# 课后习题的答案（包括程序及运行结果）

[课后习题的答案（包括程序及运行结果） 1](#_Toc423678228)

[习题一 2](#_Toc423678229)

[Exch1\_01 2](#_Toc423678230)

[Exch1\_02.c 3](#_Toc423678231)

[Exch1\_03.c 4](#_Toc423678232)

[习题二 6](#_Toc423678233)

[Exch2\_01.c 6](#_Toc423678234)

[Exch2\_02.c 7](#_Toc423678235)

[Exch2\_03.c 9](#_Toc423678236)

[Exch2\_04 10](#_Toc423678237)

[Exch2\_05.c 11](#_Toc423678238)

[Exch2\_06 12](#_Toc423678239)

[Exch2\_07 12](#_Toc423678240)

[Exch2\_08 13](#_Toc423678241)

[Exch2\_09 14](#_Toc423678242)

[Exch2\_10.c 15](#_Toc423678243)

[Exch2\_11 17](#_Toc423678244)

[Exch2\_12.c 18](#_Toc423678245)

[Exch2\_13 19](#_Toc423678246)

[Exch2\_14.c 20](#_Toc423678247)

[Exch2\_15.c 22](#_Toc423678248)

[Exch2\_16.c 25](#_Toc423678249)

[Exch2\_17.c 28](#_Toc423678250)

[Exch2\_18.c 31](#_Toc423678251)

[Exch2\_19.c 32](#_Toc423678252)

[习题三 35](#_Toc423678253)

[Exch3\_01 35](#_Toc423678254)

[Exch3\_02 38](#_Toc423678255)

[Exch3\_03.c 42](#_Toc423678256)

[Exch3\_04.c 45](#_Toc423678257)

[Exch3\_05.c 46](#_Toc423678258)

[Exch3\_06.c 50](#_Toc423678259)

[Exch3\_07.c 53](#_Toc423678260)

[Exch3\_08.c 70](#_Toc423678261)

[习题四 77](#_Toc423678262)

[Exch4\_01 77](#_Toc423678263)

[Exch4\_02 79](#_Toc423678264)

[Exch4\_02\_a.c 79](#_Toc423678265)

[Exch4\_02\_b.c 86](#_Toc423678266)

[Exch4\_03 92](#_Toc423678267)

[Exch4\_03\_a.c 92](#_Toc423678268)

[Exch4\_03\_b.c 98](#_Toc423678269)

[Exch4\_04 104](#_Toc423678270)

[Exch4\_04\_a.c 105](#_Toc423678271)

[Exch4\_04\_b.c 108](#_Toc423678272)

[Exch4\_04\_c.c 112](#_Toc423678273)

[Exch4\_05 116](#_Toc423678274)

[Exch4\_06.c 117](#_Toc423678275)

[Exch4\_07.c 120](#_Toc423678276)

[Exch4\_08.c 124](#_Toc423678277)

[Exch4\_09 128](#_Toc423678278)

[Exch4\_10.c 129](#_Toc423678279)

[Exch4\_11 135](#_Toc423678280)

[Exch4\_12.c 137](#_Toc423678281)

[Exch4\_13.c 144](#_Toc423678282)

[习题五 150](#_Toc423678283)

[Exch5\_01 150](#_Toc423678284)

[Exch5\_02.c 151](#_Toc423678285)

[Exch5\_03.c 155](#_Toc423678286)

[Exch5\_04.c 159](#_Toc423678287)

[Exch5\_05 164](#_Toc423678288)

[习题六 165](#_Toc423678289)

[Exch6\_01 165](#_Toc423678290)

[Exch6\_02 165](#_Toc423678291)

[Exch6\_03 166](#_Toc423678292)

[Exch6\_04 166](#_Toc423678293)

[Exch6\_05.c 167](#_Toc423678294)

[Exch6\_06 172](#_Toc423678295)

[Exch6\_07 173](#_Toc423678296)

[Exch6\_08 174](#_Toc423678297)

[Exch6\_09.c 174](#_Toc423678298)

[Exch6\_10.c 182](#_Toc423678299)

## 习题一

### Exch1\_01

（1）一阶非线性常微分方程 （2）二阶非线性偏微分方程（3）二阶线性偏微分方程

（4）三阶非线性常微分方程 （5）四阶线性常微分方程

### Exch1\_02.c

#include "stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main( )

{

int n,i,k;

double \*x, \*y, h, forward\_difference, backward\_difference, central\_difference,exact,err;

double f(double x);

double df(double x);

double ddf(double x);

n=10;

h=1.0/n; //不要写成h=1/n，这样会导致h=0, C语言中整数除以整数的结果是取整）

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

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

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

{

x[i]=i\*h;

y[i]=exp(2\*x[i]);

}

k=2;

forward\_difference=(y[k+1]-y[k])/h;

backward\_difference=(y[k]-y[k-1])/h;

central\_difference=(y[k+1]-y[k-1])/(2\*h);

exact=df(x[k]);

printf("一阶导数的计算\n");

printf("向前差分--------------向后差分--------------中心差分\n");

printf("%f %f %f\n", forward\_difference,

backward\_difference, central\_difference);

printf("对应的各种方法的误差分别为 \n");

printf("%f %f %f\n", fabs(forward\_difference-exact), fabs

(backward\_difference-exact), fabs(central\_difference-exact));

k=4;

central\_difference=(y[k+1]-2\*y[k]+y[k-1])/(h\*h);

exact=ddf(x[k]);

printf("二阶导数的计算\n");

printf("用中心差分的计算结果是 %f, 与精确值之间的误差是%f \n", central\_difference,

fabs(central\_difference-exact));

}

double f(double x)

{

return exp(2\*x);

}

double df(double x)

{

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

}

double ddf(double x)

{

return 4\*exp(2\*x);

}

运行结果：



### Exch1\_03.c

#include "stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main( )

{

int n,i,k;

double \*x, \*y, h, forward\_difference, backward\_difference, central\_difference,exact,err;

double f(double x);

double df(double x);

double ddf(double x);

n=20;

h=1.0/n; //不要写成h=1/10，这样会导致h=0, C语言中整数除以整数的结果是取整）

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

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

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

{

x[i]=i\*h;

y[i]=exp(2\*x[i]);

}

k=4;

forward\_difference=(y[k+1]-y[k])/h;

backward\_difference=(y[k]-y[k-1])/h;

central\_difference=(y[k+1]-y[k-1])/(2\*h);

exact=df(x[k]);

printf("一阶导数的计算\n");

printf("向前差分--------------向后差分--------------中心差分\n");

printf("%f %f %f\n", forward\_difference, backward\_difference, central\_difference);

printf("对应的各种方法的误差分别为 \n");

printf("%f %f %f\n", fabs(forward\_difference-exact), fabs(backward\_difference-exact), fabs(central\_difference-exact));

k=8;

central\_difference=(y[k+1]-2\*y[k]+y[k-1])/(h\*h);

exact=ddf(x[k]);

printf("二阶导数的计算\n");

printf("用中心差分的计算结果是 %f, 与精确值之间的误差是%f \n", central\_difference, fabs(central\_difference-exact));

}

double f(double x)

{

return exp(2\*x);

}

double df(double x)

{

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

}

double ddf(double x)

{

return 4\*exp(2\*x);

}

运行结果：



结论：与上一题的结果相比较，在更细的剖分情况下计算结果更好些。

## 习题二

### Exch2\_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;

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));

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

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

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

{

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

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

}

}

double f(double x, double y) //右端项函数

{

return x+1-y;

}

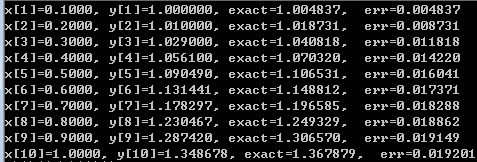
double yexact(double x) //精确解y

{

return x+exp(-x);

}

运行结果：



### Exch2\_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,epsilon, ytemp1,ytemp2,err;

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));

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

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

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("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

}

}

double f(double x, double y) //右端项函数

{

return x+1-y;

}

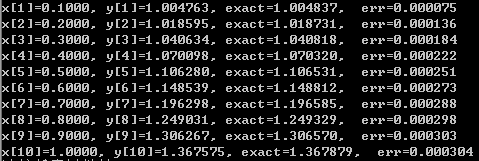
double yexact(double x) //精确解y

{

return x+exp(-x);

}

运行结果：



### Exch2\_03.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,y\_predict,err;

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

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

}

}

double f(double x, double y) //右端项函数

{

return x+1-y;

}

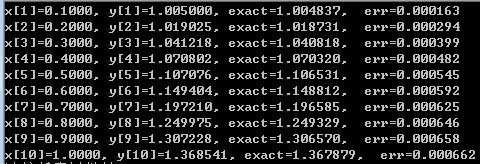
double yexact(double x) //精确解y

{

return x+exp(-x);

}

运行结果：



### Exch2\_04

解：数值格式的局部截断误差为

可见这是一个二阶方法。

### Exch2\_05.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 = 4;

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=%.8f, exact=%f, error=%.4e\n", x1, y1, yexact, err);

i++;

x0 = x1;

y0 = y1;

} while (i <= N);

}

double exact(double x)

{

double z;

z =x\*x-2\*x+2-exp(-x);

return z;

}

double f(double x, double y)

{

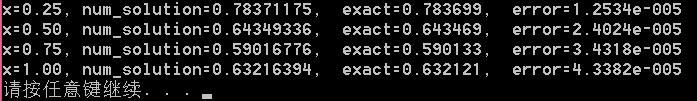
double z;

z = x\*x-y;

return z;

}

运行结果：



### Exch2\_06

解：两步数值格式 的局部截断误差为

可见这是一个二阶方法。

### Exch2\_07

解：三步数值格式的局部截断误差为

可见这是一个四阶方法。

### Exch2\_08

解：设计数值格式的局部截断误差为

要使，就必须成立



解之得唯一解：从而得四阶数值格式（2-59）。

### Exch2\_09

解：分析。四阶Adams隐式方法（2-60）在本题中即为



即

这样就成了一个显格式。程序如下：

#include "stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main()

{

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

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;

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));

y[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;

}

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

{

ytemp= y[i] + (9 \* (x[i + 1]+1) + 19 \* f(x[i], y[i]) - 5 \* f(x[i - 1], y[i - 1]) + f(x[i - 2], y[i - 2]))\*h / 24.0;

y[i + 1] = ytemp / (1.0 + 9 \* h / 24.0);

}

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

{

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

}

free(x); free(y);

}

double exact(double x)

{

double z;

z = x+exp(-x);

return z;

}

double f(double x, double y)

{

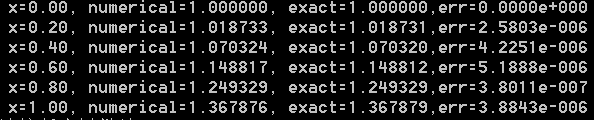
double z;

z =-y+x+1;

return z;

}

运行结果：



### Exch2\_10.c

#include "stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

void main()

{

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

int i, N;

double exact(double x);

double f(double x, double y);

a = 1.0;

b = 2.0;

N = 20;

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));

y[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;

}

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

{

y\_predict = 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; //预估

y[i + 1] = y[i] + (9 \* f(x[i + 1], y\_predict) + 19 \* f(x[i], y[i]) - 5 \* f(x[i - 1], y[i - 1]) + f(x[i - 2], y[i - 2]))\*h / 24.0; //校正

}

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

{

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

}

free(x); free(y);

}

double exact(double x)

{

double z;

z = -1.0 / x;

return z;

}

double f(double x, double y)

{

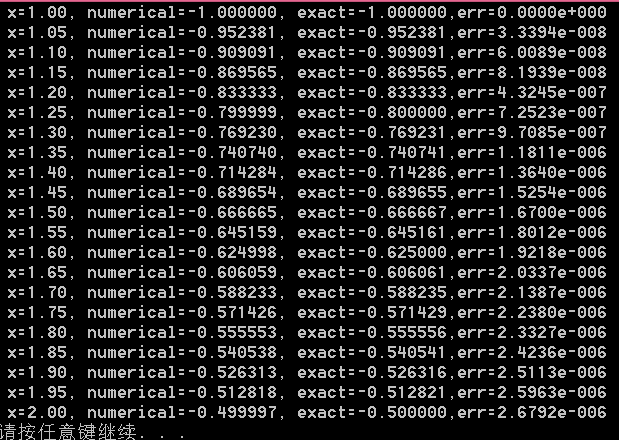
double z;

z = 1.0 / (x\*x) - y / x - y\*y;

return z;

}

运行结果：



### Exch2\_11

解：令 则原方程化为如下一阶方程组问题：



### Exch2\_12.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 = 5;

h = (b - a) / m;

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

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

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

y1 = -0.4;

y2 = -0.6;

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,c;

