');

title('upwind scheme at t=0.5');

运行此m文件即可作出迎风格式数值解的图。

Egch4\_sec1\_02.c

// 这是用Lax-Friedrichs格式求解对流方程，求解区间为x in [-1,2], t in [0,1]

//最后打印t=0.5时，x in [0,1]的数值解

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

double a, h, tau, r, \*x, \*t, \*\*u;

int j, k, m, n;

double phi(double x);

double exact(double x, double t);

FILE \*fp;

m = 300;

n = 200;

h = 3.0 / m;

tau = 1.0 / n;

a = 1.0;

r = a\*tau / h;

printf("r=%.4f\n", r);

if (r > 1.0)

{

printf("stability condition is not satisfied!\n");

return;

}

x = (double \*)malloc(sizeof(double)\*(m + 1));

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= m; j++)

x[j] = -1.0 + j\*h;

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (j = 0; j <= m; j++)

u[j] = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= m; j++)

u[j][0] = phi(x[j]);

for (k = 0; k < n; k++)

{

for (j = k + 1; j <= m-(k+1); j++)

u[j][k + 1] = (1.0+r)\*u[j - 1][k]/2.0 + (1.0 - r)\*u[j+1][k]/2;

}

if (fopen\_s(&fp, "Lax-Friedrichs.dat", "w") != 0)//写入数据文件，到时将此文件拷贝到Matlab的工作目录下可以直接画图。

{

printf("cannot open the file!\n");

return;

}

for (j = 100; j <= 200; j++)

{

printf("x=%.2f, numerical=%.2f\n", x[j], u[j][n / 2]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][n / 2]); //写到dat文件中，第一列是x坐标，第二列是在相应x坐标处的u值。

//这样用matlab可以直接读取数据文件从而画出直观图来。

}

fclose(fp);

free(x); free(t);

for (j = 0; j <= m; j++)

free(u[j]);

free(u);

}

double phi(double x)

{

if (x <= 0)

return 0.0;

else

return 1.0;

}

double exact(double x, double t)

{

if (x <= t)

return 0.0;

else

return 1.0;

}

程序运行后在当前目录下会有一个数据文件Lax\_Friedrichs.dat，把数据文件Lax\_Friedrichs.dat文件拷到matlab软件的工作目录下，然后新建一个m脚本文件，文件内容为：

load Lax\_Friedrichs.dat

x=Lax\_Friedrichs(:,1);

u=Lax\_Friedrichs(:,2);

axis([0 1 -0.5 1.5]);

hold on;

plot(x,u);

xlabel('x');

ylabel('u');

title('Lax-Friedrichs scheme at t=0.5');

运行此m文件即可作出Lax-Friedrichs格式数值解的图。

Egch4\_sec1\_03.c

// 这是用Lax-Wendroff格式求解对流方程，求解区间为x in [-1,2], t in [0,1]

//最后打印t=0.5时，x in [0,1]的数值解

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

double a, h, tau, r, \*x, \*t, \*\*u;

int j, k, m, n;

double phi(double x);

double exact(double x, double t);

FILE \*fp;

m = 300;

n = 200;

h = 3.0 / m;

tau = 1.0 / n;

a = 1.0;

r = a\*tau / h;

printf("r=%.4f\n", r);

if (r > 1.0)

{

printf("stability condition is not satisfied!\n");

return;

}

x = (double \*)malloc(sizeof(double)\*(m + 1));

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= m; j++)

x[j] = -1.0 + j\*h;

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (j = 0; j <= m; j++)

u[j] = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= m; j++)

u[j][0] = phi(x[j]);

for (k = 0; k < n; k++)

{

for (j = k + 1; j <= m - (k + 1); j++)

u[j][k + 1] = (1.0 + r)\*r\*u[j - 1][k] / 2.0 +(1.0-r\*r)\*u[j][k]+ r\*(r-1.0)\*u[j + 1][k] / 2;

}

if (fopen\_s(&fp, "Lax\_Wendroff.dat", "w") != 0)//写入数据文件，到时将此文件拷贝到Matlab的工作目录下可以直接画图。

{

printf("cannot open the file!\n");

return;

}

for (j = 100; j <= 200; j++)

{

printf("x=%.2f, numerical=%.2f\n", x[j], u[j][n / 2]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][n / 2]); //写到dat文件中，第一列是x坐标，第二列是在相应x坐标处的u值。

//这样用matlab可以直接读取数据文件从而画出直观图来。

}

fclose(fp);

free(x); free(t);

for (j = 0; j <= m; j++)

free(u[j]);

free(u);

}

double phi(double x)

{

if (x <= 0)

return 0.0;

else

return 1.0;

}

double exact(double x, double t)

{

if (x <= t)

return 0.0;

else

return 1.0;

}

程序运行后在当前目录下会有一个数据文件Lax\_Wendroff.dat，把数据文件Lax\_Wendroff.dat文件拷到matlab软件的工作目录下，然后新建一个m脚本文件，文件内容为：

load Lax\_Wendroff.dat

x=Lax\_Wendroff(:,1);

u=Lax\_Wendroff(:,2);

axis([0 1 -0.5 1.5]);

hold on;

plot(x,u);

xlabel('x');

ylabel('u');

title('Lax-Wendroff scheme at t=0.5');

运行此m文件即可作出Lax-Wendroff格式数值解的图。

Egch4\_sec1\_04.c

// 这是用Beam-Warming格式求解对流方程，求解区间为x in [-2,2], t in [0,1]

//最后打印t=0.5时，x in [0,1]的数值解

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

double a, h, tau, r, \*x, \*t, \*\*u;

int j, k, m, n;

double phi(double x);

double exact(double x, double t);

FILE \*fp;

m = 400;

n = 200;

h = 4.0 / m;

tau = 1.0 / n;

a = 1.0;

r = a\*tau / h;

printf("r=%.4f\n", r);

if (r > 2.0)

{

printf("stability condition is not satisfied!\n");

return;

}

x = (double \*)malloc(sizeof(double)\*(m + 1));

t = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= m; j++)

x[j] = -2.0 + j\*h;

for (k = 0; k <= n; k++)

t[k] = k\*tau;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (j = 0; j <= m; j++)

u[j] = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= m; j++)

u[j][0] = phi(x[j]);

for (k = 0; k < n; k++)

{

for (j = (k+1)\*2; j <= m; j++)

u[j][k + 1] = -r\*(1.0 - r)\*u[j - 2][k] / 2.0+ (2.0 - r)\*r\*u[j - 1][k] + (1.0 - r)\*(2.0 - r)\*u[j][k] / 2.0;

}

if (fopen\_s(&fp, "Beam\_Warming.dat", "w") != 0)//写入数据文件，到时将此文件拷贝到Matlab的工作目录下可以直接画图。

{

printf("cannot open the file!\n");

return;

}

for (j = 200; j <= 300; j++)

{

printf("x=%.2f, numerical=%.2f\n", x[j], u[j][n / 2]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][n / 2]); //写到dat文件中，第一列是x坐标，第二列是在相应x坐标处的u值。

//这样用matlab可以直接读取数据文件从而画出直观图来。

}

fclose(fp);

free(x); free(t);

for (j = 0; j <= m; j++)

free(u[j]);

free(u);

}

double phi(double x)

{

if (x <= 0)

return 0.0;

else

return 1.0;

}

double exact(double x, double t)

{

if (x <= t)

return 0.0;

else

return 1.0;

}

程序运行后在当前目录下会有一个数据文件Beam\_Warming.dat，把数据文件Beam\_Warming.dat文件拷到matlab软件的工作目录下，然后新建一个m脚本文件，文件内容为：

load Beam\_Warming.dat

x=Beam\_Warming(:,1);

u=Beam\_Warming(:,2);

axis([0 1 -0.5 1.5]);

hold on;

plot(x,u);

xlabel('x');

ylabel('u');

title('Beam-Warming scheme at t=0.5');

运行此m文件即可作出Beam-Warming格式数值解的图。

Egch4\_sec2\_01.c

//这是一个波动方程初边值问题的显格式程序，一阶收敛。

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

int m,n,i,j,k;

double pi,a,h,tau,r,\*x,\*t,\* \*u;

double phi(double x);

double psi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

pi=3.14159265359;

m=100;

n=50;

a=1.0;

h=pi/m;

tau=1.0/n;

r=a\*tau/h;

r=r\*r;

printf("r=%.4f\n",r);

x=(double \*)malloc(sizeof(double)\*(m+1));

for(i=0;i<=m;i++)

x[i]=i\*h;

t=(double \*)malloc(sizeof(double)\*(n+1));

for(k=0;k<=n;k++)

t[k]=k\*tau;

u=(double \* \*)malloc(sizeof(double \*)\*(m+1));

for(i=0;i<=m;i++)

u[i]=(double \*)malloc(sizeof(double)\*(n+1));

for(i=0;i<=m;i++)

{

u[i][0]=phi(x[i]); //initial condition

u[i][1]=phi(x[i])+tau\*psi(x[i]);

}

for(k=1;k<=n;k++)

{

u[0][k]=alpha(t[k]); //boundary condition

u[m][k]=beta(t[k]);

}

for(k=1;k<n;k++)

{

for(i=1;i<m;i++)

{

u[i][k+1]=r\*u[i-1][k]+2\*(1-r)\*u[i][k]+r\*u[i+1][k]-u[i][k-1]+tau\*tau\*f(x[i],t[k]);

}

}

k=4\*n/5; // 确定需要打印的时间层

j=m/10; //x方向每隔j个打印一下

for(i=j;i<m;i=i+j)

{

printf("(x,t)=(%.2f,%.2f), numerical=%f, error=%.4e\n",x[i],t[k],u[i][k], fabs(u[i][k]-exact(x[i],t[k])));