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

c = 0.2\*exp(2\*x);

z[0] = c\*(sin(x)-2\*cos(x));

z[1] = c\*(4\*sin(x)-3\*cos(x));

return z;

}

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

{

double \*z;

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

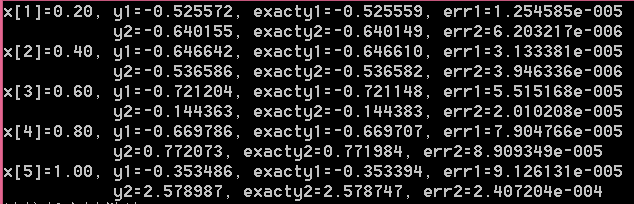
z[0] = y2;

z[1] =2\*y2-2\*y1+exp(2\*x)\*sin(x);

return z;

}

运行结果：



### Exch2\_13

证明：由于 是（2-76）的解，且分别是（2-72）、（2-73）和（2-74）的解，由等价条件（2-70）中的混合条件就有，

且，

另外还有， 和 

从而混合条件成为

整理后可得 ，

解上述关于s,t的方程可得（2-77）、（2-78）。

### Exch2\_14.c

#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 = 1.0; //alpha,beta为两点边界条件

beta = exp(1.0);

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 -4\*x;

}

double q(double x)

{

return (4\*x\*x-2);

}

double f(double x)

{

return 0.0;

}

//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;

}

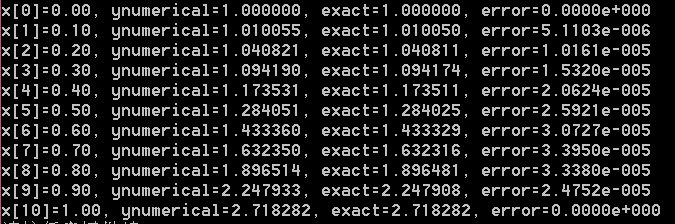
double exact(double x)

{

return exp(x\*x);

}

运行结果：



### Exch2\_15.c

#include "stdafx.h"

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

int m;

double h;

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 = 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;

lambda = 1.0;

mu = -2.0;

alpha = 1.0;

beta =0.0;

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 -4\*x;

}

double q(double x)

{

return (4\*x\*x-2);

}

double f(double x)

{

return 0;

}

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;

}

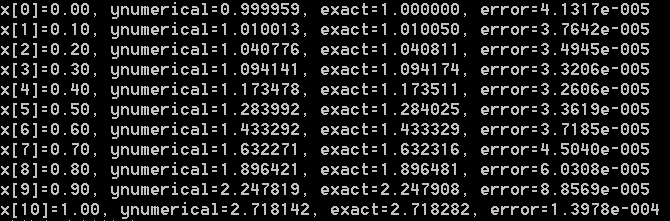
double exact(double x)

{

return exp(x\*x);

}

运行结果：



### Exch2\_16.c

#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, 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 = 5;

a = 0.0; //边界左端点

b = 1.0; //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 / 5;

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

printf("x=%.2f, ynumerical=%.10f,exact=%.10f, err=%.4e\n", x[i], y[i],exact(x[i]), fabs(exact(x[i])-y[i]));

}

double f(double x)

{

return x;

}

double q(double x)

{

return -1.0;

}

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;

} while (i >= 0);

free(g);

free(w);

return ans;

}

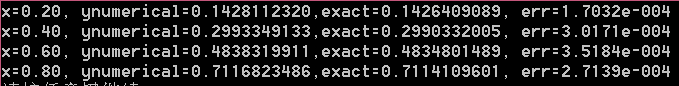
double exact(double x)

{

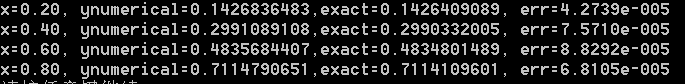
return 2\*exp(1.0)\*(exp(x)-exp(-x))/(exp(2.0)-1)-x;

}

步长为0.2时的运行结果：



将上述程序中m=5换成m=10，即步长为0.1时的运行结果：



### Exch2\_17.c

#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, 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 = 10;

h = (b - a) / m;

alpha = (exp(2.0)-4\*exp(1.0)-1)/(1-exp(2.0));

beta = 4.0 / (exp(2.0) - 1);

lambda = 1.0;

mu = -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 = 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);

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

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 x;

}

double q(double x)

{

return -1.0;

}

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;

} while (i >= 0);

free(g); free(w);

return ans;

}

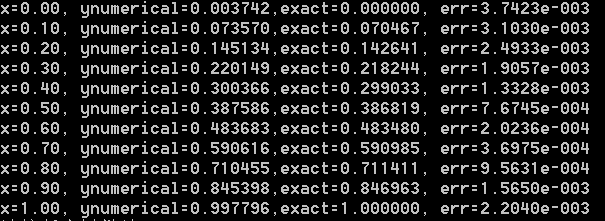
double exact(double x)

{

return 2\*exp(1.0)\*(exp(x)-exp(-x))/(exp(2.0)-1)-x;

}

运行结果：



### Exch2\_18.c

#include "stdafx.h"

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

void main()

{

double u1[4], u2[4], v[4],c,x;

int i;

u1[0] = 0.1428112320;

u1[1] = 0.2993349133;

u1[2] = 0.4838319911;

u1[3] = 0.7116823486;

u2[0] = 0.1426836483;

u2[1] = 0.2991089108;

u2[2] = 0.4835684407;

u2[3] = 0.7114790651;

for (i = 0; i < 4; i++)

{

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

x = 0.2\*(i + 1);

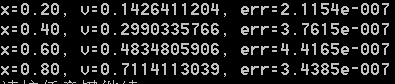
c = 2\*exp(1.0)\*(exp(x)-exp(-x))/(exp(2.0)-1)-x;

printf("x=%.2f, v=%.10f, err=%.4e\n", x, v[i], fabs(c - v[i]));

}

}

运行结果：



### Exch2\_19.c

#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, 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 = 5;

a = 0.0; //边界左端点

b = 1.0; //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;

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

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;

}

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)

{

return -1.0;

}

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;

} while (i >= 0);

free(g);

free(w);

return ans;

}

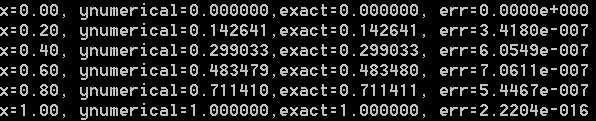
double exact(double x)

{

return 2 \* exp(1.0)\*(exp(x) - exp(-x)) / (exp(2.0) - 1) - x;

}

运行结果：



## 习题三

### Exch3\_01

解：先进行理论分析。前三种网格的网比分别是都满足稳定性条件，所以数值结果收敛于精确解。在本例中，可以计算出



所以误差主项为。这样就可以预测出以下数值结果：不变，减半时误差结果无明显变化；不变，减半，误差结果变为原来的1/4；和同时减半，误差结果变为原来的1/4. 另外，第四种网格的网比为1，不收敛，误差会以指数增长，不收敛，数值结果无效。

#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 = 4;

n = 80;

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(0.5 / h); //x=0.5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 0.0;

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return t\*sin(1.0);

}

double f(double x, double t)

{

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

}

double exact(double x, double t)

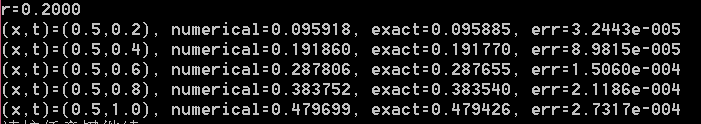
{

return t\*sin(x);

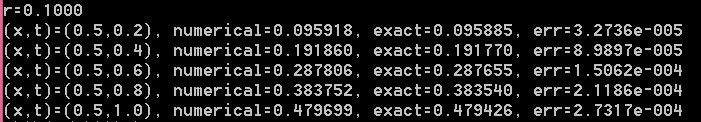
}

运行结果：

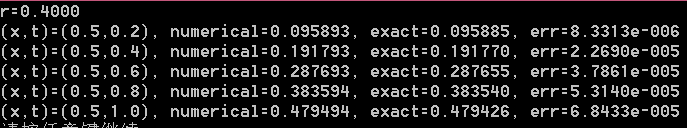
网格1：



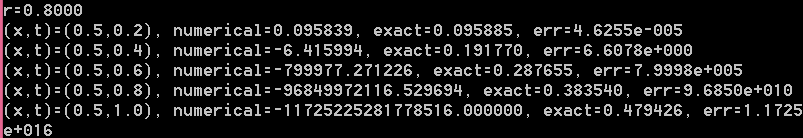
网格2：



网格3：



网格4：



从表中可见，预测结果与实际输出结果吻合。

### Exch3\_02

解：先进行理论分析。由于向后欧拉方法无条件稳定，所以三种网格步长的数值结果都收敛于精确解。在本例中，可以计算出



所以误差主项为。这样就可以预测出以下数值结果：减半，不变，误差结果变为原来的1/4； 不变，减半时误差结果无明显变化； 和同时减半，误差结果变为原来的1/4.

#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 = 4;

n = 20;

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(0.5 / h); //x=0.5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 0.0;

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return t\*sin(1.0);

}

double f(double x, double t)

{

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

}

double exact(double x, double t)

{

return t\*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);

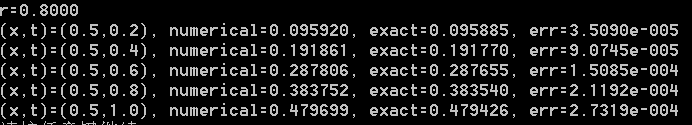
free(w);

return ans;

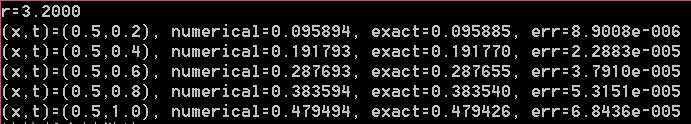
}

运行结果：

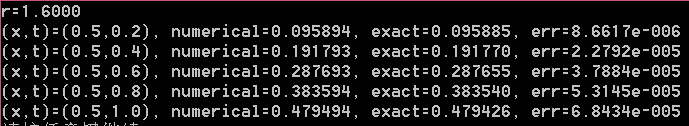
网格1：



网格2：



网格3：



从表中可见，预测结果与实际输出结果吻合。

### Exch3\_03.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, 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 = 10;

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(0.5 / h); //x=0.5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=%.10f, exact=%.10f, 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 0.0;

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return t\*sin(1.0);

}

double f(double x, double t)

{

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

}

double exact(double x, double t)

{

return t\*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);

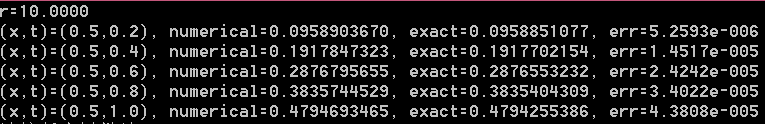
free(w);

return ans;

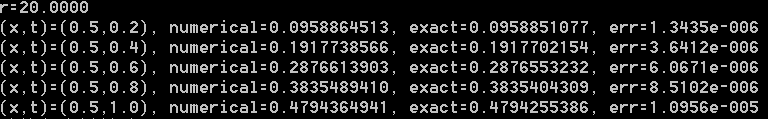
}

运行结果：

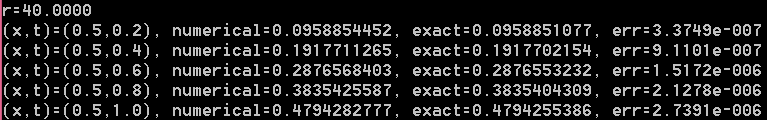
m=n=10



m=n=20



m=n=40



### Exch3\_04.c

#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] = 0.0958903670;

u1[1] = 0.1917847323;

u1[2] = 0.2876795655;

u1[3] = 0.3835744529;

u1[4] = 0.4794693465;

u2[0] = 0.0958864513;

u2[1] = 0.1917738566;

u2[2] = 0.2876613903;

u2[3] = 0.3835489410;

u2[4] = 0.4794364941;

u3[0] = 0.0958854452;

u3[1] = 0.1917711265;

u3[2] = 0.2876568403;

u3[3] = 0.3835425587;

u3[4] = 0.4794282777;

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

{

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

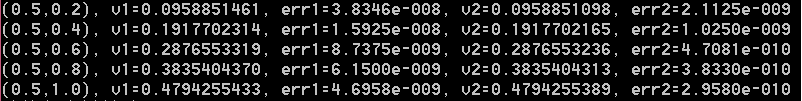
v2[i] = (4 \* u3[i] - u2[i]) / 3.0;

c =sin(0.5)\*0.2\*(i+1);

printf("(%.1f,%.1f), v1=%.10f, err1=%.4e, v2=%.10f, err2=%.4e\n", 0.5, 0.2\*(i + 1), v1[i], fabs(c - v1[i]), v2[i], fabs(c - v2[i])); }

}

运行结果：



### Exch3\_05.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 = 10;

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(0.5 / h);

j = int(0.2 / tau);

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

{

printf("(x,t)=(0.5,%.1f), numerical=%f, error=%.4e\n", t[k], u[xnumber][k], fabs(u[xnumber][k] - exact(0.5, 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 0.0;

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return t\*sin(1.0);

}

double f(double x, double t)

{

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

}

double exact(double x, double t)

{

return t\*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);

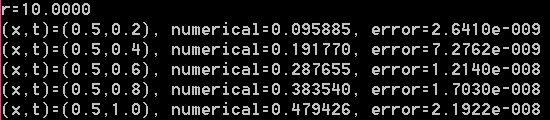
free(w);

return ans;

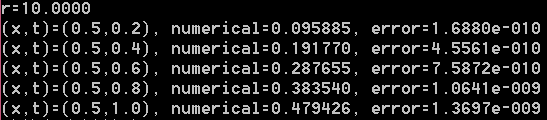
}

运行结果：

m=n=10



m=20,n=40



### Exch3\_06.c

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

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

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);

double exact(double x, double t);

m = 10;

n = 40;

h = 1.0 / m;

tau = 1.0 / n;

a = 1.0;

lambda = 0.0;

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);

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); free(ans);

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

free(u[i]);

free(u);

}

double f(double x, double t)

{

double z;

z = x\*x;

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

}

double phi(double x)

{

return pow(x,4.0);

}

double alpha(double t)

{

return 0.0;

}

double beta(double t)

{

return 5.0\*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;

}

double exact(double x, double t)

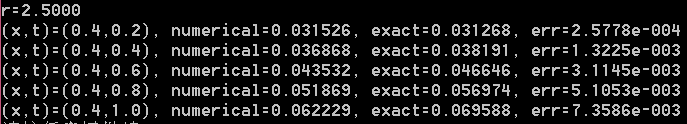
{

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

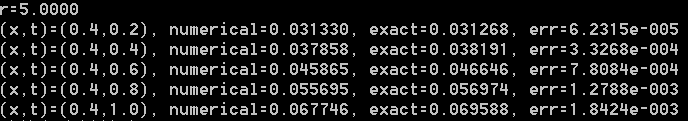
}

运行结果：

m = 10, n = 40;



m = 20, n = 80;



从上面两个数值结果看出，当时间、空间步长同时减半后，误差降为原来误差的1/4，从而

原数值格式是二阶的。

### Exch3\_07.c

**1. 采用Peaceman-Rachford格式的程序**

// 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 = 2.0;

m = 80;

n = 120;

L = 160;

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 / 5;//用于确定y方向每隔多少个点打印结果

k = 3 \* L / 5;//用于确定t=1.2时的时间层

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)

{

return 2\*(t-1)\* exp(t\*t+x + y);

}

double phi(double x, double y)

{

return exp(x + y);

}

double g1(double y, double t)

{

return exp(t\*t+y);

}

double g2(double y, double t)

{

return exp(t\*t + y+1);

}

double g3(double x, double t)

{

return exp(t\*t + x);

}

double g4(double x, double t)

{

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

}

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;

}

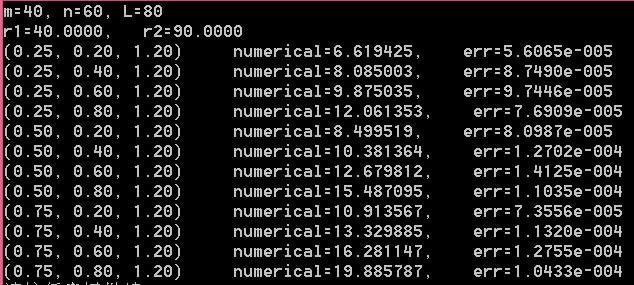
double exact(double x, double y, double t)

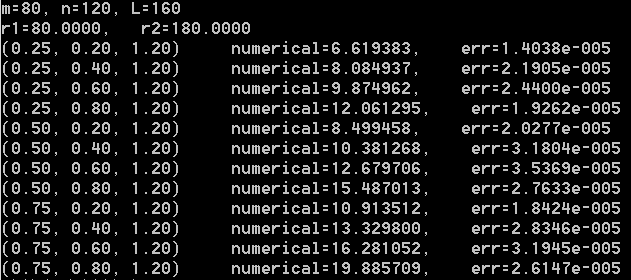
{

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

}

程序运行结果：





**2. 采用D’Yakonov格式的程序**

// 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 = 2.0;

m = 40;

n = 60;

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

}

}

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 / 5;//用于确定y方向每隔多少个点打印结果

k = 3 \* L / 5;//用于确定t=1.2时的时间层

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)

{

return 2 \* (t - 1)\* exp(t\*t + x + y);

}

double phi(double x, double y)

{

return exp(x + y);

}

double g1(double y, double t)

{

return exp(t\*t + y);

}

double g2(double y, double t)

{

return exp(t\*t + y + 1);

}

double g3(double x, double t)

{

return exp(t\*t + x);

}

double g4(double x, double t)

{

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

}

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(t\*t + y+x);;

}

程序运行结果与Peaceman-Rachford格式的结果完全相同。

**3. 采用Douglas格式的程序**

// 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 = 2.0;

m = 80;

n =120;

L = 160;

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 / 5;//用于确定y方向每隔多少个点打印结果

k = 3 \* L / 5;//用于确定t=1.2时的时间层

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)

{

return 2 \* (t - 1)\* exp(t\*t + x + y);

}

double phi(double x, double y)

{

return exp(x + y);

}

double g1(double y, double t)

{

return exp(t\*t + y);

}

double g2(double y, double t)

{

return exp(t\*t + y + 1);

}

double g3(double x, double t)

{

return exp(t\*t + x);

}

double g4(double x, double t)

{

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

}

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 +t\*t);

}

程序运行结果与Peaceman-Rachford格式的结果完全相同。

### Exch3\_08.c

//选用紧ADI－ D'Yakonov方法，关于时间二阶，关于空间四阶。

//求解二维抛物型方程初边值问题//紧差分之－交替方向 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, const11, const12, const21, const22;

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 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 = 2.0;

m = 80;

n = 120;

L = 320;

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++)

{

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;

temp = temp+epsilon\_x\_epsilon\_y(u, i, j, k)+r2\*epsilon\_x\_delta\_y(u,i,j,k)/2.0+r1\*epsilon\_y\_delta\_x(u,i,j,k)/2.0+r1\*r2\*delta\_x\_delta\_y(u,i,j,k)/4.0;

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

gap\_i = m / 4;//用于确定x方向每隔多少个点打印结果

gap\_j = n / 5;//用于确定y方向每隔多少个点打印结果

k = 3 \* L / 5;//用于确定t=1.2时的时间层

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)

{

return 2 \* (t - 1)\* exp(t\*t + x + y);

}

double phi(double x, double y)

{

return exp(x + y);

}

double g1(double y, double t)

{

return exp(t\*t + y);

}

double g2(double y, double t)

{

return exp(t\*t + y + 1);

}

double g3(double x, double t)

{

return exp(t\*t + x);

}

double g4(double x, double t)

{

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

}

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;

}

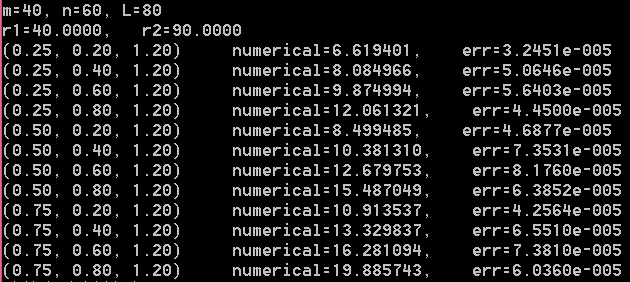
double exact(double x, double y, double t)

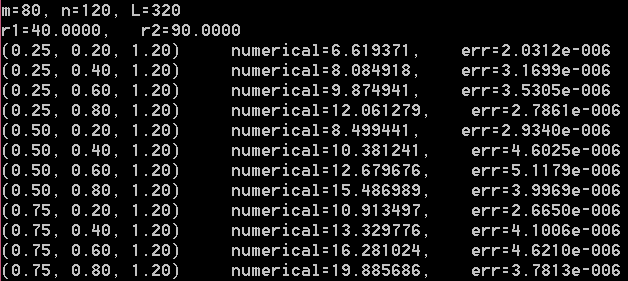
{

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

}

程序运行结果：

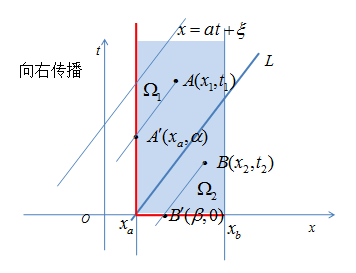




## 习题四

### Exch4\_01

1. 证：原方程也有一族特征线，其方程为（为任意常数），且沿着特征线，函数为常值。在这些特征线中有一条临界线，如下图所示，这条特征线将求解区域分成两部分和。由于原方程表示的波函数是向右传播的，所以在内的点处的函数的值依赖于左边界条件，而在内的点处的函数的值则依赖于初始条件。为了得到整个求解区域内任意一点处的精确解，分三步走。



首先，在平面内容易得出临界线的方程为：



其次，考察内的任意一点处的函数值。注意到过点的特征线方程为，且易知沿着这条特征线，点在左边界上的投影点为，也就是这条特征线与左边界的交点，易知。于是，点处的值就是点处的值， 即有。再由点的任意性知，内的任意一点处的值为



最后，再考察内的任意一点处的函数值。注意到过点的特征线方程为，且易知沿着这条特征线，点在轴上的投影点为，也就是这条特征线与轴的交点，易知。于是，点处的值就是点处的值， 即有。再由点的任意性知，内的任意一点处的值为



综上，原一阶双曲型方程初边值问题的精确解为

证毕。

### Exch4\_02

由于方程表示的是一个向左传播的波的运动，所以根据教材中分析可知，原方程的精确解为



### Exch4\_02\_a.c

（迎风显格式）

// 这是用迎风显格式求解对流方程，求解区间为x in [0,1], t in [0,1]

//最后打印t=0.04, 0.2, 0.8时，x in [0,1]的数值解

#include"stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

double pi=3.14159265359;

void main()

{

doublexa, xb, a, h, ta,tb, tau, r, \*x, \*t, \* \*u;

int j, k, m, n;

double phi(double x);

double psi(double t);

FILE \*fp;

xa=0.0; //空间区域为[xa, xb]

xb=1.0;

m =20;

h = (xb-xa) / m; //空间步长

ta=0.0;

tb=1.0; //设时间区域为[0,1]

n=100;

tau = (tb-ta)/n; //时间步长

a = -2.0;

r = a\*tau / h; //参数r

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

if (fabs(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] = xa+j\*h; //x方向的节点坐标

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

t[k] = ta+k\*tau; //t方向的节点坐标

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]); //初始条件

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

u[m][k]=psi(t[k]); //波向左传播，右边界条件

for (k = 0; k < n; k++) //迎风格式

{

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

u[j][k + 1] =(1.0 + r)\*u[j][k] -r\*u[j + 1][k];

}

if (fopen\_s(&fp, "upwind\_t1.dat", "w")!= 0)//准备将t=0.04时刻的数据写入到文件upwind\_t1.dat中

{

printf("cannot open the file!\n");

return;

}

else

{

k=4; //确定出t=0.04时刻的时间层

printf("Data at time t=0.04\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

printf("\n\n");

fclose(fp);

}

if (fopen\_s(&fp, "upwind\_t2.dat", "w")!= 0)

{

printf("cannot open the file!\n");

return;

}

else

{

k=20; //确定出t=0.2时刻的时间层

printf("Data at time t=0.2\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

printf("\n\n");

fclose(fp);

}

if (fopen\_s(&fp, "upwind\_t3.dat", "w")!= 0)

{

printf("cannot open the file!\n");

return;

}

else

{

k=80; //确定出t=0.8时刻的时间层

printf("Data at time t=0.8\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

fclose(fp);

}

//程序运行完后将当前目录下的dat文件拷贝到Matlab的工作目录下可以直接画图。这样用matlab可以直接读取数据文件从而画出直观图来。

free(x); free(t);

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

free(u[j]);

free(u);

}

double phi(double x)

{

return 1.0+sin(2\*pi\*x);