}

}

double phi(double x)

{

return sin(x);

}

double psi(double x)

{

return sin(x);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 0.0;

}

double f(double x,double t)

{

return 2\*sin(x)\*exp(t);

}

double exact(double x, double t)

{

return sin(x)\*exp(t);

}

Egch4\_sec2\_02.c

//这是一个波动方程初边值问题的三层显格式，二阶收敛。

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

int m,n,i,j,k;

double pi,a,h,tau,r,\*x,\*t,\* \*u;

double phi(double x);

double psi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

pi=3.14159265359;

m=100;

n=50;

a=1.0;

h=pi/m;

tau=1.0/n;

r=a\*tau/h;

r=r\*r;

printf("r=%.4f\n",r);

x=(double \*)malloc(sizeof(double)\*(m+1));

for(i=0;i<=m;i++)

x[i]=i\*h;

t=(double \*)malloc(sizeof(double)\*(n+1));

for(k=0;k<=n;k++)

t[k]=k\*tau;

u=(double \* \*)malloc(sizeof(double \*)\*(m+1));

for(i=0;i<=m;i++)

u[i]=(double \*)malloc(sizeof(double)\*(n+1));

for(i=0;i<=m;i++)

{

u[i][0]=phi(x[i]); //第0层时间层初始信息

}

for(k=1;k<=n;k++)

{

u[0][k]=alpha(t[k]); //boundary condition

u[m][k]=beta(t[k]);

}

for(i=1;i<m;i++) //第1层时间层上的信息

u[i][1]=(r\*u[i-1][0]+2\*(1-r)\*u[i][0]+r\*u[i+1][0]+tau\*tau\*f(x[i],t[0])+2\*tau\*psi(x[i]))/2.0;

for(k=1;k<n;k++)

{

for(i=1;i<m;i++)

{

u[i][k+1]=r\*u[i-1][k]+2\*(1-r)\*u[i][k]+r\*u[i+1][k]-u[i][k-1]+tau\*tau\*f(x[i],t[k]);

}

}

k=4\*n/5; //确定要打印的时间层

j=m/10; //x方向每隔j个打印一下。

for(i=j;i<m;i=i+j)

{

printf("(x,t)=(%.2f,%.2f), numerical=%f, error=%.4e\n",x[i],t[k],u[i][k], fabs(u[i][k]-exact(x[i],t[k])));

}

}

double phi(double x)

{

return sin(x);

}

double psi(double x)

{

return sin(x);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 0.0;

}

double f(double x,double t)

{

return 2\*sin(x)\*exp(t);

}

double exact(double x, double t)

{

return sin(x)\*exp(t);

}

Egch4\_sec3\_01.c

//这是一个波动方程初边值问题的隐格式程序，二阶收敛。

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

int m,n,i,j,k;

double a,pi,h,tau,r,\*x,\*t,\* \*u,\*a1,\*b,\*c,\*d,\*ans;

double phi(double x);

double ddphi(double x);

double psi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m=400;

n=100;

a=1.0;

pi=3.14159265359;

h=pi/m;

tau=1.0/n;

r=a\*tau/h;

r=r\*r;

printf("r=%.4f\n",r);

x=(double \*)malloc(sizeof(double)\*(m+1));

for(i=0;i<=m;i++)

x[i]=i\*h;

t=(double \*)malloc(sizeof(double)\*(n+1));

for(k=0;k<=n;k++)

t[k]=k\*tau;

u=(double \* \*)malloc(sizeof(double \*)\*(m+1));

for(i=0;i<=m;i++)

u[i]=(double \*)malloc(sizeof(double)\*(n+1));

for(i=0;i<=m;i++)

{

u[i][0]=phi(x[i]); //initial condition

}

for(k=1;k<=n;k++)

{

u[0][k]=alpha(t[k]); //boundary condition

u[m][k]=beta(t[k]);

}

for(i=1;i<m;i++) //第1层时间层上的信息

u[i][1]=(r\*u[i-1][0]+2\*(1-r)\*u[i][0]+r\*u[i+1][0]+tau\*tau\*f(x[i],t[0])+2\*tau\*psi(x[i]))/2.0;

a1=(double \*)malloc(sizeof(double)\*(m-1));

b=(double \*)malloc(sizeof(double)\*(m-1));

c=(double \*)malloc(sizeof(double)\*(m-1));

d=(double \*)malloc(sizeof(double)\*(m-1));

ans=(double \*)malloc(sizeof(double)\*(m-1));

for(k=1;k<n;k++)

{

for(i=1;i<m;i++)

{

d[i-1]=r\*(u[i-1][k-1]+u[i+1][k-1])/2.0-(1+r)\*u[i][k-1]+2\*u[i][k]+tau\*tau\*f(x[i],t[k]);

a1[i-1]=-0.5\*r;

b[i-1]=1.0+r;

c[i-1]=a1[i-1];

}

d[0]=d[0]+0.5\*r\*u[0][k+1];

d[m-2]=d[m-2]+0.5\*r\*u[m][k+1];

ans=chase\_algorithm(a1,b,c,m-1,d);

for(i=0;i<m-1;i++)

u[i+1][k+1]=ans[i];

}

free(ans);

k=4\*n/5; //确定要打印的时间层

j=m/10; //x方向每隔j个打印一下。

for(i=j;i<m;i=i+j)

{

printf("(x,t)=(%.2f,%.2f), numerical=%f, error=%.4e\n",x[i],t[k],u[i][k], fabs(u[i][k]-exact(x[i],t[k])));

}

free(a1); free(b); free(c); free(d); free(x); free(t);

}

double phi(double x)

{

return sin(x);

}

double psi(double x)

{

return sin(x);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 0.0;

}

double f(double x,double t)

{

return 2\*sin(x)\*exp(t);

}

double exact(double x, double t)

{

return sin(x)\*exp(t);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans,\*g,\*w,p;

int i;

ans=(double \*)malloc(sizeof(double)\*n);

g=(double \*)malloc(sizeof(double)\*n);

w=(double \*)malloc(sizeof(double)\*n);

g[0]=d[0]/b[0];

w[0]=c[0]/b[0];

for(i=1;i<n;i++)

{

p=b[i]-a[i]\*w[i-1];

g[i]=(d[i]-a[i]\*g[i-1])/p;

w[i]=c[i]/p;

}

ans[n-1]=g[n-1];

i=n-2;

do

{

ans[i]=g[i]-w[i]\*ans[i+1];

i=i-1;

//printf("i=%d, Now it's OK!\n",i);

}

while(i>=0);

free(g);

free(w);

return ans;

}

Egch4\_sec4\_01.c

//这是二阶双曲型方程的紧差分格式，关于时间二阶收敛，关于空间四阶收敛。

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

int m,n,i,j,k;

double a,pi,h,tau,r,\*x,\*t,\* \*u,\*a1,\*b,\*c,\*d,\*ans,c1,c2;

double phi(double x);

double ddphi(double x);

double psi(double x);

double alpha(double t);

double beta(double t);

double f(double x, double t);

double exact(double x, double t);

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m=200;

n=50;

a=1.0;

pi=3.14159265359;

h=pi/m;

tau=1.0/n;

r=a\*tau/h;

r=r\*r;

printf("r=%.4f\n",r);

x=(double \*)malloc(sizeof(double)\*(m+1));

for(i=0;i<=m;i++)

x[i]=i\*h;

t=(double \*)malloc(sizeof(double)\*(n+1));

for(k=0;k<=n;k++)

t[k]=k\*tau;

u=(double \* \*)malloc(sizeof(double \*)\*(m+1));

for(i=0;i<=m;i++)

u[i]=(double \*)malloc(sizeof(double)\*(n+1));

for(i=0;i<=m;i++)

{

u[i][0]=phi(x[i]); //initial condition

}

for(k=1;k<=n;k++)

{

u[0][k]=alpha(t[k]); //boundary condition

u[m][k]=beta(t[k]);

}

for(i=1;i<m;i++) //第1层时间层上的信息

u[i][1]=(r\*u[i-1][0]+2\*(1-r)\*u[i][0]+r\*u[i+1][0]+tau\*tau\*f(x[i],t[0])+2\*tau\*psi(x[i]))/2.0;

a1=(double \*)malloc(sizeof(double)\*(m-1));

b=(double \*)malloc(sizeof(double)\*(m-1));

c=(double \*)malloc(sizeof(double)\*(m-1));

d=(double \*)malloc(sizeof(double)\*(m-1));

ans=(double \*)malloc(sizeof(double)\*(m-1));

c1=1.0-6\*r;

c2=10.0+12\*r;

for(k=1;k<n;k++)

{

for(i=1;i<m;i++)

{

d[i-1]=(-c1)\*(u[i-1][k-1]+u[i+1][k-1])-c2\*u[i][k-1]+2\*(u[i-1][k]+10\*u[i][k]+u[i+1][k])+tau\*tau\*(f(x[i-1],t[k])+10\*f(x[i],t[k])+f(x[i+1],t[k]));

a1[i-1]=c1;

b[i-1]=c2;

c[i-1]=a1[i-1];

}

d[0]=d[0]-c1\*u[0][k+1];

d[m-2]=d[m-2]-c1\*u[m][k+1];

ans=chase\_algorithm(a1,b,c,m-1,d);

for(i=0;i<m-1;i++)

u[i+1][k+1]=ans[i];

}

free(ans);

k=4\*n/5; //确定要打印的时间层

j=m/10; //x方向每隔j个打印一下。

for(i=j;i<=m/2;i=i+j)

{

printf("(x,t)=(%.2f,%.2f), numerical=%f, error=%.4e\n",x[i],t[k],u[i][k], fabs(u[i][k]-exact(x[i],t[k])));