}

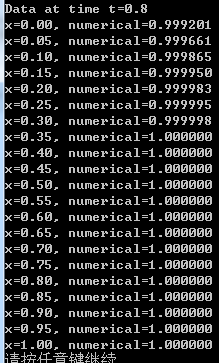
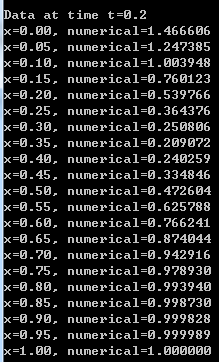
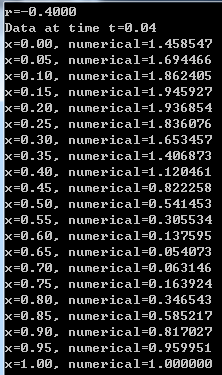
double psi(double t)

{

return 1.0;

}

程序运行后得到显示在屏幕上的结果为：



然后将当前目录下的三个后缀为dat的文件拷贝到Matlab的bin目录下，接着，打开Matlab程序，新建一个脚本文件，文件内容如下：

clear;

loadupwind\_t1.dat

x=upwind\_t1(:,1);

u=upwind\_t1(:,2);

axis([0 1 -0.2 2.2]);

holdon;

t=0.04;

xx=0:0.01:1;

fori=1:101

if(t>(1-xx(i))/2)

y(i)=1.0;

else

y(i)=1.0+sin(2\*pi\*(xx(i)+2\*t));

end

end

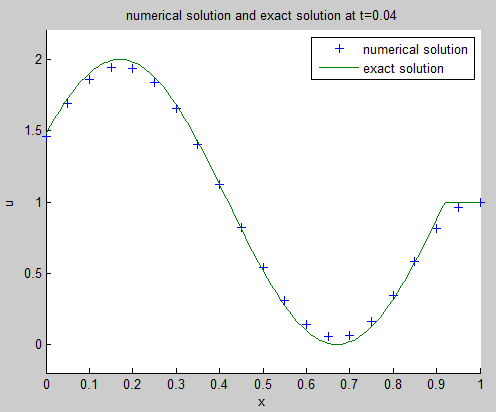
plot(x,u,'+', xx,y);

xlabel('x');

ylabel('u');

title('numerical solution and exact solution at t=0.04');

运行该Matlab程序就可以得到t=0.04时的数值解与精确解的图像了。



再修改程序如下：

clear;

loadupwind\_t2.dat

x=upwind\_t2(:,1);

u=upwind\_t2(:,2);

axis([0 1 -0.2 2.2]);

holdon;

t=0.2;

xx=0:0.01:1;

fori=1:101

if(t>(1-xx(i))/2)

y(i)=1.0;

else

y(i)=1.0+sin(2\*pi\*(xx(i)+2\*t));

end

end

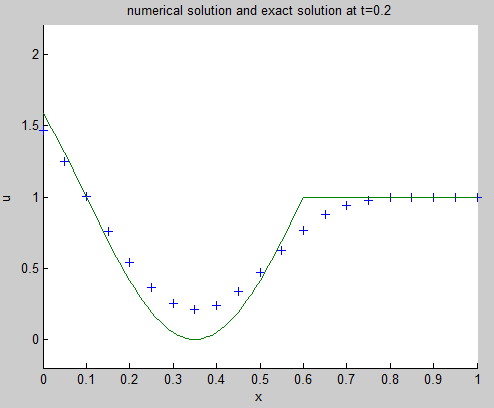
plot(x,u,'+', xx,y);

xlabel('x');

ylabel('u');

title('numerical solution and exact solution at t=0.2');

运行该Matlab程序就可以得到t=0.2时的数值解与精确解的图像了。



再修改程序如下：

clear;

loadupwind\_t3.dat

x=upwind\_t3(:,1);

u=upwind\_t3(:,2);

axis([0 1 -0.2 2.2]);

holdon;

t=0.8;

xx=0:0.01:1;

fori=1:101

if(t>(1-xx(i))/2)

y(i)=1.0;

else

y(i)=1.0+sin(2\*pi\*(xx(i)+2\*t));

end

end

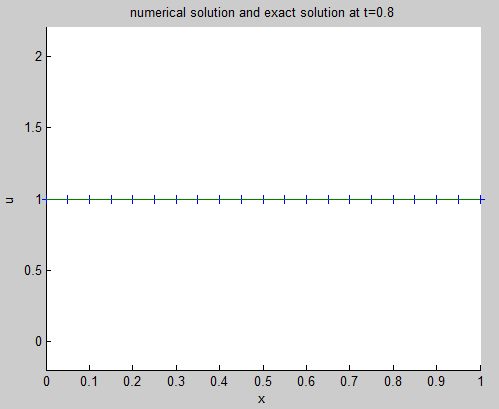
plot(x,u,'+', xx,y);

xlabel('x');

ylabel('u');

title('numerical solution and exact solution at t=0.8');

运行该Matlab程序就可以得到t=0.8时的数值解与精确解的图像了。



### Exch4\_02\_b.c

迎风隐格式为：（将关于时间的向前差商改成向后差商即可从显格式变成隐格式）



此格式可以显化为，j从m-1开始倒着算到j=0。

// 这是用迎风隐格式求解对流方程，求解区间为x in [0,1], t in [0,1]

//最后打印t=0.04, 0.2, 0.8时，x in [0,1]的数值解

#include"stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

double pi=3.14159265359;

void main()

{

doublexa, xb, a, h, ta,tb, tau, r, \*x, \*t, \* \*u, t\_time;

int j, k, m, n, tk;

double phi(double x);

double psi(double t);

FILE \*fp;

xa=0.0; //空间区域为[xa, xb]

xb=1.0;

m =20;

h = (xb-xa) / m; //空间步长

ta=0.0;

tb=1.0; //设时间区域为[0,1]

n=100;

tau = (tb-ta)/n; //时间步长

a = -2.0;

r = a\*tau / h; //参数r

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

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

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

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

x[j] = xa+j\*h; //x方向的节点坐标

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

t[k] = ta+k\*tau; //t方向的节点坐标

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]); //初始条件

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

u[m][k]=psi(t[k]); //波向左传播，右边界条件

for (k = 1; k <= n; k++) //迎风格式

{

for (j = m-1; j >=0; j--)

u[j][k] =(u[j][k-1] -r\*u[j + 1][k])/(1-r);

}

if (fopen\_s(&fp, "upwind\_t1\_b.dat", "w")!= 0)//准备将t=0.04时刻的数据写入到文件upwind\_t1.dat中

{

printf("cannot open the file!\n");

return;

}

else

{

k=4; //确定出t=0.04时刻的时间层

printf("Data at time t=0.04\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

printf("\n\n");

fclose(fp);

}

if (fopen\_s(&fp, "upwind\_t2\_b.dat", "w")!= 0)

{

printf("cannot open the file!\n");

return;

}

else

{

k=20; //确定出t=0.2时刻的时间层

printf("Data at time t=0.2\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

printf("\n\n");

fclose(fp);

}

if (fopen\_s(&fp, "upwind\_t3\_b.dat", "w")!= 0)

{

printf("cannot open the file!\n");

return;

}

else

{

k=80; //确定出t=0.8时刻的时间层

printf("Data at time t=0.8\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

fclose(fp);

}

//程序运行完后将当前目录下的dat文件拷贝到Matlab的工作目录下可以直接画图。这样用matlab可以直接读取数据文件从而画出直观图来。

free(x); free(t);

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

free(u[j]);

free(u);

}

double phi(double x)

{

return 1.0+sin(2\*pi\*x);

}

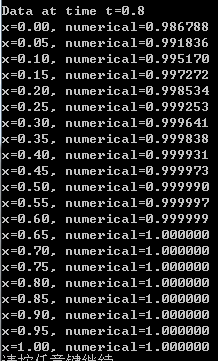
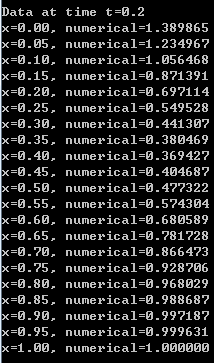
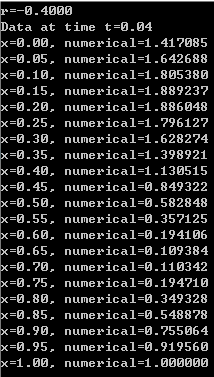
double psi(double t)

{

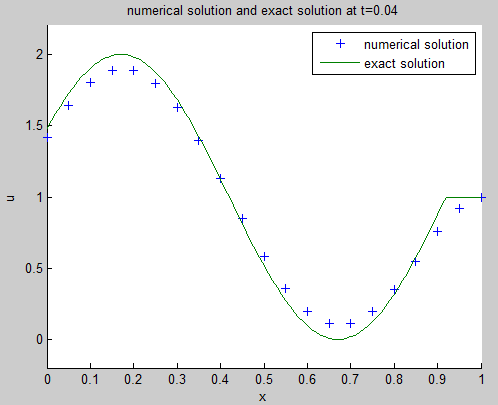
return 1.0;

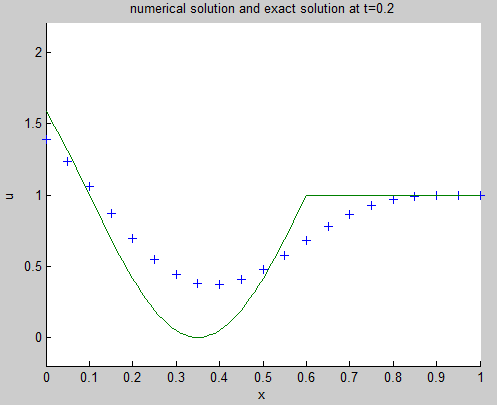
}

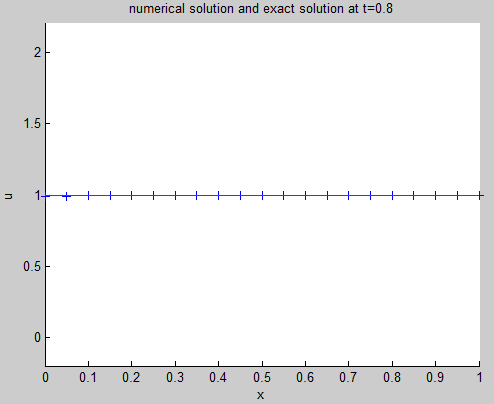
程序运行后得到显示在屏幕上的结果为：



然后将当前目录下的三个数据文件（upwind\_t1\_b.dat， upwind\_t2\_b.dat， upwind\_t3\_b.dat）拷贝到Matlab的bin目录下，接着仿照前面显格式的情况新建一个脚本文件，就可以分别得到三个时间层上的数值解与精确解的图像了，如下所示：







### Exch4\_03

由于方程表示的是一个向右传播的波的运动，所以习题1分析可知，原方程的精确解为



### Exch4\_03\_a.c

迎风显格式：

// 这是用迎风显格式求解对流方程，求解区间为x in [0,1], t in [0,1]

//最后打印t=0.04, 0.2, 0.8时，x in [0,1]的数值解

#include"stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

double pi=3.14159265359;

void main()

{

doublexa, xb, a, h, ta,tb, tau, r, \*x, \*t, \* \*u;

int j, k, m, n;

double phi(double x);

double psi(double t);

FILE \*fp;

xa=0.0; //空间区域为[xa, xb]

xb=1.0;

m =20;

h = (xb-xa) / m; //空间步长

ta=0.0;

tb=1.0; //设时间区域为[0,1]

n=100;

tau = (tb-ta)/n; //时间步长

a = 2.0;

r = a\*tau / h; //参数r

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

if (fabs(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] = xa+j\*h; //x方向的节点坐标

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

t[k] = ta+k\*tau; //t方向的节点坐标

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]); //初始条件

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

u[0][k]=psi(t[k]); //波向右传播，左边界条件

for (k = 0; k < n; k++) //迎风格式

{

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

u[j][k+1] =r\*u[j-1][k]+(1-r)\*u[j][k];

}

if (fopen\_s(&fp, "data\_t1\_a.dat", "w")!= 0)//准备将t=0.04时刻的数据写入到文件data\_t1\_a.dat中,后面类似

{

printf("cannot open the file!\n");

return;

}

else

{

k=4; //确定出t=0.04时刻的时间层

printf("Data at time t=0.04\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

printf("\n\n");

fclose(fp);

}

if (fopen\_s(&fp, "data\_t2\_a.dat", "w")!= 0)

{

printf("cannot open the file!\n");

return;

}

else

{

k=20; //确定出t=0.2时刻的时间层

printf("Data at time t=0.2\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

printf("\n\n");

fclose(fp);

}

if (fopen\_s(&fp, "data\_t3\_a.dat", "w")!= 0)

{

printf("cannot open the file!\n");

return;

}

else

{

k=80; //确定出t=0.8时刻的时间层

printf("Data at time t=0.8\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

fclose(fp);

}

//程序运行完后将当前目录下的dat文件拷贝到Matlab的工作目录下可以直接画图。这样用matlab可以直接读取数据文件从而画出直观图来。

free(x);free(t);

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

free(u[j]);

free(u);

}

double phi(double x)

{

return 1.0+sin(2\*pi\*x);

}

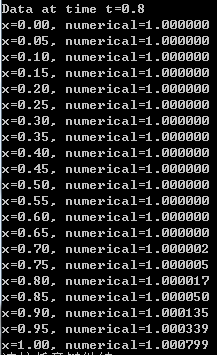
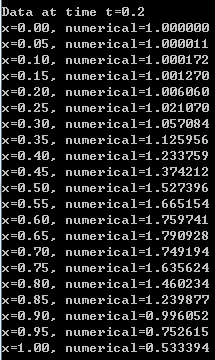
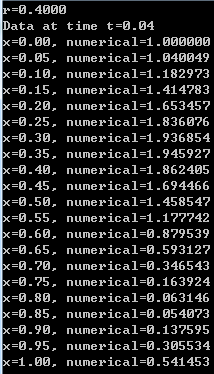
double psi(double t)

{

return 1.0;

}

程序运行结果（屏幕显示）：



然后将当前目录下的三个数据文件（data\_t1\_a.dat， data\_t2\_a.dat， data\_t3\_a.dat）拷贝到Matlab的bin目录下，接着新建一个脚本文件(如下所示), 依次修改相应的内容，就可以得到三个时间层上的数值解与精确解的图像了。

clear;

loaddata\_t1\_a.dat

x=data\_t1\_a(:,1);

u=data\_t1\_a(:,2);

axis([0 1 -0.2 2.2]);

holdon;

t=0.04;

xx=0:0.01:1;

fori=1:101

if(t>xx(i)/2)

y(i)=1.0;

else

y(i)=1.0+sin(2\*pi\*(xx(i)-2\*t));

end

end

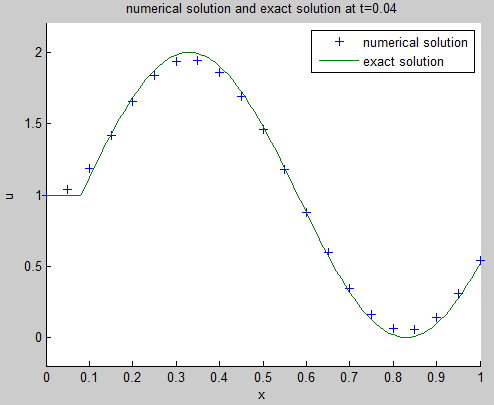
plot(x,u,'+', xx,y);

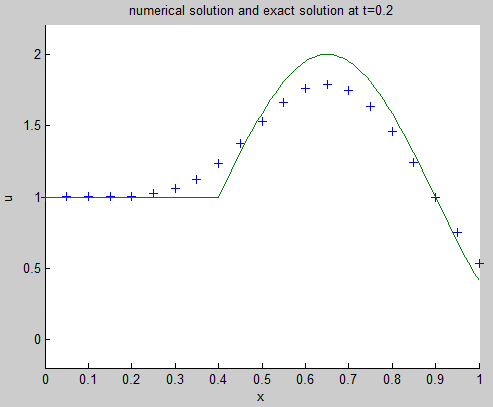
xlabel('x');

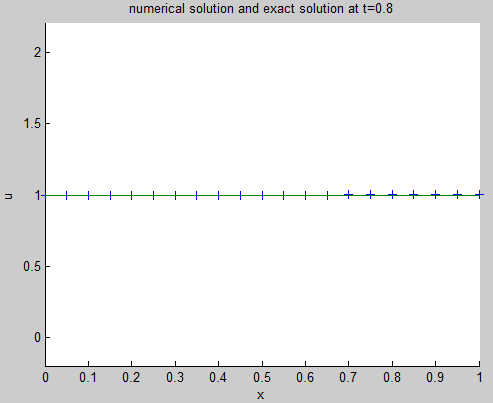
ylabel('u');

title('numerical solution and exact solution at t=0.04');

运行此脚本文件可得以下结果：







### Exch4\_03\_b.c

迎风隐格式：

此格式可以显化为，j从1开始倒着算到j=m。

// 这是用迎风隐格式求解对流方程，求解区间为x in [0,1], t in [0,1]

//最后打印t=0.04, 0.2, 0.8时，x in [0,1]的数值解

#include"stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

double pi=3.14159265359;

void main()

{

doublexa, xb, a, h, ta,tb, tau, r, \*x, \*t, \* \*u;

int j, k, m, n;

double phi(double x);

double psi(double t);

FILE \*fp;

xa=0.0; //空间区域为[xa, xb]

xb=1.0;

m =20;

h = (xb-xa) / m; //空间步长

ta=0.0;

tb=1.0; //设时间区域为[0,1]

n=100;

tau = (tb-ta)/n; //时间步长

a = 2.0;

r = a\*tau / h; //参数r

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

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

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

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

x[j] = xa+j\*h; //x方向的节点坐标

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

t[k] = ta+k\*tau; //t方向的节点坐标

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]); //初始条件

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

u[0][k]=psi(t[k]); //波向右传播，左边界条件

for (k = 1; k <= n; k++) //迎风格式

{

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

u[j][k] =(u[j][k-1]+r\*u[j-1][k])/(1+r);

}

if (fopen\_s(&fp, "data\_t1\_b.dat", "w")!= 0)//准备将t=0.04时刻的数据写入到文件data\_t1\_a.dat中,后面类似

{

printf("cannot open the file!\n");

return;

}

else

{

k=4; //确定出t=0.04时刻的时间层

printf("Data at time t=0.04\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

printf("\n\n");

fclose(fp);

}

if (fopen\_s(&fp, "data\_t2\_b.dat", "w")!= 0)

{

printf("cannot open the file!\n");

return;

}

else

{

k=20; //确定出t=0.2时刻的时间层

printf("Data at time t=0.2\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

printf("\n\n");

fclose(fp);

}

if (fopen\_s(&fp, "data\_t3\_b.dat", "w")!= 0)

{

printf("cannot open the file!\n");

return;

}

else

{

k=80; //确定出t=0.8时刻的时间层

printf("Data at time t=0.8\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][k]);//屏幕输出，也可以不用输出。

fprintf(fp, "%f, %f\n", x[j], u[j][k]); //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。

}

fclose(fp);

}

//程序运行完后将当前目录下的dat文件拷贝到Matlab的工作目录下可以直接画图。这样用matlab可以直接读取数据文件从而画出直观图来。

free(x);free(t);

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

free(u[j]);

free(u);

}

double phi(double x)

{

return 1.0+sin(2\*pi\*x);

}

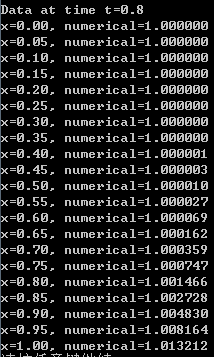
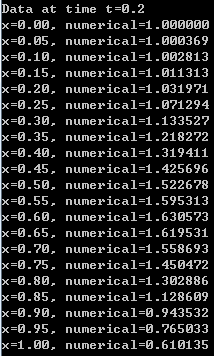
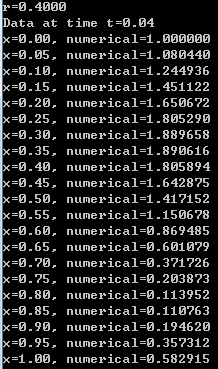
double psi(double t)

{

return 1.0;

}

程序运行结果（屏幕显示）：



然后将当前目录下的三个数据文件（data\_t1\_b.dat， data\_t2\_b.dat， data\_t3\_b.dat）拷贝到Matlab的bin目录下，接着新建一个脚本文件(如下所示), 依次修改相应的内容，就可以得到三个时间层上的数值解与精确解的图像了。

clear;

loaddata\_t1\_b.dat

x=data\_t1\_b(:,1);

u=data\_t1\_b(:,2);

axis([0 1 -0.2 2.2]);

holdon;

t=0.04;

xx=0:0.01:1;

fori=1:101

if(t>xx(i)/2)

y(i)=1.0;

else

y(i)=1.0+sin(2\*pi\*(xx(i)-2\*t));

end

end

plot(x,u,'+', xx,y);

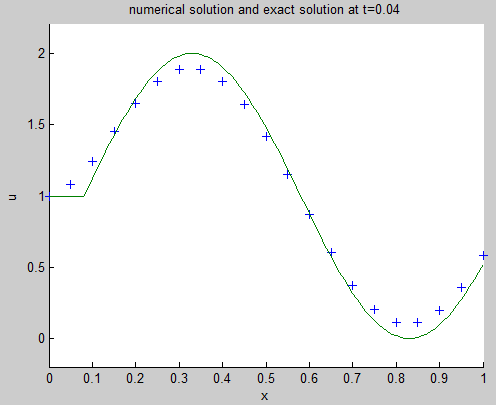
xlabel('x');

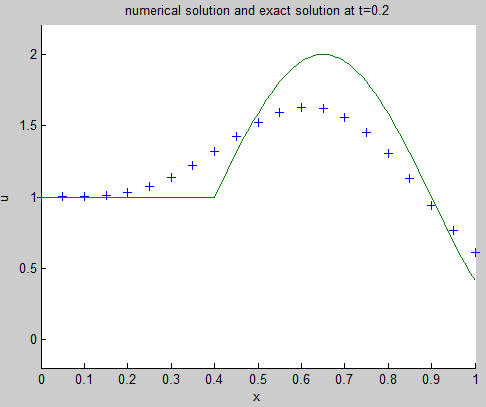
ylabel('u');

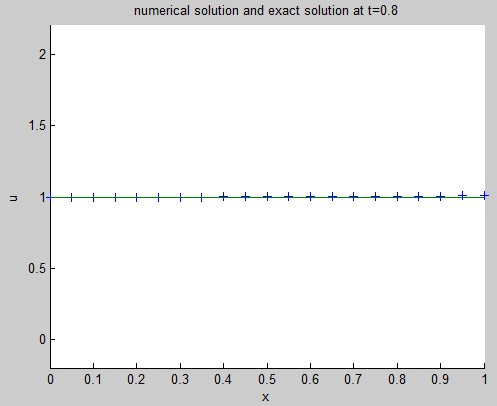
title('numerical solution and exact solution at t=0.04');

运行此脚本文件可得以下结果：

数值解用“+”，精确解是实线。







### Exch4\_04

首先证明精确解。证明：仿教材中分析，原对流方程有一族特征线（其中为任意常数），在这族特征线中有一条临界线：，这临界线将原求解区域分成上无界部分和下直角三角形区域。对于在内的任意一点， 则，其中为点沿着过自身的特征线投到左边界上的投影点。 再由边界条件知，左边界点处的值又等于与之等高的右边界点处的值，即。 最后，其中为点沿着过自身的特征线投到*x*轴上的投影点。这样就有，从而在内的任意一点处的值为。另外，在内的点的值直接可以借特征线投影到*x*轴上，立即得到内的任意一点处的值为。证毕。

以下为三种格式计算数值解并与精确解进行比较。

### Exch4\_04\_a.c

Lax-Friedrichs格式

// Exch4\_04\_a.cpp : 定义控制台应用程序的入口点。

// 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=0.0, t=0.05 和t=1.0时，x in [-1,2]的数值解

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

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

int j, k, m, n;

double phi(double x);

double exact(double x, double t);

m = 20;

n = 100;

xa = -1.0;

xb = 2.0;

h = (xb-xa)/ 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] = xa + 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 = 1; j <= m - 1; j++)

u[j][k + 1] = (1.0 + r)\*u[j - 1][k] / 2.0 + (1.0 - r)\*u[j + 1][k] / 2;

u[m][k + 1] = r\*u[m - 1][k] + (1.0 - r)\*u[m][k];//注意此处需要补充右边界信息，为此用了迎风格式计算右边界的数值解

u[0][k + 1] = u[m][k + 1];//然后根据边界条件得到左边界数值解

}

printf("at time t=0.0\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][0]);//屏幕输出，由于是初始状态无误差，故不必输出误差

}

printf("\n");

printf("at time t=0.05\n");

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

{

printf("x=%.2f, numerical=%f, err=%.4e\n", x[j], u[j][5], fabs(u[j][5] - exact(x[j], t[5])));

}

printf("\n");

printf("at time t=1.0\n");

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

{

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

}

free(x); free(t);

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

free(u[j]);

free(u);

}

double phi(double x)

{

if ((x < 0) || (x>1))

return 0.0;

else

return 1.0;

}

double exact(double x, double t)

{

if (t <= x+1)

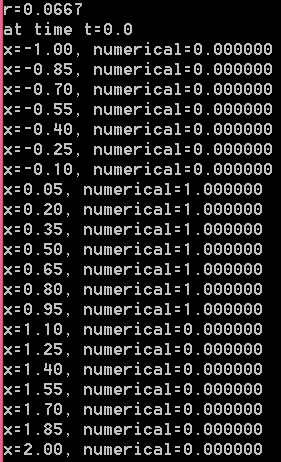
return phi(x-t);