}

free(a1); free(b); free(c); free(d); free(x); free(t);

}

double phi(double x)

{

return sin(x);

}

double psi(double x)

{

return sin(x);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 0.0;

}

double f(double x,double t)

{

return 2\*sin(x)\*exp(t);

}

double exact(double x, double t)

{

return sin(x)\*exp(t);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans,\*g,\*w,p;

int i;

ans=(double \*)malloc(sizeof(double)\*n);

g=(double \*)malloc(sizeof(double)\*n);

w=(double \*)malloc(sizeof(double)\*n);

g[0]=d[0]/b[0];

w[0]=c[0]/b[0];

for(i=1;i<n;i++)

{

p=b[i]-a[i]\*w[i-1];

g[i]=(d[i]-a[i]\*g[i-1])/p;

w[i]=c[i]/p;

}

ans[n-1]=g[n-1];

i=n-2;

do

{

ans[i]=g[i]-w[i]\*ans[i+1];

i=i-1;

//printf("i=%d, Now it's OK!\n",i);

}

while(i>=0);

free(g);

free(w);

return ans;

}

Egch4\_sec5\_01.c

//求解二维双曲型方程初边值问题的交替方向隐格式，用的是教材（4－65）和 ( 4－66 ) 的格式

//这是右端未添加辅助项的交替格式（左端还是添加了一项的，为的是实现分解）

#include"stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main()

{

double a, b, T, r1, r2, dx, dy, dt, \*x, \*y, \*t, \*\* \*u, \*\*v;

double \*a1, \*b1, \*c1, \*d1, \*a2, \*b2, \*c2, \*d2, \*ans, temp;

int i, j, k, m, n, L, gap\_i, gap\_j;

double f(double x, double y, double t);

double phi(double x, double y);

double psi(double x, double y);

double g1(double y, double t);

double g2(double y, double t);

double g3(double x, double t);

double g4(double x, double t);

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

double exact(double x, double y, double t);

a = 1.0;

b = 1.0;

T = 1.0;

m = 20;

n = 40;

L = 80;

dx = a / m;

dy = b / n;

dt = T / L;

temp = dt / dx;

r1 = temp\*temp;

temp = dt / dy;

r2 = temp\*temp;

printf("m=%d, n=%d, L=%d\n", m, n, L);

printf("r1=%.4f, r2=%.4f\n", r1, r2);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*dx;

y = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= n; j++)

y[j] = j\*dy;

t = (double \*)malloc(sizeof(double)\*(L + 1));

for (k = 0; k <= L; k++)

t[k] = k\*dt;

u = (double \* \* \*)malloc(sizeof(double \* \*)\*((m + 1)\*(n + 1)\*(L + 1)));

v = (double \* \*)malloc(sizeof(double \*)\*((m + 1)\*(n + 1)));

for (i = 0; i <= m; i++)

{

u[i] = (double \* \*)malloc(sizeof(double \*)\*((n + 1)\*(L + 1)));

v[i] = (double \*)malloc(sizeof(double)\*(n + 1));

}

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

u[i][j] = (double \*)malloc(sizeof(double)\*(L + 1));

}

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

{

u[i][j][0] = phi(x[i], y[j]); //初始条件,第0层上的信息

}

}

for (i = 1; i< m; i++)

{

for (j = 1; j < n; j++)

{

u[i][j][1] = u[i][j][0] + dt\*psi(x[i], y[j]) + (r1\*(u[i - 1][j][0] - 2 \* u[i][j][0] + u[i + 1][j][0]) + r2\*(u[i][j - 1][0] - 2 \* u[i][j][0] + u[i][j + 1][0]) + dt\*dt\*f(x[i], y[j], t[0])) / 2.0; //初始条件,第1层上的信息

}

}

for (k = 1; k <= L; k++) //边界条件

{

for (j = 0; j <= n; j++)

{

u[0][j][k] = g1(y[j], t[k]);

u[m][j][k] = g2(y[j], t[k]);

}

for (i = 1; i <= m - 1; i++)

{

u[i][0][k] = g3(x[i], t[k]);

u[i][n][k] = g4(x[i], t[k]);

}

}

a1 = (double \*)malloc(sizeof(double)\*(m - 1));

b1 = (double \*)malloc(sizeof(double)\*(m - 1));

c1 = (double \*)malloc(sizeof(double)\*(m - 1));

d1 = (double \*)malloc(sizeof(double)\*(m - 1));

for (i = 0; i< m - 1; i++)

{

a1[i] = -r1 / 2.0;

b1[i] = 1.0 + r1;

c1[i] = a1[i];

}

a2 = (double \*)malloc(sizeof(double)\*(n - 1));

b2 = (double \*)malloc(sizeof(double)\*(n - 1));

c2 = (double \*)malloc(sizeof(double)\*(n - 1));

d2 = (double \*)malloc(sizeof(double)\*(n - 1));

for (j = 0; j < n - 1; j++)

{

a2[j] = -r2 / 2.0;

b2[j] = 1.0 + r2;

c2[j] = a2[j];

}

for (k = 1; k < L; k++)

{

for (j = 1; j <= n - 1; j++)//固定j

{

for (i = 1; i <= m - 1; i++)

{

d1[i-1]=2 \* u[i][j][k] + dt\*dt\*f(x[i], y[j], t[k])+r1\*(u[i-1][j][k-1]+u[i+1][j][k-1])/2.0+r2\*(u[i][j-1][k-1]+u[i][j+1][k-1])/2.0-(1+r1+r2)\*u[i][j][k-1]; //不增加右端辅助项时

}

v[0][j] = -r2\*(u[0][j - 1][k + 1] + u[0][j + 1][k + 1]) / 2.0 + (1 + r2)\*u[0][j][k + 1];

d1[0] = d1[0] + r1\*v[0][j] / 2.0;

v[m][j] = -r2\*(u[m][j - 1][k + 1] + u[m][j + 1][k + 1]) / 2.0 + (1 + r2)\*u[m][j][k + 1];

d1[m - 2] = d1[m - 2] + r1\*v[m][j] / 2.0;

ans = chase\_algorithm(a1, b1, c1, m - 1, d1);

for (i = 1; i <= m - 1; i++)

v[i][j] = ans[i - 1];

free(ans);

}

for (i = 1; i <= m - 1; i++)//固定i

{

for (j = 1; j <= n - 1; j++)

d2[j - 1] = v[i][j];

d2[0] = d2[0] + r2\*u[i][0][k + 1] / 2.0;

d2[n - 2] = d2[n - 2] + r2\*u[i][n][k + 1] / 2.0;

ans = chase\_algorithm(a2, b2, c2, n - 1, d2);

for (j = 1; j <= n - 1; j++)

u[i][j][k + 1] = ans[j - 1];

free(ans);

}

}//end for k

k = L ; //确定t=1.00时的时间层

gap\_i = m / 5;//用于确定x方向每隔多少个点打印结果

gap\_j = n / 4;//用于确定y方向每隔多少个点打印结果

for (i = gap\_i; i <= m - 1; i = i + gap\_i)

{

for (j = gap\_j; j <= n - 1; j = j + gap\_j)

{

temp = fabs(exact(x[i], y[j], t[k]) - u[i][j][k]);

printf("(%.2f, %.2f, %.2f) numerical=%f, err=%.4e\n", x[i], y[j], t[k], u[i][j][k], temp);

}

}

free(x); free(y); free(t);

free(a1); free(b1); free(c1); free(d1);

free(a2); free(b2); free(c2); free(d2);

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

free(u[i][j]);

}

for (i = 0; i <= m; i++)

{

free(u[i]);

}

free(u);

}

double f(double x, double y, double t)

{

double z,temp;

z = x\*x + y\*y;

temp = 1 - 2 \* z / (1 + z);

return 4\*t\*temp/((1+z)\*(1+z));

}

double phi(double x, double y)

{

return 0.0;

}

double psi(double x, double y)

{

return 1.0/(1+x\*x+y\*y);

}

double g1(double y, double t)

{

return t/(1+y\*y);

}

double g2(double y, double t)

{

return t/(2+y\*y);

}

double g3(double x, double t)

{

return t/(1+x\*x);

}

double g4(double x, double t)

{

return t/(2+x\*x);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

double exact(double x, double y, double t)

{

return t/(1+x\*x+y\*y);

}

Egch4\_sec6\_01.c

//二维双曲型－ADI－左端加了一个辅助项－紧差分，关于时间二阶，关于空间四阶。

//求解二维双曲方程初边值问题：紧差分之－交替方向 ADI

#include"stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main()

{

double a, b, T, r1, r2, dx, dy, dt, \*x, \*y, \*t, \*\* \*u, \*\*v, const11, const12, const21, const22;

double \*a1, \*b1, \*c1, \*d1, \*a2, \*b2, \*c2, \*d2, \*ans, temp;

int i, j, k, m, n, L, gap\_i, gap\_j;

double f(double x, double y, double t);

double phi(double x, double y);

double psi(double x, double y);

double g1(double y, double t);

double g2(double y, double t);

double g3(double x, double t);

double g4(double x, double t);

double epsilon\_x\_delta\_y(double \* \* \*u, int i, int j, int k);

double epsilon\_y\_delta\_x(double \* \* \*u, int i, int j, int k);

double epsilon\_x\_epsilon\_y(double \* \* \*u, int i, int j, int k);

double delta\_x\_delta\_y(double \* \* \*u, int i, int j, int k);

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

double exact(double x, double y, double t);

a = 1.0;

b = 1.0;

T = 1.0;

m = 10;

n = 20;