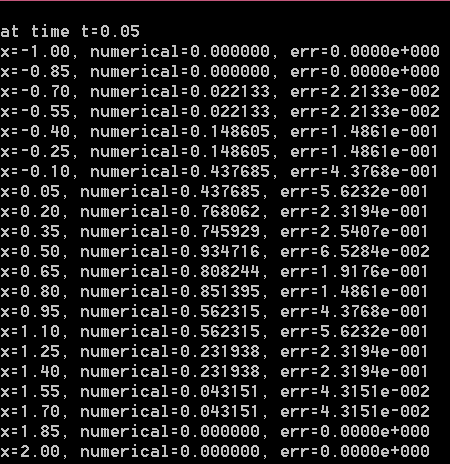
else

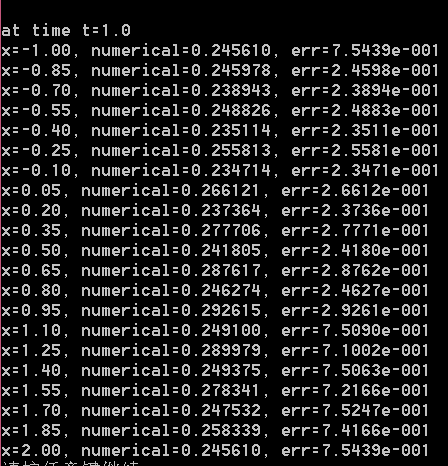
return phi(3+x-t);

}

程序运行结果：







### Exch4\_04\_b.c

Lax-Wendroff格式

// Exch4\_04\_b.cpp : 定义控制台应用程序的入口点。

// 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=0.0, t=0.05 和t=1.0时，x in [-1,2]的数值解

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

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

int j, k, m, n;

double phi(double x);

double exact(double x, double t);

m = 20;

n = 100;

xa = -1.0;

xb = 2.0;

h = (xb - xa) / 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] = xa + 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 = 1; j <= m - 1; j++)

u[j][k + 1] = r\* (r+1.0)\*u[j - 1][k] / 2.0 + (1.0-r\*r)\*u[j][k]+r\*(r-1.0)\*u[j + 1][k] / 2;

u[m][k + 1] = r\*u[m - 1][k] + (1.0 - r)\*u[m][k];//注意此处需要补充右边界信息，为此用了迎风格式计算右边界的数值解

u[0][k + 1] = u[m][k + 1];//然后根据边界条件得到左边界数值解

}

printf("at time t=0.0\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][0]);//屏幕输出，由于是初始状态无误差，故不必输出误差

}

printf("\n");

printf("at time t=0.05\n");

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

{

printf("x=%.2f, numerical=%f, err=%.4e\n", x[j], u[j][5], fabs(u[j][5] - exact(x[j], t[5])));

}

printf("\n");

printf("at time t=1.0\n");

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

{

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

}

free(x); free(t);

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

free(u[j]);

free(u);

}

double phi(double x)

{

if ((x < 0) || (x>1))

return 0.0;

else

return 1.0;

}

double exact(double x, double t)

{

if (t <= x + 1)

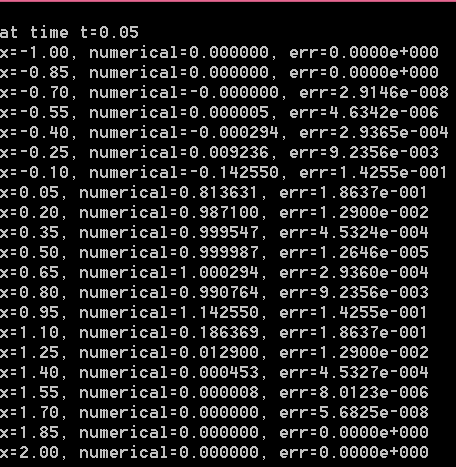
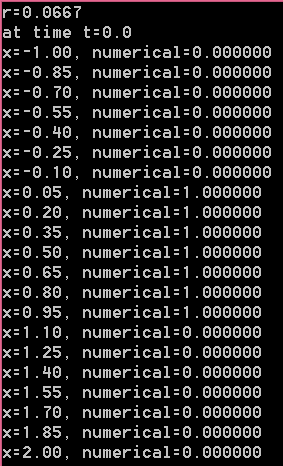
return phi(x - t);

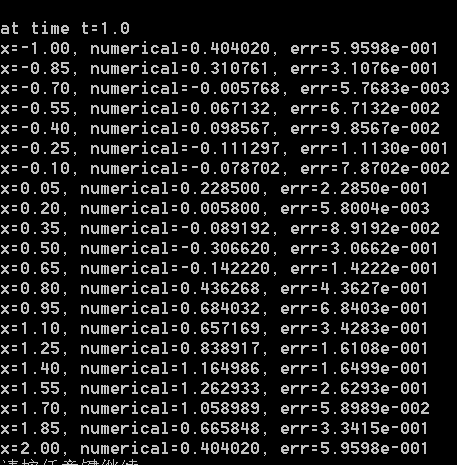
else

return phi(3 + x - t);

}

程序运行结果：





### Exch4\_04\_c.c

Beam-Warming格式

// Exch4\_04\_c.cpp : 定义控制台应用程序的入口点。

// 这是用Beam-Warming 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=0.0, t=0.05 和t=1.0时，x in [-1,2]的数值解

#include "stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main()

{

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

int j, k, m, n;

double phi(double x);

double exact(double x, double t);

m = 20;

n = 100;

xa = -1.0;

xb = 2.0;

h = (xb - xa) / 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] = xa + 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 = 2; j <= m ; j++)

u[j][k + 1] =u[j][k]-r\*(u[j][k]-u[j-1][k])- r\*(1.0-r)\*(u[j][k]-2\*u[j-1][k]+u[j-2][k])/ 2;

u[0][k + 1] = u[m][k + 1];//上面已经算出右边界的数值解，然后根据边界条件可得到左边界数值解

u[1][k + 1] = r\*u[0][k] + (1.0 - r)\*u[1][k];//为计算下一层上j=2处的数值解，还缺j=1处的数值解，为此用迎风格式通过j=0和j=1处的数值解进行计算

}

printf("at time t=0.0\n");

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

{

printf("x=%.2f, numerical=%f\n", x[j], u[j][0]);//屏幕输出，由于是初始状态无误差，故不必输出误差

}

printf("\n");

printf("at time t=0.05\n");

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

{

printf("x=%.2f, numerical=%f, err=%.4e\n", x[j], u[j][5], fabs(u[j][5] - exact(x[j], t[5])));

}

printf("\n");

printf("at time t=1.0\n");

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

{

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

}

free(x); free(t);

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

free(u[j]);

free(u);

}

double phi(double x)

{

if ((x < 0) || (x>1))

return 0.0;

else

return 1.0;

}

double exact(double x, double t)

{

if (t <= x + 1)

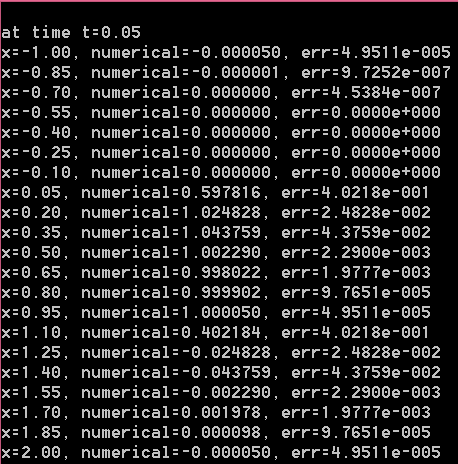
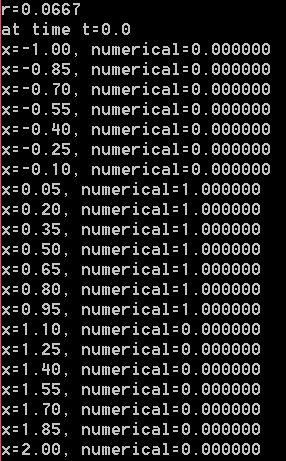
return phi(x - t);

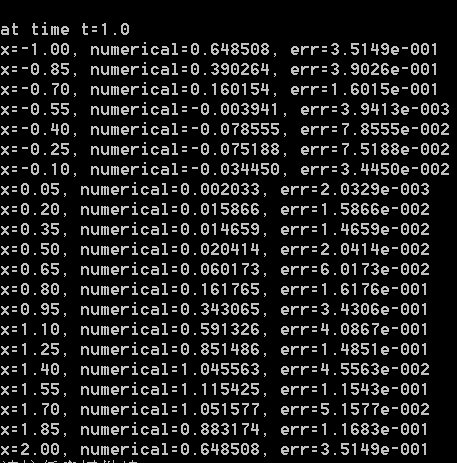
else

return phi(3 + x - t);

}

程序运行结果：





### Exch4\_05

解：首先对求解区域进行常规的一致网格剖分，得到网格节点。再将原方程弱化，使之在虚拟节点上成立，即有



然后先对时间偏导数进行差商近似，即取

 误差为

再对偏导数进行差商近似，即取

误差为。将上面两式代入离散节点处的方程，并记  则有



整理后将数值解代替精确解并忽略高阶小项，联合初边值条件可得原对流方程初边值问题的Crank-Nicolson格式：



易见，此格式的局部截断误差为。下面分析其增长因子。设，代入数值格式中就有，这样就得增长因子为



易得。注意到正规矩阵，从而无条件稳定。

### Exch4\_06.c

//二阶显格式求解二阶双曲型方程初边值问题

#include"stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

intm,n,i,j,k;

doublea,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);

m=40;

n=100;

a=1.0;

h=2.0/m;

tau=2.0/n;

r=a\*tau/h;

r=r\*r;

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

printf("h=%f, tau=%f\n", h, tau);

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

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

x[i]=-1.0+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]);

}

}

j=3\*m/5; //确定要打印的空间位置

k=n/10; //t方向每隔k个打印一下

for(i=n/2;i<=n;i=i+k)

{

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

}

}

double phi(double x)

{

return 0.0;

}

double psi(double x)

{

return 1.0/(1.0+x\*x);

}

double alpha(double t)

{

return t/2.0;

}

double beta(double t)

{

return t/2.0;

}

double f(doublex,double t)

{

return 2\*t\*(1.0-2\*x\*x-3.0\*pow(x,4))/pow((1.0+x\*x),4);

}

double exact(double x, double t)

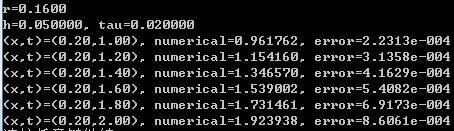
{

return t/(1.0+x\*x);

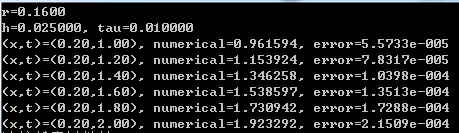
}

程序运行结果：

h=1/20, τ=1/50



h=1/40, τ=1/100



### Exch4\_07.c

//二阶隐格式求解二阶双曲型方程初边值问题

#include"stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

intm,n,i,j,k;

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

double phi(double x);

doubleddphi(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=100;

n=100;

a=1.0;

h=2.0/m;

tau=2.0/n;

r=a\*tau/h;

r=r\*r;

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

printf("h=%f, tau=%f\n", h, tau);

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

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

x[i]=-1.0+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);

j=3\*m/5; //确定要打印的空间位置

k=n/10; //t方向每隔k个打印一下

for(i=n/2;i<=n;i=i+k)

{

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

}

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

}

double phi(double x)

{

return 0.0;

}

double psi(double x)

{

return 1.0/(1.0+x\*x);

}

double alpha(double t)

{

return t/2.0;

}

double beta(double t)

{

return t/2.0;

}

double f(doublex,double t)

{

return 2\*t\*(1.0-2\*x\*x-3.0\*pow(x,4))/pow((1.0+x\*x),4);

}

double exact(double x, double t)

{

return t/(1.0+x\*x);

}

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

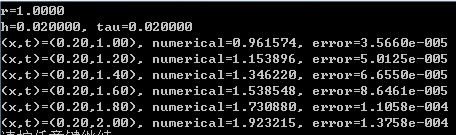
free(w);

return ans;

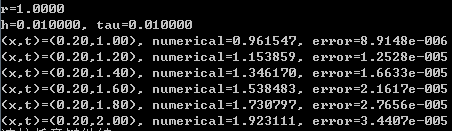
}

程序运行结果：

h=τ=1/50



h=τ=1/100



### Exch4\_08.c

//关于空间四阶，关于时间二阶的紧差分格式求解二阶双曲型方程初边值问题

#include"stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

void main( )

{

intm,n,i,j,k;

doublea,h,tau,r,\*x,\*t,\* \*u,\*a1,\*b,\*c,\*d,\*ans,c1,c2;

double phi(double x);

doubleddphi(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=80; //m=40;

n=160; //n=40;

a=1.0;

h=2.0/m;

tau=2.0/n;

r=a\*tau/h;

r=r\*r;

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

printf("h=%f, tau=%f\n", h, tau);

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

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

x[i]=-1.0+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);

j=3\*m/5; //确定要打印的空间位置

k=n/10; //t方向每隔k个打印一下

for(i=n/2;i<=n;i=i+k)

{

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

}

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

}

double phi(double x)

{

return 0.0;

}

double psi(double x)

{

return 1.0/(1.0+x\*x);

}

double alpha(double t)

{

return t/2.0;

}

double beta(double t)

{

return t/2.0;

}

double f(doublex,double t)

{

return 2\*t\*(1.0-2\*x\*x-3.0\*pow(x,4))/pow((1.0+x\*x),4);

}

double exact(double x, double t)

{

return t/(1.0+x\*x);

}

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

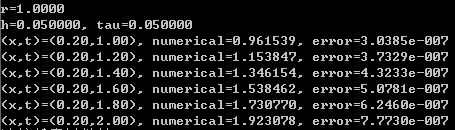
free(w);

return ans;

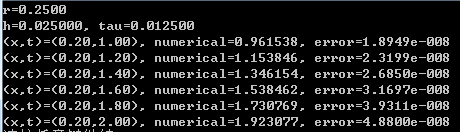
}

程序运行结果：

h=τ=1/20



h=1/40, τ=1/80



### Exch4\_09

a. 格式（1）局部截断误差的分析如下：

将格式（1）具体写出来就是



于是，此格式的局部截断误差为

 接下来考察。由教材（4-61）知，记 *P* 为点，则



同理， 将上面两式代入局部截断误差项，在精确解*u*函数的光滑性较高时，就有



b. 稳定性的讨论。设，代入格式（1）并利用（4-68）至（4-71）得



即

从中解出增长因子， 易见，，从而格式（1）无条件稳定。

c. 证明：从（2）的第二式中解出，然后代入第一式，就得，整理后就是格式（1），证毕。

### Exch4\_10.c

//求解二维双曲型方程初边值问题的交替方向隐格式

//这是右端添加辅助一个辅助项的交替格式（左端也添加了一项的，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]);

}

v[0][j] = -r2\*(u[0][j - 1][k + 1] + u[0][j + 1][k + 1]+ u[0][j - 1][k - 1] + u[0][j + 1][k - 1]) / 2.0 + (1 + r2)\*(u[0][j][k + 1]+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]+ u[m][j - 1][k - 1] + u[m][j + 1][k - 1]) / 2.0 + (1 + r2)\*(u[m][j][k + 1]+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]+r2\*(u[i][j-1][k-1]+u[i][j+1][k-1])/2.0-(1+r2)\*u[i][j][k-1];

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;

}

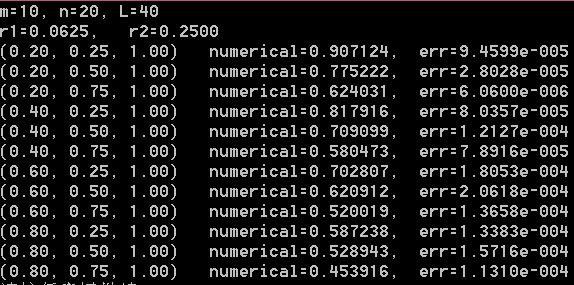
double exact(double x, double y, double t)

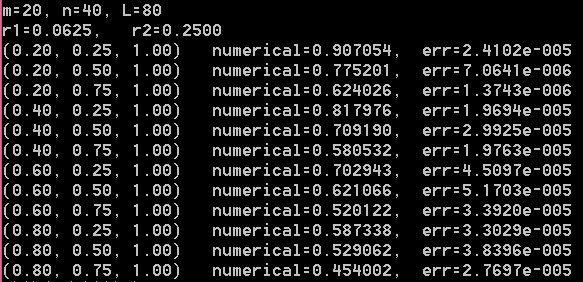
{

return t / (1 + x\*x + y\*y);

}

程序运行结果：





### Exch4\_11

a. 格式（3）的局部截断误差分析如下：

将格式（3）具体写出来就是



仿第9题分析知此格式的局部截断误差为

接下来分别考察和。记 *P* 为点，则第9题中已得





且 

将上面三个式子代入局部截断误差项，在精确解*u*函数的光滑性较高时，就有



b. 稳定性的讨论。设，代入格式（3）并利用（4-68）至（4-71）得

即



也就是满足以下方程：

其中，。

从中解出增长因子， 易见，，从而格式（1）无条件稳定。

c. 证明：从（4）的第二式中解出，然后代入第一式，就得，整理后就是格式（3），证毕。

d. 证明：格式（3）可以改写成





即，

就是（5），证毕。

e. 证明：从（6）的第三式中解出代入第二式，得



再代入第一式，得



这就是（5），证毕。

### Exch4\_12.c

//根据格式（4）设计的程序

//求解二维双曲型方程初边值问题的交替方向隐格式

//辅助项总共添加了三项（左边一项，右边两项）

#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++)

{

temp = u[i - 1][j - 1][k] + u[i + 1][j - 1][k] + u[i - 1][j + 1][k] + u[i + 1][j + 1][k];

temp = temp - 2 \* (u[i - 1][j][k] + u[i][j - 1][k] + u[i + 1][j][k] + u[i][j + 1][k]);

temp = temp + 4 \* u[i][j][k];

temp = r1\*r2\*temp / 2.0;

d1[i - 1] = 2 \* u[i][j][k] + dt\*dt\*f(x[i], y[j], t[k]) + temp;

}

v[0][j] = -r2\*(u[0][j - 1][k + 1] + u[0][j - 1][k - 1] + u[0][j + 1][k + 1] + u[0][j + 1][k - 1]) / 2 + (1 + r2)\*(u[0][j][k + 1] + 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] + u[m][j + 1][k + 1] + u[m][j + 1][k - 1]) / 2 + (1 + r2)\*(u[m][j][k + 1] + 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];

temp = u[i][0][k + 1] + u[i][0][k - 1];//相当于S\_i0

d2[0] = d2[0] + r2\*temp/ 2.0;

temp = u[i][n][k + 1] + u[i][n][k - 1];//相当于S\_in

d2[n - 2] = d2[n - 2] + r2\*temp/ 2.0;

ans = chase\_algorithm(a2, b2, c2, n - 1, d2); //算出来的ans 相当于S\_ij

for (j = 1; j <= n - 1; j++)

u[i][j][k + 1] = ans[j - 1]-u[i][j][k-1];//通过S\_ij计算u[i][j][k+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;

}

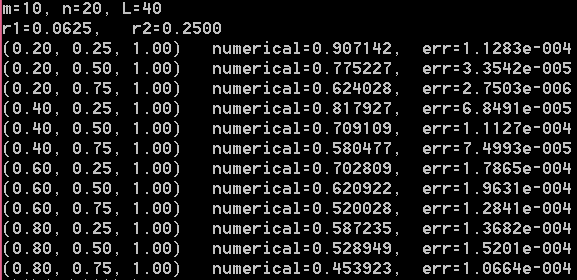
double exact(double x, double y, double t)

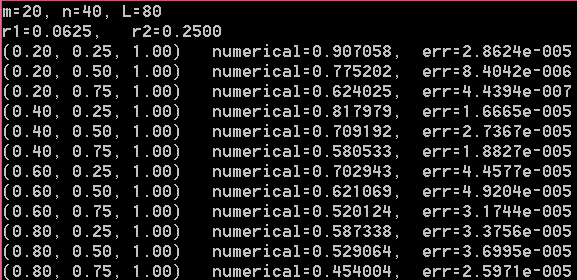
{

return t / (1 + x\*x + y\*y);

}

程序运行结果：





### Exch4\_13.c

// 根据格式（6）设计的程序：

//求解二维双曲型方程初边值问题的交替方向隐格式

//辅助项总共添加了三项（左边一项，右边两项）格式（6）正是孙志忠书上的格式

#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, \*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] = dt\*dt\*f(x[i], y[j], t[k]) + 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]);

}

temp = u[0][j - 1][k + 1] - 2 \* u[0][j][k + 1] + u[0][j + 1][k + 1] + u[0][j - 1][k - 1] - 2 \* u[0][j][k - 1] + u[0][j + 1][k - 1];

temp = temp - 2 \* (u[0][j - 1][k] - 2 \* u[0][j][k] + u[0][j + 1][k]);

v[0][j]= -r2\*temp / 2.0 + u[0][j][k + 1] - 2 \* u[0][j][k] + u[0][j][k - 1];

d1[0] = d1[0] + r1\*v[0][j] / 2.0;

temp = u[m][j - 1][k + 1] - 2 \* u[m][j][k + 1] + u[m][j + 1][k + 1] + u[m][j - 1][k - 1] - 2 \* u[m][j][k - 1] + u[m][j + 1][k - 1];

temp = temp - 2 \* (u[m][j - 1][k] - 2 \* u[m][j][k] + u[m][j + 1][k]);

v[m][j] = -r2\*temp / 2.0 + u[m][j][k + 1] - 2 \* u[m][j][k] + 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];

temp = u[i][0][k + 1] - 2 \* u[i][0][k] + u[i][0][k - 1];//相当于S\_i0

d2[0] = d2[0] + r2\*temp/ 2.0;

temp = u[i][n][k + 1] - 2 \* u[i][n][k] + u[i][n][k - 1];//相当于S\_in

d2[n - 2] = d2[n - 2] + r2\*temp/ 2.0;

ans = chase\_algorithm(a2, b2, c2, n - 1, d2);//算出来的是S\_ij

for (j = 1; j <= n - 1; j++)

u[i][j][k + 1] = ans[j - 1]+2\*u[i][j][k]-u[i][j][k-1]; //算出来的ans[j-1]相当于S\_ij,再利用S\_ij算U\_ij^(k+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);

}

程序运行结果：与上一题数值结果完全一样。

## 习题五

### Exch5\_01

证明：事实上，数值格式（5-3）写成矩阵形式后即为（5-6），可以证明（5-6）的系数矩阵记为*C*是对称正定的，从而（5-6）唯一可解。下证矩阵*C*正定。事实上，设有非零向量 其中，则



不妨设在边界节点上，则由（5-42）知，



由于不是零向量，则上式一定恒大于零，否则得到矛盾。这样说明矩阵*C*正定，于是以*C*为系数矩阵的线性方程组（5-6）唯一可解，即（5-3）唯一可解。以上分析还可以推广用于证明格式（5-33）唯一可解。

另证：解的唯一性。假设有数值解和都满足数值格式（5-3），则有满足



于是，由定理5.2知，，从而得到，即解唯一。

### Exch5\_02.c

#include "stdafx.h"

#include "math.h"

#include "stdlib.h"

#include "stdio.h"

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,num;

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 = 1.0;

xb = 2.0;

ya = 0.0;

yb = 3.0;

m = 20;

n =30;

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 =fabs( 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 = n / 6;//确定y方向每隔k个打印一下

num = m / 4;//确定x的位置 x＝1.25时，对应的x整数指标为num

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

{

printf("at (1.25, %.3f), numerical=%f, err=%.4e\n", y[j], u[num][j], fabs(exact(x[num], y[j]) - u[num][j]));//打印在x=1.25时的信息

}

num = 3 \* m / 4;//确定x的位置 x＝1.75时，对应的x整数指标为num

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

{

printf("at (1.75, %.3f), numerical=%f, err=%.4e\n", y[j], u[num][j], fabs(exact(x[num], y[j]) - u[num][j]));//打印在x=1.75时的信息

}

}

double leftboundary(double y)

{

return log(1.0+2\*y\*y);

}

double rightboundary(double y)

{

return log(4.0+2\*y\*y);

}

double bottomboundary(double x)

{

return 2\*log(x);

}

double topboundary(double x)

{

return log(18.0+x\*x);

}

double f(double x, double y)

{

double temp1, temp2,z;

temp1 = x\*x;

temp2 = y\*y;

z = temp1 + 2 \* temp2;

return (4\*temp2-2\*temp1)/(z\*z);

}

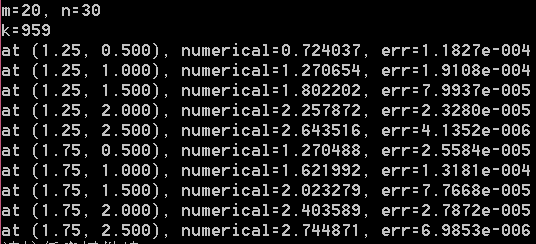
double exact(double x, double y)