L = 40;

dx = a / m;

dy = b / n;

dt = T / L;

r1 = (dt\*dt) / (dx\*dx);

r2 = (dt\*dt) / (dy\*dy);

printf("m=%d, n=%d, L=%d\n", m, n, L);

printf("r1=%.4f, r2=%.4f\n", r1, r2);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = i\*dx;

y = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= n; j++)

y[j] = j\*dy;

t = (double \*)malloc(sizeof(double)\*(L + 1));

for (k = 0; k <= L; k++)

t[k] = k\*dt;

u = (double \* \* \*)malloc(sizeof(double \*)\*((m + 1)\*(n + 1)\*(L + 1)));

for (i = 0; i <= m; i++)

u[i] = (double \* \*)malloc(sizeof(double \*)\*((n + 1)\*(L + 1)));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

u[i][j] = (double \*)malloc(sizeof(double)\*(L + 1));

}

v = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

v[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

{

u[i][j][0] = phi(x[i], y[j]); //第0层初始条件

}

}

for (i = 1; i< m; i++)

{

for (j = 1; j < n; j++)

{

u[i][j][1] = u[i][j][0] + dt\*psi(x[i], y[j]) + (r1\*(u[i - 1][j][0] - 2 \* u[i][j][0] + u[i + 1][j][0]) + r2\*(u[i][j - 1][0] - 2 \* u[i][j][0] + u[i][j + 1][0]) + dt\*dt\*f(x[i], y[j], t[0])) / 2.0; //初始条件,第1层上的信息

}

}

for (k = 1; k <= L; k++)

{

for (j = 0; j <= n; j++)

{

u[0][j][k] = g1(y[j], t[k]); //左边界条件

u[m][j][k] = g2(y[j], t[k]); //右边界条件

}

for (i = 1; i <= m - 1; i++)

{

u[i][0][k] = g3(x[i], t[k]); //下边界条件

u[i][n][k] = g4(x[i], t[k]); //上边界条件

}

}

const11 = 1.0 / 12.0 - r1 / 2.0;

const12 = 10.0 / 12.0 + r1;

const21 = 1.0 / 12.0 - r2 / 2.0;

const22 = 10.0 / 12.0 + r2;

a1 = (double \*)malloc(sizeof(double)\*(m - 1));

b1 = (double \*)malloc(sizeof(double)\*(m - 1));

c1 = (double \*)malloc(sizeof(double)\*(m - 1));

d1 = (double \*)malloc(sizeof(double)\*(m - 1));

for (i = 0; i< m - 1; i++)

{

a1[i] = const11;

b1[i] = const12;

c1[i] = a1[i];

}

a2 = (double \*)malloc(sizeof(double)\*(n - 1));

b2 = (double \*)malloc(sizeof(double)\*(n - 1));

c2 = (double \*)malloc(sizeof(double)\*(n - 1));

d2 = (double \*)malloc(sizeof(double)\*(n - 1));

for (j = 0; j < n - 1; j++)

{

a2[j] = const21;

b2[j] = const22;

c2[j] = a2[j];

}

for (k = 1; k < L; k++)

{

for (j = 1; j <= n - 1; j++)//固定j

{

for (i = 1; i <= m - 1; i++)

{

temp = f(x[i - 1], y[j - 1], t[k]) + 10 \* f(x[i - 1], y[j], t[k]) + f(x[i - 1], y[j + 1], t[k]);

temp = temp + f(x[i + 1], y[j - 1], t[k]) + 10 \* f(x[i + 1], y[j], t[k]) + f(x[i + 1], y[j + 1], t[k]);

temp = temp + 10 \* (f(x[i], y[j - 1], t[k]) + 10 \* f(x[i], y[j], t[k]) + f(x[i], y[j + 1], t[k]));

temp = dt\*dt\*temp / 144.0;

temp = temp + r1\*epsilon\_y\_delta\_x(u, i, j, k-1) / 2.0 + r2\*epsilon\_x\_delta\_y(u, i, j, k-1) / 2.0- epsilon\_x\_epsilon\_y(u, i, j, k-1) + 2\*epsilon\_x\_epsilon\_y(u, i, j, k);

d1[i - 1] = temp;

}

v[0][j] = const21\*(u[0][j - 1][k + 1] + u[0][j + 1][k + 1]) + const22\*u[0][j][k + 1];

v[m][j] = const21\*(u[m][j - 1][k + 1] + u[m][j + 1][k + 1]) + const22\*u[m][j][k + 1];

d1[0] = d1[0] - const11\*v[0][j];

d1[m - 2] = d1[m - 2] - const11\*v[m][j];

ans = chase\_algorithm(a1, b1, c1, m - 1, d1);

for (i = 1; i <= m - 1; i++)

v[i][j] = ans[i - 1];

free(ans);

}

for (i = 1; i <= m - 1; i++)//固定i

{

for (j = 1; j <= n - 1; j++)

d2[j - 1] = v[i][j];

d2[0] = d2[0] - const21\*u[i][0][k + 1];

d2[n - 2] = d2[n - 2] - const21\*u[i][n][k + 1];

ans = chase\_algorithm(a2, b2, c2, n - 1, d2);

for (j = 1; j <= n - 1; j++)

u[i][j][k + 1] = ans[j - 1];

free(ans);

}

}//end for k

k = L; //确定t=1.00时的时间层

gap\_i = m / 5;//用于确定x方向每隔多少个点打印结果

gap\_j = n / 4;//用于确定y方向每隔多少个点打印结果

for (i = gap\_i; i <= m - 1; i = i + gap\_i)

{

for (j = gap\_j; j <= n - 1; j = j + gap\_j)

{

temp = fabs(exact(x[i], y[j], t[k]) - u[i][j][k]);

printf("(%.2f, %.2f, %.2f) numerical=%f, err=%.4e\n", x[i], y[j], t[k], u[i][j][k], temp);

}

}

free(x); free(y); free(t);

free(a1); free(b1); free(c1); free(d1);

free(a2); free(b2); free(c2); free(d2);

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

free(u[i][j]);

}

for (i = 0; i <= m; i++)

{

free(u[i]); free(v[i]);

}

free(u); free(v);

}

double f(double x, double y, double t)

{

double z, temp;

z = x\*x + y\*y;

temp = 1 - 2 \* z / (1 + z);

return 4 \* t\*temp / ((1 + z)\*(1 + z));

}

double phi(double x, double y)

{

return 0.0;

}

double psi(double x, double y)

{

return 1.0 / (1 + x\*x + y\*y);

}

double g1(double y, double t)

{

return t / (1 + y\*y);

}

double g2(double y, double t)

{

return t / (2 + y\*y);

}

double g3(double x, double t)

{

return t / (1 + x\*x);

}

double g4(double x, double t)

{

return t / (2 + x\*x);

}

double epsilon\_x\_delta\_y(double \* \* \*u, int i, int j, int k)

{

double temp1, temp2, temp3;

temp1 = u[i - 1][j - 1][k] + 10 \* u[i][j - 1][k] + u[i + 1][j - 1][k];

temp2 = u[i - 1][j][k] + 10 \* u[i][j][k] + u[i + 1][j][k];

temp3 = u[i - 1][j + 1][k] + 10 \* u[i][j + 1][k] + u[i + 1][j + 1][k];

return (temp1 + temp3 - 2 \* temp2) / 12.0;

}

double epsilon\_y\_delta\_x(double \* \* \*u, int i, int j, int k)

{

double temp1, temp2, temp3;

temp1 = u[i - 1][j - 1][k] + 10 \* u[i - 1][j][k] + u[i - 1][j + 1][k];

temp2 = u[i][j - 1][k] + 10 \* u[i][j][k] + u[i][j + 1][k];

temp3 = u[i + 1][j - 1][k] + 10 \* u[i + 1][j][k] + u[i + 1][j + 1][k];

return (temp1 + temp3 - 2 \* temp2) / 12.0;

}

double epsilon\_x\_epsilon\_y(double \* \* \*u, int i, int j, int k)

{

double temp1, temp2, temp3;

temp1 = u[i - 1][j - 1][k] + 10 \* u[i][j - 1][k] + u[i + 1][j - 1][k];

temp2 = u[i - 1][j][k] + 10 \* u[i][j][k] + u[i + 1][j][k];

temp3 = u[i - 1][j + 1][k] + 10 \* u[i][j + 1][k] + u[i + 1][j + 1][k];

return (temp1 + temp3 + 10 \* temp2) / 144.0;

}

double delta\_x\_delta\_y(double \* \* \*u, int i, int j, int k)

{

double temp1, temp2, temp3;

temp1 = u[i - 1][j - 1][k] - 2 \* u[i][j - 1][k] + u[i + 1][j - 1][k];

temp2 = u[i - 1][j][k] - 2 \* u[i][j][k] + u[i + 1][j][k];

temp3 = u[i - 1][j + 1][k] - 2 \* u[i][j + 1][k] + u[i + 1][j + 1][k];

return temp1 + temp3 - 2 \* temp2;

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

double exact(double x, double y, double t)

{

return t / (1 + x\*x + y\*y);

}

## 第五章

Egch5\_sec1\_01.c

// elliptic.cpp : 椭圆方程边值问题，五点菱形格式，二阶收敛。

#include "stdafx.h"

#include "math.h"

#include "stdlib.h"

#include "stdio.h"

double pi = 3.14159265359;

void main()

{

double xa, xb, ya, yb, dx, dy, alpha, beta, gamma, err, maxerr;

double \*x, \*y, \*\*u, \*\*temp;

int m, n, i, j, k;

double leftboundary(double y);

double rightboundary(double y);

double bottomboundary(double x);

double topboundary(double x);

double f(double x, double y);

double exact(double x, double y);

xa = 0.0;

xb = 2.0;

ya = 0.0;

yb = 1.0;

m = 128;

n = 64;

printf("m=%d, n=%d\n", m, n);

dx = (xb - xa) / m;

dy = (yb - ya) / n;

beta = 1.0 / (dx\*dx);

gamma = 1.0 / (dy\*dy);

alpha = 2 \* (beta + gamma);