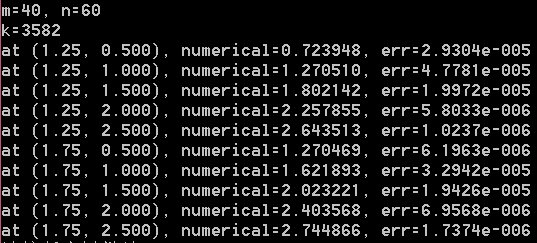
{

return log(x\*x+2\*y\*y);

}

程序运行结果：





### Exch5\_03.c

#include "stdafx.h"

#include "math.h"

#include "stdlib.h"

#include "stdio.h"

void main()

{

double xa, xb, ya, yb, dx, dy, beta, gamma, err, maxerr;

double \*x, \*y, \*\*u, \*\*g, \*\*temp, kexi, eta1, eta2;

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

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 = 1.0;

xb = 2.0;

ya = 0.0;

yb = 3.0;

m = 20;

n = 30;

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 = fabs(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 = n / 6;//确定y方向每隔k个打印一下

num = m / 4;//确定x的位置 x＝1.25时，对应的x整数指标为num

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

{

printf("at (1.25, %.3f), numerical=%f, err=%.4e\n", y[j], u[num][j], fabs(exact(x[num], y[j]) - u[num][j]));//打印在x=1.25时的信息

}

num = 3 \* m / 4;//确定x的位置 x＝1.75时，对应的x整数指标为num

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

{

printf("at (1.75, %.3f), numerical=%f, err=%.4e\n", y[j], u[num][j], fabs(exact(x[num], y[j]) - u[num][j]));//打印在x=1.75时的信息

}

free(x); free(y);

}

double leftboundary(double y)

{

return log(1.0 + 2 \* y\*y);

}

double rightboundary(double y)

{

return log(4.0 + 2 \* y\*y);

}

double bottomboundary(double x)

{

return 2 \* log(x);

}

double topboundary(double x)

{

return log(18.0 + x\*x);

}

double f(double x, double y)

{

double temp1, temp2, z;

temp1 = x\*x;

temp2 = y\*y;

z = temp1 + 2 \* temp2;

return (4 \* temp2 - 2 \* temp1) / (z\*z);

}

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;

}

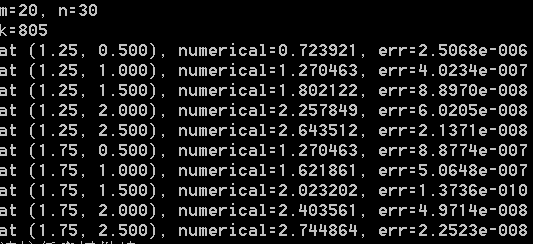
double exact(double x, double y)

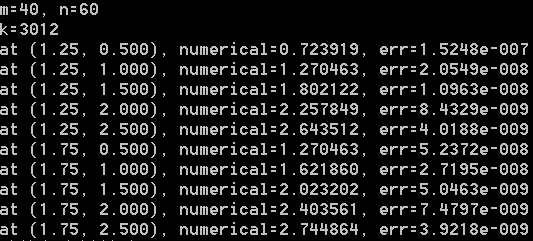
{

return log(x\*x + 2 \* y\*y);

}

程序运行结果：





### Exch5\_04.c

#include "stdafx.h"

#include "math.h"

#include "stdlib.h"

#include "stdio.h"

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, num;

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 = 1.0;

xb = 2.0;

ya = 0.0;

yb = 3.0;

m = 40;

n = 60;

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 = n / 6;//确定y方向每隔k个打印一下

num = m / 4;//确定x的位置 x＝1.25时，对应的x整数指标为num

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

{

printf("at (1.25, %.3f), numerical=%f, err=%.4e\n", y[j], u[num][j], fabs(exact(x[num], y[j]) - u[num][j]));//打印在x=1.25时的信息

}

num = 3 \* m / 4;//确定x的位置 x＝1.75时，对应的x整数指标为num

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

{

printf("at (1.75, %.3f), numerical=%f, err=%.4e\n", y[j], u[num][j], fabs(exact(x[num], y[j]) - u[num][j]));//打印在x=1.75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)

{

double temp1, temp2, z;

temp1 = x\*x;

temp2 = y\*y;

z = temp1 + 2 \* temp2;

return (4 \* temp2 - 2 \* temp1) / (z\*z);

}

double lambda\_function(double x, double y)

{

return 1.0;

}

double phi1(double y)

{

double z;

z = 1.0 + 2 \* y\*y;

return 2.0/z-log(z);

}

double phi2(double y)

{

double z;

z = 2 + y\*y;

return 2.0 / z + log(2 \* z);

}

double psi1(double x)

{

return -2\*log(x);

}

double psi2(double x)

{

double z;

z = x\*x + 18.0;

return 12.0/z+log(z);

}

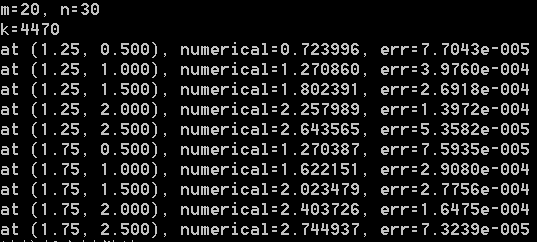
double exact(double x, double y)

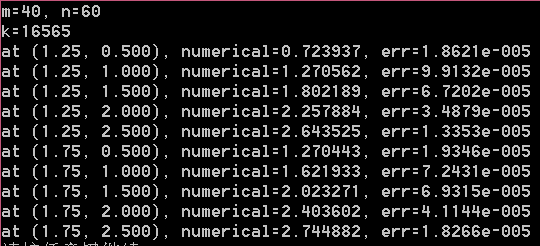
{

return log(x\*x + 2 \* y\*y);

}

程序运行结果：





### Exch5\_05

解：引入中间函数，则原四阶方程边值问题可以转化为以下二阶方程边值问题：



也就是可以分解为两个椭圆型方程的边值问题来求解，这两个问题分别是：

 和 

先用五点菱形格式求解第一个问题，得到其数值格式为：



求解得到数值解。然后再用五点菱形格式求解第二个问题，即



最终求解得到原来的函数的数值解。

## 习题六

### Exch6\_01

证：反证法。设存在使得，不妨设，则由的连续性知，存在的一个邻域记为，在内. 现在构造[0,1]上的函数：



显然且与已知条件矛盾，于是，假设不成立，从而再由的连续性知，，

同理。综上就有

### Exch6\_02

证：易见 

当即时，就有，从而Cauchy不等式成立。

当即时，取参数，就有



，从而Cauchy不等式成立。

### Exch6\_03

证：在区间上显然有，由罗尔定理，存在

使得。于是，在区间上，



故 

于是，



现将上式关于从0加到，就有，再用（6-19）即证得（6-27）。另外，易见，

于是，

这样就有 



现将上式关于从0加到，就有，即



联合（6-27）就得到，即（6-28）。证毕。

### Exch6\_04

证：由（6-19）及（6-27）易得（6-29）。下面利用对偶技巧证明（6-30）。设有定义在[0 , 1]上的函数满足以下边值问题：，则利用分部积分、（6-24）易知对任意有



再由的任意性取，利用插值误差（6-27）、（6-29）及所满足的方程就有



从而，也就是（6-30）成立。

### Exch6\_05.c

#include"stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

int n, elem; //n表示剖分数，node表示节点数，elem表示单元数

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 gauss = 0.5773502692; //数值积分中的高斯点

void main()

{

int m, i, j, k, t, row, coln, e;

doubleea[2][2], alpha[2],sum1, sum2;

double \* \*a, \*b, \*a1, \*b1, \*c1;

double \* \*g;

double f(double x);

double phi(inti, double x);

double fun1(inti, double x);

double exact(double x);

double fun2(inti, double x);

doubledxexact(double x);

double fun3(inti, double x);

double integral(double a, double b, inti, double (\* fun)(int, double));

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

m = 40; //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;

}

g = (double \*\*)malloc(sizeof(double \*)\*elem); //动态分配内存

for (e = 0; e<elem; e++)

g[e] = (double \*)malloc(sizeof(double)\* 2); //放置单元荷载

//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[e][k] = integral(xco[i], xco[j], lnd[e][k], fun1);

// 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[e][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;

b[0] = 0; b[m] = 0;

}

a[0][0] = 1; a[m][m] = 1;

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

k = m / 5;//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));

}

double f(double x) //右端项f(x)

{

double z;

z = -(x+2)\*exp(x);

return z;

}

double phi(inti, double x) //基函数

{

doubletemp,z;

temp = fabs(x - xco[i]);

if (temp <= h)

z = 1.0 - temp / h;

else

z = 0.0;

return z;

}

double fun1(inti, 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 x\*(exp(x)-exp(1.0));

}

double fun2(inti, 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;

}

doubledxexact(double x)//u(x)的导函数

{

return (x+1.0)\*exp(x)-exp(1.0);

}

double fun3(inti, 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, inti, 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));

returnans;

}

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

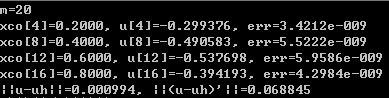
free(g);

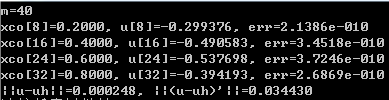
free(w);

returnans;

}

程序运行结果：





### Exch6\_06

证：任取，则因从而就有。于是，



可得



从而 

故



同理，

将上面两式相加，就得



其中。从而Poincaré不等式成立。

### Exch6\_07

证：设围成的闭区域为*D*，且区域*D*的边界（即三条线段构成的封闭曲线）设为*L*，逆时针方向，利用Green公式

，

从而有

注意到*L*由三条线段组成，由积分区域的可加性，先计算。

由于线段*AB*的方程可以写成

故取*x*为参数，就得



同理，

于是，

注意到



从而

即

### Exch6\_08

证：显然对坐标变换（6-57），将一般单元变至标准单元。由于二重积分的换元公式为



其中， 

这里用到了，故结论成立。

### Exch6\_09.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>

int m, n;

int node, elem, limnode;

int \* \*lnd;

double area, \*xco, \*yco, \*u;

#define pi 3.1415926535898

void main()

{

int \*limnd;

int i, j, k, t, e, row, coln;

double x, y, q, sum, summ;

double a, b, c, d, dx, dy;

double \* \*A, \*rhs, \*w, \*graduh;

double alpha[3], beta[3], ea[3][3], g[3];

double v(double x, double y);

double vx(double x, double y);

double vy(double x, double y);

double f(double x, 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 = pi;

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 = sin(x)\*sin(pi\*y);

return z;

}

double vx(double x, double y)//精确解关于x的偏导数

{

double z;

z =cos(x)\*sin(pi\*y);

return z;

}

double vy(double x, double y)//精确解关于y的偏导数

{

double z;

z = pi\*sin(x)\*cos(pi\*y);

return z;

}

double f(double x, double y)//方程的右端项

{

double z;

z = (pi\*pi+1.0)\*v(x,y);

return z;

}

double \*fun\_lambda(int e, double x, double y) //单元e上的基函数lambda0,lambda1, lambda2

{

int i, 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

{

int i, 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

{

int i, 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

{

int i, 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;

int i, 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];

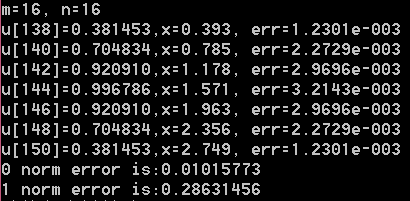
x[i] = b[i] - sum;

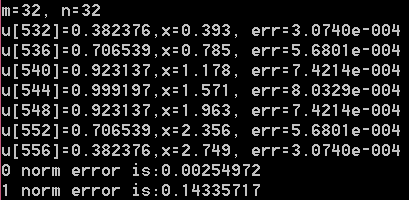
}

return x;

}

程序运行结果：





### Exch6\_10.c

//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<1,0<t<=T;

//u(x,0)=psi(x), 0<=x<=1;

//u(0,t)=u(pi,t)=0; 0<t<=T

#include"stdafx.h"

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

double h, tau, T, \*xco, \*\*alpha;

int m, n;

#define pi 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 =16;

n =16 ;

printf("m=%d, n=%d\n", m, n);

h = 1.0 / m; //空间步长

T = 1.0; //时间终点

tau = T / n; //时间步长

a1 = 1.0; //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, alpha[i][n], fabs(exact(xco[i], T) - alpha[i][n]));

}

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

free(alpha[i]);

free(xco); free(alpha); free(t);

}

double f(double x, double t)//右端项函数f(x,t)

{

return 0;

}

double phi(inti, 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 psi(double x)

{

return sin(pi\*x);

}

double fun1(inti, double x, double t)//算单元荷载时的被积函数

{

return f(x,t)\*phi(i, x);

}

double integral(double a, double b, inti, double t, 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(double x, double t)

{

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

}

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

}

程序运行结果：