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = xa + i\*dx;

y = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= n; j++)

y[j] = ya + j\*dy;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

temp = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

{

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

temp[i] = (double \*)malloc(sizeof(double)\*(n + 1));

}

for (j = 0; j <= n; j++)

{

u[0][j] = leftboundary(y[j]);

u[m][j] = rightboundary(y[j]);

}

for (i = 1; i<m; i++)

{

u[i][0] = bottomboundary(x[i]);

u[i][n] = topboundary(x[i]);

}

//设置迭代初值

for (i = 1; i<m; i++)

{

for (j = 1; j<n; j++)

u[i][j] = 0.0;

}

//temp二维数组用于存放u 数组在迭代过程中的中间值

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

temp[i][j] = u[i][j];

}

k = 0; //k为迭代计数器,此处用Gauss-Seidel迭代

do

{

maxerr = 0.0;

for (i = 1; i<m; i++)

{

for (j = 1; j<n; j++)

{

temp[i][j] = (f(x[i], y[j]) + beta\*(u[i - 1][j] + temp[i + 1][j]) + gamma\*(u[i][j - 1] + temp[i][j + 1])) / alpha;

err = temp[i][j] - u[i][j];

if (err>maxerr)

maxerr = err;

u[i][j] = temp[i][j];

}

}

k = k + 1;

} while (maxerr>0.5\*1e-10);

printf("k=%d\n", k);

k = m / 4;//确定x方向每隔k个打印一下

for (i = k; i < m; i = i + k)

{

printf("at (%.2f, 0.25), numerical=%f, err=%.4e\n", x[i], u[i][n / 4], fabs(exact(x[i], y[n / 4]) - u[i][n / 4]));//在y=0.25时的信息

}

k = m / 4;//确定x方向每隔k个打印一下

for (i = k; i < m; i = i + k)

{

printf("at (%.2f, 0.50), numerical=%f, err=%.4e\n", x[i], u[i][n / 2], fabs(exact(x[i], y[n / 2]) - u[i][n / 2]));//在y=0.5时的信息

}

}

double leftboundary(double y)

{

return sin(pi\*y);

}

double rightboundary(double y)

{

return exp(1.0)\*exp(1.0)\*sin(pi\*y);

}

double bottomboundary(double x)

{

return 0.0;

}

double topboundary(double x)

{

return 0.0;

}

double f(double x, double y)

{

return (pi\*pi - 1)\*exp(x)\*sin(pi\*y);

}

double exact(double x, double y)

{

return exp(x)\*sin(pi\*y);

}

Egch5\_sec2\_01.c

// 这是椭圆型方程边值问题的九点紧差分格式，四阶收敛。

#include "stdafx.h"

#include "math.h"

#include "stdlib.h"

#include "stdio.h"

double pi = 3.14159265359;

void main()

{

double xa, xb, ya, yb, dx, dy, alpha, beta, gamma, err, maxerr;

double \*x, \*y, \*\*u, \*\*g, \*\*temp, kexi,eta1,eta2;

int m, n, i, j, k;

double leftboundary(double y);

double rightboundary(double y);

double bottomboundary(double x);

double topboundary(double x);

double f(double x, double y);

double \* \*Gij(double \*x, double \*y, int m, int n);

double exact(double x, double y);

xa = 0.0;

xb = 2.0;

ya = 0.0;

yb = 1.0;

m = 64;

n = 32;

printf("m=%d, n=%d\n", m, n);

dx = (xb - xa) / m;

dy = (yb - ya) / n;

beta = 1.0 / (dx\*dx);

gamma = 1.0 / (dy\*dy);

kexi = beta + gamma;

eta1 = 10 \* beta - 2 \* gamma;

eta2 = 10 \* gamma - 2 \* beta;

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = xa + i\*dx;

y = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= n; j++)

y[j] = ya + j\*dy;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

temp = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

{

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

temp[i] = (double \*)malloc(sizeof(double)\*(n + 1));

}

for (j = 0; j <= n; j++)

{

u[0][j] = leftboundary(y[j]);

u[m][j] = rightboundary(y[j]);

}

for (i = 1; i<m; i++)

{

u[i][0] = bottomboundary(x[i]);

u[i][n] = topboundary(x[i]);

}

//设置迭代初值

for (i = 1; i<m; i++)

{

for (j = 1; j<n; j++)

u[i][j] = 0.0;

}

g = Gij(x, y, m, n);

//temp二维数组用于存放u 数组在迭代过程中的中间值,也就是旧值

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

temp[i][j] = u[i][j];

}

k = 0; //k为迭代计数器,此处用Gauss-Seidel迭代

do

{

maxerr = 0.0;

for (i = 1; i<m; i++)

{

for (j = 1; j<n; j++)

{

temp[i][j] = (g[i][j]-kexi\*(u[i-1][j-1]+temp[i-1][j+1]+u[i+1][j-1]+temp[i+1][j+1])-eta1\*(u[i-1][j]+temp[i+1][j])-eta2\*(u[i][j-1]+temp[i][j+1])) / (-20\*kexi);

err = temp[i][j] - u[i][j];

if (err>maxerr)

maxerr = err;

u[i][j] = temp[i][j];

}

}

k = k + 1;

} while (maxerr>0.5\*1e-10);

printf("k=%d\n", k);

k = m / 4;//确定x方向每隔k个打印一下

for (i = k; i < m; i = i + k)

{

printf("at (%.2f, 0.25), numerical=%f, err=%.4e\n", x[i], u[i][n / 4], fabs(exact(x[i], y[n / 4]) - u[i][n / 4]));//在y=0.25时的信息

}

k = m / 4;//确定x方向每隔k个打印一下

for (i = k; i < m; i = i + k)

{

printf("at (%.2f, 0.50), numerical=%f, err=%.4e\n", x[i], u[i][n / 2], fabs(exact(x[i], y[n / 2]) - u[i][n / 2]));//在y=0.5时的信息

}

free(x); free(y);

}

double leftboundary(double y)

{

return sin(pi\*y);

}

double rightboundary(double y)

{

return exp(1.0)\*exp(1.0)\*sin(pi\*y);

}

double bottomboundary(double x)

{

return 0.0;

}

double topboundary(double x)

{

return 0.0;

}

double f(double x, double y)

{

return (pi\*pi - 1)\*exp(x)\*sin(pi\*y);

}

double \* \*Gij(double \*x, double \*y, int m, int n) //计算-12 epsilon\_x\_epsilon\_y f\_ij

{

double \* \*ans,temp1,temp2,temp3;

int i, j;

ans = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

ans[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i <= m; i++)

{

for (j = 0; j <= n; j++)

ans[i][j] = 0.0;

}

for (i = 1; i < m; i++)

{

for (j = 1; j < n; j++)

{

temp1 = f(x[i - 1], y[j - 1]) + 10 \* f(x[i], y[j - 1]) + f(x[i + 1], y[j - 1]);

temp2 = f(x[i - 1], y[j]) + 10 \* f(x[i], y[j]) + f(x[i + 1], y[j]);

temp3 = f(x[i - 1], y[j + 1]) + 10 \* f(x[i], y[j + 1]) + f(x[i + 1], y[j + 1]);

ans[i][j] = -(temp1 + temp3 + 10 \* temp2) / 12.0;

}

}

return ans;

}

double exact(double x, double y)

{

return exp(x)\*sin(pi\*y);

}

Egch5\_sec3\_01.c

//椭圆方程混合边界条件的差分格式，二阶收敛。

//

#include "stdafx.h"

#include "math.h"

#include "stdlib.h"

#include "stdio.h"

double pi = 3.14159265359;

void main()

{

double xa, xb, ya, yb, dx, dy, alpha, beta, gamma, maxerr;

double \*x, \*y, \*\*u, \*\*v, \*\*lambda, kexi, eta, \*d, temp;

int m, n, i, j, k;

double f(double x, double y);

double lambda\_function(double x, double y);

double phi1(double y);

double phi2(double y);

double psi1(double x);

double psi2(double x);

double exact(double x, double y);

xa = 0.0;

xb = 2.0;

ya = 0.0;

yb = 1.0;

m = 64;

n = 32;

printf("m=%d, n=%d\n", m, n);

dx = (xb - xa) / m;

dy = (yb - ya) / n;

beta = 1.0 / (dx\*dx);

gamma = 1.0 / (dy\*dy);

alpha = 2 \* (beta + gamma);

kexi = 2.0 / dx;

eta = 2.0 / dy;

x = (double \*)malloc(sizeof(double)\*(m + 1));

for (i = 0; i <= m; i++)

x[i] = xa + i\*dx;

y = (double \*)malloc(sizeof(double)\*(n + 1));

for (j = 0; j <= n; j++)

y[j] = ya + j\*dy;

u = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

v = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

lambda = (double \* \*)malloc(sizeof(double \*)\*(m + 1));

for (i = 0; i <= m; i++)

{

u[i] = (double \*)malloc(sizeof(double)\*(n + 1));

v[i] = (double \*)malloc(sizeof(double)\*(n + 1));

lambda[i] = (double \*)malloc(sizeof(double)\*(n + 1));

}

//设置迭代初值,u\_ij放旧值，v\_ij放新值。

for (i = 0; i<= m; i++)

{

for (j = 0; j <= n; j++)

{

u[i][j] = 0.0;

v[i][j] = 0.0;

lambda[i][j] = lambda\_function(x[i], y[j]);

}

}

d = (double \*)malloc(sizeof(double)\*(m + 1));//用于放置右端项

k = 0; //k为迭代计数器,此处用Gauss-Seidel迭代

do

{

maxerr = 0.0;

for (i = 0; i <= m; i++)

d[i] = f(x[i], y[0]) - eta\*psi1(x[i]);

d[0] = d[0] - kexi\*phi1(y[0]);

d[m] = d[m] + kexi\*phi2(y[0]);

v[0][0] = (d[0] + 2 \* gamma\*u[0][1] + 2 \* beta\*u[1][0]) / (alpha + (kexi + eta)\*lambda[0][0]);

for (i = 1; i < m; i++)

v[i][0] = (d[i] + 2 \* gamma\*u[i][1] + beta\*(v[i - 1][0] + u[i + 1][0])) / (alpha + eta\*lambda[i][0]);

v[m][0] = (d[m] + 2 \* gamma\*u[m][1] + 2 \* beta\*v[m - 1][0]) / (alpha + (kexi + eta)\*lambda[m][0]);

for (j = 1; j < n; j++)

{

for (i = 0; i <= m; i++)

d[i] = f(x[i], y[j]);

d[0] = d[0] - kexi\*phi1(y[j]);

d[m] = d[m] + kexi\*phi2(y[j]);

v[0][j] = (d[0] + gamma\*(u[0][j + 1] + v[0][j - 1]) + 2 \* beta\*u[1][j]) / (alpha + kexi\*lambda[0][j]);

for (i = 1; i < m; i++)

v[i][j] = (d[i] + gamma\*(v[i][j - 1] + u[i][j + 1]) + beta\*(v[i - 1][j] + u[i + 1][j])) / alpha;

v[m][j]= (d[m] + gamma\*(v[m][j - 1] + u[m][j + 1]) + 2 \* beta\*v[m - 1][j]) / (alpha + kexi\*lambda[m][j]);

}

for (i = 0; i<=m; i++)

d[i] = f(x[i], y[n]) + eta\*psi2(x[i]);

d[0] = d[0] - kexi\*phi1(y[n]);

d[m] = d[m] + kexi\*phi2(y[n]);

v[0][n] = (d[0] + 2 \* beta\*u[1][n] + 2 \* gamma\*v[0][n - 1]) / (alpha + (kexi+eta)\*lambda[0][n]);

for (i = 1; i < m; i++)

v[i][n] = (d[i] + beta\*(v[i - 1][n] + u[i + 1][n]) + 2 \* gamma\*v[i][n - 1]) / (alpha + eta\*lambda[i][n]);

v[m][n] = (d[m] + 2 \* beta\*v[m - 1][n] + 2 \* gamma\*v[m][n - 1]) / (alpha + (kexi + eta)\*lambda[m][n]);

for (i = 0; i<=m; i++)

{

for (j = 0; j<=n; j++)

{

temp = fabs(u[i][j] - v[i][j]);

if (temp>maxerr)

maxerr = temp;

u[i][j] = v[i][j];

}

}

k = k + 1;

} while ((maxerr>0.5\*1e-10)&&(k<=1e+8));

printf("k=%d\n", k);

k = m / 4;//确定x方向每隔k个打印一下

for (i = k; i < m; i = i + k)

{

printf("at (%.2f, 0.25), numerical=%f, err=%.4e\n", x[i], u[i][n / 4], fabs(exact(x[i], y[n / 4]) - u[i][n / 4]));//在y=0.25时的信息

}

k = m / 4;//确定x方向每隔k个打印一下

for (i = k; i < m; i = i + k)

{

printf("at (%.2f, 0.50), numerical=%f, err=%.4e\n", x[i], u[i][n / 2], fabs(exact(x[i], y[n / 2]) - u[i][n / 2]));//在y=0.5时的信息

}

free(x); free(y); free(d);

for (i = 0; i <= m; i++)

{

free(u[i]); free(v[i]); free(lambda[i]);

}

free(u); free(v); free(lambda);

}

double f(double x, double y)

{

return (pi\*pi - 1)\*exp(x)\*sin(pi\*y);

}

double lambda\_function(double x, double y)

{

return x\*x + y\*y;

}

double phi1(double y)

{

return (1 - y\*y)\*sin(pi\*y);

}

double phi2(double y)

{

return (5.0 + y\*y)\*exp(2.0)\*sin(pi\*y);

}

double psi1(double x)

{

return pi\*exp(x);

}

double psi2(double x)

{

return -pi\*exp(x);

}

double exact(double x, double y)

{

return exp(x)\*sin(pi\*y);

}

## 第六章

Egch6\_sec1\_01.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

int \* \*lnd; //lnd放置单元的节点编号

double \*xco, \*u; //数组xco放置节点的坐标，数组u放置节点的数值解

double h; //h 为步长

double pi = 3.14159265359; //定义圆周率pi

double gauss = 0.5773502692; //数值积分中的高斯点

void main()

{

int m, elem, i, j, k, t, row, coln, e;

double ea[2][2], alpha[2],g[2],sum1, sum2;

double \* \*a, \*b, \*a1, \*b1, \*c1;

double f(double x);

double phi(int i, double x);

double fun1(int i, double x);

double exact(double x);

double fun2(int i, double x);

double dxexact(double x);

double fun3(int i, double x);

double integral(double a, double b, int i, double (\* fun)(int, double));

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m = 16; //总的剖分数

printf("m=%d\n", m);

elem = m; //总的单元数

h = 1.0 / m; //网格尺度即空间步长

xco = (double \*)malloc(sizeof(double)\*(m+1));//为节点坐标动态分配内存，放置在数组xco[]中

for (i = 0; i<=m; i++)

xco[i] = i\*h; //节点坐标

lnd = (int \*\*)malloc(sizeof(int \*)\*elem);

for (e = 0; e<elem; e++)

lnd[e] = (int \*)malloc(sizeof(int)\* 2);

//为单元节点编号动态分配内存，放置在二维数组lnd[]中，lnd[e][i]表示第e个单元

//第i个节点，e是整体单元编号，i是局部节点编号，lnd[e][i]是整体节点编号。

for (e = 0; e<elem; e++)

{

lnd[e][0] = e;

lnd[e][1] = e + 1;

} //即编号为e的单元上的第0个节点编号为e，第一个节点编号为e+1

a = (double \*\*)malloc(sizeof(double \*)\*(m+1)); //动态分配内存,二维数组a[ ][ ]放置总刚度矩阵，即线性方程组系数矩阵

for (i = 0; i<=m; i++)

a[i] = (double \*)malloc(sizeof(double)\*(m+1));

b = (double \*)malloc(sizeof(double)\*(m+1));//一维数组b[ ]放置总荷载即线性方程组右端项

for (i = 0; i<=m; i++) //初始化系数矩阵及右端项

{

for (j = 0; j<=m; j++)

a[i][j] = 0;

b[i] = 0;

}

//计算单元刚度矩阵

alpha[0] = -1.0 / h;//phi0'(x)

alpha[1] = 1.0 / h; //phi1'(x)

for (i = 0; i<2; i++)

for (j = 0; j<2; j++)

ea[i][j] = alpha[i] \* alpha[j] \* h;

for (e = 0; e<elem; e++)

{

i = lnd[e][0];

j = lnd[e][1];

//利用两点高斯数值积分公式计算单元荷载

for (k = 0; k <= 1; k++)

g[k] = integral(xco[i], xco[j], lnd[e][k], fun1);//放置单元荷载

// 合成整体刚度矩阵

for (i = 0; i<2; i++)

{

for (j = 0; j<2; j++)

{

row = lnd[e][i]; //确定整体行编号以明确合成总刚度矩阵时的位置

coln = lnd[e][j]; //确定整体列编号以明确合成总刚度矩阵时的位置

a[row][coln] = a[row][coln] + ea[i][j];

}

k = lnd[e][i]; //确定右端项行编号以明确合成总荷载时的位置

b[k] = g[i] + b[k]; //合成总荷载即整体右端项

}

}//完成e循环

//修改边界条件

for (t = 0; t<=m; t++)

{

a[t][0] = 0; a[0][t] = 0; a[t][m] = 0; a[m][t] = 0;

}

a[0][0] = 1; a[m][m] = 1;b[0] = 0; b[m] = 0;

a1 = (double \*)malloc(sizeof(double)\*(m + 1));//存储矩阵a[ ][ ]中的下次对角线元素

b1 = (double \*)malloc(sizeof(double)\*(m + 1));//存储矩阵a[ ][ ]中的主对角线元素

c1 = (double \*)malloc(sizeof(double)\*(m + 1));//存储矩阵a[ ][ ]中的上次对角线元素

a1[0] = 0.0;

c1[m] = 0.0;

for (i = 0; i <=m ; i++)

{

if (i!=0)

a1[i] = a[i][i - 1];

if (i!=m)

c1[i] = a[i][i + 1];

b1[i] = a[i][i];

}

for (i = 0; i <= m; i++)

free(a[i]);

free(a);

u=chase\_algorithm(a1, b1, c1,m+1, b);//追赶法求解三对角线性方程组

free(a1);free(b1);free(c1);free(b);

k = m / 8;//确定每隔几个点打印一次

for (i = k; i<m; i=i+k) //打印在各节点处的数值解并与精确解作比较

printf("xco[%d]=%.4f, u[%d]=%f, err=%.4e\n", i, xco[i], i, u[i], fabs(u[i] - exact(xco[i])));

//计算数值解与精确解在范数下的误差

sum1 = 0.0;

sum2 = 0.0;

for (e = 0; e < elem; e++)

{

i = lnd[e][0];

j = lnd[e][1];

sum1 = sum1 + integral(xco[i], xco[j], i, fun2);

sum2 = sum2 + integral(xco[i], xco[j], i, fun3);

}

printf("||u-uh||=%f, ||(u-uh)'||=%f\n", sqrt(sum1), sqrt(sum2));

for(e=0;e<elem;e++)

free(lnd[e]);

free(lnd);free(xco);free(u);

}

double f(double x) //右端项f(x)

{

double z;

z = ((16 \* pi\*pi - 4)\*sin(4 \* pi\*x) - 16 \* pi\*cos(4 \* pi\*x))\*exp(2 \* x);

return z;

}

double phi(int i, double x) //基函数

{

double temp,z;

temp = fabs(x - xco[i]);

if (temp <= h)

z = 1.0 - temp / h;

else

z = 0.0;

return z;

}

double fun1(int i, double x)//算单元荷载时的被积函数

{

return f(x)\*phi(i, x);

}

double exact(double x)//精确解u(x)

{

return sin(4 \* pi\*x)\*exp(2 \* x);

}

double fun2(int i, double x)//算||u-uh||^2时的被积函数

{

double temp, z;

temp = u[i] \* phi(i, x) + u[i + 1] \* phi(i+1,x);

z = exact(x) - temp;

return z\*z;

}

double dxexact(double x)//u(x)的导函数

{

return (2 \* pi\*cos(4 \* pi\*x) + sin(4 \* pi\*x))\*exp(2 \* x) \* 2;

}

double fun3(int i, double x)//算||(u-uh)'||^2时的被积函数

{

double temp, z;

temp = (u[i + 1] - u[i]) / h;

z = dxexact(x) - temp;

return z\*z;

}

double integral(double a, double b, int i, double (\* fun)(int,double))//在区间[a,b]上对被积函数fun(i,x)进行数值积分（两点高斯公式）

{

double mid, w, ans;

mid = (b + a) / 2.0;

w = (b - a) / 2.0;

ans = w\*((\* fun)(i, mid + w\*gauss) + (\* fun)(i, mid - w\*gauss));

return ans;

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d)

{

double \* ans, \*g, \*w, p;

int i;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i >= 0);

free(g);

free(w);

return ans;

}

Egch6\_sec4\_01.c

//二阶椭圆型方程零边值问题(二维矩形区域）的程序（L2范数2阶收敛，H1范数1阶收敛）

#include"stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

intm,n;

intnode,elem,limnode;

int \* \*lnd;

double area,\*xco,\*yco,\*u;

void main()

{

int \*limnd;

inti,j,k,t,e,row,coln;

doublex,y,q,sum,summ;

doublea,b,c,d,dx,dy;

double \* \*A, \*rhs, \*w, \*graduh;

double alpha[3],beta[3],ea[3][3],g[3];

double v(doublex,double y);

doublevx(double x, double y);

doublevy(double x, double y);

double f(doublex,double y);

double \*fun\_lambda(int e, double x, double y);

double \*d\_lambda(int e);

double uh(int e, double x, double y);

double \*d\_uh(int e);

double \* GaussElimination(double \* \*a, double \*b, int N);

a=0.0;//x方向的求解区域[a,b]

b=1.0;

c=0.0;//y方向的求解区域[c,d]

d=1.0;

m=32;//m,n分别为x方向和y方向的剖分数

n=32;

printf("m=%d, n=%d\n",m,n);

dx=(b-a)/m; //x方向上的步长

dy=(d-c)/n; //y方向上的步长

node=(m+1)\*(n+1);//总节点数

elem=2\*m\*n;//总单元数（采用三角形剖分）

limnode=2\*(m+n); //边界节点数（受限节点数）

area=(b-a)\*(d-c)/elem; //单元面积

limnd=(int \*)malloc(sizeof(int)\*(limnode));

for(i=0;i<=m;i++) // 边界节点的编号

{

limnd[i]=i; //下底边节点编号

limnd[limnode-i-1]=node-1-i;//上顶边节点编号

}//此时i=m+1

for(j=1;j<n;j++)

{

limnd[i]=j\*(m+1); //左侧边上的节点编号

limnd[i+1]=limnd[i]+m;//右侧边上的节点编号

i=i+2;

}

//各单元局部节点编号与整体编号之间的关系

lnd=(int \* \*)malloc(sizeof(int \*)\*elem);

for(i=0;i<elem;i++)

lnd[i]=(int \*)malloc(sizeof(int)\*3);

lnd[0][0]=0;//0号单元的三个节点局部编号是0,1,2,整体编号是0,1,m+1

lnd[0][1]=1;

lnd[0][2]=m+1;

lnd[1][0]=m+2;//1号单元的三个节点局部编号是0,1,2,整体编号是m+2,m+1,1

lnd[1][1]=m+1;

lnd[1][2]=1;

for(e=2;e<2\*m;e++) //2~2m-1号单元的节点编号情况

{

for(i=0;i<3;i++)

lnd[e][i]=lnd[e-2][i]+1;

}

for(e=2\*m;e<elem;e++)//2m-1~elem-1号单元的节点编号情况

{

for(i=0;i<3;i++)

lnd[e][i]=lnd[e-2\*m][i]+m+1;

}

//各节点的坐标

xco=(double \*)malloc(sizeof(double)\*node);

yco=(double \*)malloc(sizeof(double)\*node);

for(i=0;i<=m;i++)

{

xco[i]=a+i\*dx;

yco[i]=c;

for(j=1;j<=n;j++)

{

t=j\*(m+1)+i;

xco[t]=xco[i];

yco[t]=c+j\*dy;

}

}

//初始化刚度矩阵A及右端项rhs(right hand side缩写）

A=(double \* \*)malloc(sizeof(double \*)\*node);

for(i=0;i<node;i++)

A[i]=(double \*)malloc(sizeof(double)\*node);

rhs=(double \*)malloc(sizeof(double)\*node);

u=(double \*)malloc(sizeof(double)\*node);

for(i=0;i<node;i++)

{

rhs[i]=0.0;

for(j=0;j<node;j++)

A[i][j]=0.0;

}

for(e=0;e<elem;e++)

{

i=lnd[e][0];

j=lnd[e][1];

k=lnd[e][2];

g[0]=area\*(f((xco[i]+xco[k])/2,(yco[i]+yco[k])/2)/2+f((xco[i]+xco[j])/2,(yco[i]+yco[j])/2)/2)/3;//单元荷载

g[1]=area\*(f((xco[j]+xco[k])/2,(yco[j]+yco[k])/2)/2+f((xco[i]+xco[j])/2,(yco[i]+yco[j])/2)/2)/3;

g[2]=area\*(f((xco[i]+xco[k])/2,(yco[i]+yco[k])/2)/2+f((xco[k]+xco[j])/2,(yco[k]+yco[j])/2)/2)/3;

w=d\_lambda(e);

for (i=0;i<3;i++) //计算单元刚度矩阵

for (j=0;j<3;j++)

ea[i][j]=(w[i]\*w[j]+w[i+3]\*w[j+3])\*area;

for(i=0;i<3;i++) //合成总刚度矩阵

{

for(j=0;j<3;j++)

{

row=lnd[e][i];

coln=lnd[e][j];

A[row][coln]=A[row][coln]+ea[i][j];

}

}

for(i=0;i<3;i++)//合成总荷载

{

k=lnd[e][i];//确定合成总荷载时所在的行

rhs[k]=g[i]+rhs[k];

}

}

for(i=0;i<limnode;i++) //处理边界条件

{

k=limnd[i];

for(t=0;t<node;t++)

{

A[t][k]=0;

A[k][t]=0;

}

A[k][k]=1;

rhs[k]=0;

}

u=GaussElimination(A,rhs,node);//高斯消去法解方程组Au=rhs

//准备输出

for(i=1;i<m;i++)

{

t=(node-1-m)/2+i;//y=0.5时对应的节点编号

if(t%(m/8)==0)

printf("u[%d]=%f,x=%.3f, err=%.4e\n",t,u[t],xco[t], fabs(u[t]-v(xco[t],0.5)));

}

//误差估计

sum=0;

summ=0;

for(e=0;e<elem;e++)

{

i=lnd[e][0];

j=lnd[e][1];

k=lnd[e][2];

alpha[0]=(xco[j]+xco[k])/2;

alpha[1]=(xco[k]+xco[i])/2;

alpha[2]=(xco[i]+xco[j])/2;

beta[0]=(yco[j]+yco[k])/2;

beta[1]=(yco[k]+yco[i])/2;

beta[2]=(yco[i]+yco[j])/2;

graduh=d\_uh(e);

for(i=0;i<3;i++)

{

q=v(alpha[i],beta[i])-uh(e,alpha[i],beta[i]); //用于计算L2范数

sum=sum+q\*q;

x=vx(alpha[i],beta[i])-graduh[0];

y=vy(alpha[i],beta[i])-graduh[1]; //用于计算H1范数

summ=summ+x\*x+y\*y;

}

}

sum=sum\*area/3;

summ=summ\*area/3;

printf("0 norm error is:%.8f\n",sqrt(sum));

printf("1 norm error is:%.8f\n",sqrt(sum+summ));

for(e=0;e<elem;e++)

{

free(lnd[e]);

}

for(i=0;i<node;i++)

free(A[i]);

free(lnd); free(A); free(rhs);free(xco);free(yco);

free(limnd); free(u);free(w);free(graduh);

}

double v(double x, double y) //精确解

{

double z;

z=x\*y\*(1-x)\*(1-y);

return z;

}

doublevx(double x, double y)//精确解关于x的偏导数

{

double z;

z=(1-2\*x)\*(y-y\*y);

return z;

}

doublevy(doublex,double y)//精确解关于y的偏导数

{

double z;

z=(1-2\*y)\*(x-x\*x);

return z;

}

double f(doublex,double y)//方程的右端项

{

double z;

z=2\*(x+y-x\*x-y\*y);

return z;

}

double \*fun\_lambda(inte,double x, double y) //单元e上的基函数lambda0,lambda1, lambda2

{

inti,j,k;

double \*lambda;

lambda=(double \*)malloc(sizeof(double)\*3);

i=lnd[e][0]; j=lnd[e][1]; k=lnd[e][2];

lambda[0]=((xco[j]-x)\*(yco[k]-y)-(yco[j]-y)\*(xco[k]-x))/(2\*area);

lambda[1]=((xco[k]-x)\*(yco[i]-y)-(yco[k]-y)\*(xco[i]-x))/(2\*area);

lambda[2]=((xco[i]-x)\*(yco[j]-y)-(yco[i]-y)\*(xco[j]-x))/(2\*area);

return lambda;

}

double \*d\_lambda(int e)//单元e上的基函数lambda的关于x,y的偏导数，都是常数

{

inti,j,k;

double \*w;

i=lnd[e][0]; j=lnd[e][1]; k=lnd[e][2];

w=(double \*)malloc(sizeof(double)\*6);

w[0]=(yco[j]-yco[k])/(2\*area);

w[1]=(yco[k]-yco[i])/(2\*area);

w[2]=(yco[i]-yco[j])/(2\*area);

w[3]=(xco[k]-xco[j])/(2\*area);

w[4]=(xco[i]-xco[k])/(2\*area);

w[5]=(xco[j]-xco[i])/(2\*area);

return w;

}

double uh(int e, double x, double y) //单元e上的数值解uh

{

inti,k;

double z,\*lambda;

z=0;

lambda=fun\_lambda(e,x,y);

for(i=0;i<3;i++)

{

k=lnd[e][i];

z=z+u[k]\*lambda[i];

}

return z;

}

double \*d\_uh(int e)//单元e上uh关于x,y的偏导数，也都是常数

{

inti,j,k;

double \*z,\*w;

i=lnd[e][0];j=lnd[e][1];k=lnd[e][2];

w=d\_lambda(e);

z=(double \*)malloc(sizeof(double)\*2);

z[0]=z[1]=0.0;

for(i=0;i<3;i++)

{

k=lnd[e][i];

z[0]=z[0]+u[k]\*w[i];

z[1]=z[1]+u[k]\*w[i+3];

}

return z;

}

double \* GaussElimination(double \* \*a, double \*b, int N) //Gauss 消去法子程序解N阶线性方程组Ax=b

{

double \*x,sum;

inti,j,k;

x=(double \*)malloc(sizeof(double)\*N);

for(k=0;k<N;k++)

{

if(fabs(a[k][k])<1e-8)

{

printf("Error!\n");

exit;

}

b[k]=b[k]/a[k][k];

for(j=k+1;j<N;j++)

a[k][j]=a[k][j]/a[k][k];

for(i=k+1;i<N;i++)

{

b[i]=b[i]-a[i][k]\*b[k];

for(j=k+1;j<N;j++)

a[i][j]=a[i][j]-a[i][k]\*a[k][j];

}

}

x[N-1]=b[N-1];

for(i=N-2;i>=0;i--)

{

sum=0;

for(j=i+1;j<N;j++)

sum=sum+a[i][j]\*x[j];

x[i]=b[i]-sum;

}

return x;

}

Egch6\_sec5\_01.c

// parabolicfem.cpp : 定义控制台应用程序的入口点。

//本程序是用有限元法作空间离散，用C-N格式作时间离散求解抛物型方程初边值问题。

//u\_t-a1 \delta u=f(x,t), 0<x<pi,0<t<=T;

//u(x,0)=psi(x), 0<=x<=pi;

//u(0,t)=u(pi,t)=0; t>0

#include"stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

double h, tau, T, \*xco, \*\*alpha;

int m, n;

#definepi 3.14159265359

double gauss = 0.5773502692; //数值积分中的高斯点

void main()

{

inti, k;

double a1, \*t, \*a, \*b, \*c, \*d, \*ans, tmid, temp1, temp2;

double f(double x, double t);

double phi(inti, double x);

double psi(double x);

double fun1(inti, double x, double t);

double integral(double a, double b, inti, double t, double(\*fun)(int, double, double));

double exact(double x, double t);

double \*chase\_algorithm(double \*a, double \*b, double \*c, int n, double \*d);

m = 32;

n = 32;

printf("m=%d, n=%d\n", m, n);

h = pi / m; //空间步长

T = 1.0; //时间终点

tau = T / n; //时间步长

a1 = 2.0; //热的扩散率,教材中原方程中的常系数a

xco = (double \*)malloc(sizeof(double)\*(m + 1)); //位移x方向上的剖分节点坐标

for (i = 0; i<= m; i++)

xco[i] = i\*h;

t = (double \*)malloc(sizeof(double)\*(n + 1));//时间t方向上的剖分节点坐标

for (k = 0; k <= n; k++)

t[k] = k\*tau;

alpha = (double \* \*)malloc(sizeof(double \*)\*(m + 1));//设置二维数组alpha(i,k), i表示空间分量,k表示时间分量

for (i = 0; i<= m; i++)

alpha[i] = (double \*)malloc(sizeof(double)\*(n + 1));

for (i = 0; i<= m; i++)

alpha[i][0] = psi(xco[i]);//用简化的方法计算初始值

a = (double \*)malloc(sizeof(double)\*(m + 1)); //计算线性方程组的系数矩阵（三对角）

b = (double \*)malloc(sizeof(double)\*(m + 1));

c = (double \*)malloc(sizeof(double)\*(m + 1));

temp1 = h / 6.0 - a1\*tau / (2 \* h);

temp2 = 2 \* h / 3.0 + a1\*tau / h;

for (i = 0; i<= m; i++)

{

a[i] = temp1; //下次对角线上的元素

b[i] = temp2; //主对角线上的元素

c[i] = temp1;

}

b[0] = b[0] / 2.0;

b[m] = b[m] / 2.0;

d = (double \*)malloc(sizeof(double)\*(m + 1)); //计算线性方程组的右端项

for (k = 0; k<n; k++)

{

tmid = (t[k] + t[k + 1]) / 2.0; //计算向量tau\*F(tmid)

for (i = 1; i< m; i++)

{

d[i] = integral(xco[i-1],xco[i],i,tmid,fun1)+integral(xco[i],xco[i+1],i,tmid,fun1);

}

d[0] = integral(xco[0], xco[1], 0, tmid, fun1);

d[m] = integral(xco[m-1], xco[m], m, tmid, fun1);

temp1 = 2.0 \* h / 3.0 - a1\*tau / h;

temp2 = h / 6.0 + a1\*tau / (2 \* h);

for (i = 1; i<= m - 1; i++)//计算(A-a\*tau\*B/2.0)\*alpha[k]

{

d[i] = d[i] + temp2\*(alpha[i - 1][k] + alpha[i + 1][k]) + temp1\*alpha[i][k];

}

d[0] = d[0] + temp1\*alpha[0][k]/2.0+temp2\*alpha[1][k];

d[m] = d[m] + temp2\*alpha[m - 1][k] +temp1\*alpha[m][k]/2.0;

//处理零边界条件

d[0] = 0.0;d[m] = 0.0;

b[0] = 1.0;c[0] = 0.0;

a[1] = 0.0;a[m] = 0.0;

c[m - 1] = 0.0;b[m] = 1.0;

ans = chase\_algorithm(a, b, c, m + 1, d);

for (i = 0; i<= m; i++)

alpha[i][k + 1] = ans[i];

free(ans);

}

free(a); free(b); free(c); free(d);

k = m / 8;

for(i = k; i< m; i = i + k )

{

printf("alpha[%d][%d]=%f, err=%.4e\n", i, n / 2, alpha[i][n / 2], fabs(exact(xco[i], T / 2.0) - alpha[i][n / 2]));

}

for (i = 0; i<= m; i++)

free(alpha[i]);

free(xco); free(alpha); free(t);

}

double f(doublex, doublet)//右端项函数f(x,t)

{

return 0.0;

}

double phi(inti, doublex) //基函数

{

double temp, z;

temp = fabs(x - xco[i]);

if (temp <= h)

z = 1.0 - temp / h;

else

z = 0.0;

return z;

}

double psi(doublex)

{

return 5 \* sin(2 \* x) - 30 \* sin(3 \* x);

}

double fun1(inti, doublex, doublet)//算单元荷载时的被积函数

{

return f(x,t)\*phi(i, x);

}

double integral(doublea, doubleb, inti, doublet, double(\*fun)(int, double, double))//在区间[a,b]上对被积函数fun(i,x,t)进行数值积分（两点高斯公式）

{

double mid, w, ans;

mid = (b + a) / 2.0;

w = (b - a) / 2.0;

ans = w\*((\*fun)(i, mid + w\*gauss,t) + (\*fun)(i, mid - w\*gauss,t));

returnans;

}

double exact(doublex, doublet)

{

return 5 \* exp(-8.0\*t)\*sin(2 \* x) - 30 \* exp(-18.0\*t)\*sin(3 \* x);

}

double \* chase\_algorithm(double \*a, double \*b, double \*c, intn, double \*d)

{

double \* ans, \*g, \*w, p;

inti;

ans = (double \*)malloc(sizeof(double)\*n);

g = (double \*)malloc(sizeof(double)\*n);

w = (double \*)malloc(sizeof(double)\*n);

g[0] = d[0] / b[0];

w[0] = c[0] / b[0];

for (i = 1; i<n; i++)

{

p = b[i] - a[i] \* w[i - 1];

g[i] = (d[i] - a[i] \* g[i - 1]) / p;

w[i] = c[i] / p;

}

ans[n - 1] = g[n - 1];

i = n - 2;

do

{

ans[i] = g[i] - w[i] \* ans[i + 1];

i = i - 1;

} while (i>= 0);

free(g);

free(w);

returnans;

}