

间断有限元第五次作业报告

九所 韩若愚

2022.5.10

目录

1	题目	1
2	真解	2
3	算法	2
3.1	bad DG	2
3.2	LDG	3
4	数值结果	4
4.1	bad DG	4
4.2	LDG	5
5	分析	5
6	代码	6

1 题目

Code the following schemes for

$$\begin{cases} u_t = u_{xx}, & 0 \leq x \leq 2\pi \\ u(x, 0) = \sin(x) \end{cases}$$

Take the final time as $T = 1$.

1. bad DG scheme. Show pictures of the exact solution and the numerical solution.
2. LDG method with the central flux. Show error tables.
3. LDG method with alternating flux. Show error tables.

2 真解

不妨假设 $u(x, t) = A(x)B(t)$, 将之带入方程, 得到:

$$A \cdot B' = A'' \cdot B$$

假设 A, B 不为 0, 则 $B'/B = A''/A$. 这个方程等号左边是只和 t 有关的函数, 右边是只和 x 有关的函数, 所以 $B'/B = A''/A = c = \text{const}$. 于是解两个常微分方程 $B' = cB$, $A'' = cA$, 且注意初值条件 $u(x, 0) = A(x)B(0) = \sin(x)$, 得到 $B = \exp(-t)$, $A = \sin(x)$. 所以 u 为:

$$u(x, t) = e^{-t} \sin(x).$$

显然此时 u 满足方程, u 是方程的真解。

3 算法

3.1 bad DG

将方程 $u_t = u_{xx}$ 看成:

$$u_t - (u_x)_x = 0$$

上面的方程显然是守恒形式, 通量函数为: $f = u_x$. 可以用之前求解守恒律使用的方法求解。

先对单元 $[0, 2\pi]$ 进行均匀剖分。假设将区间均匀剖分为 n 份, 令:

$$0 = x_{\frac{1}{2}} < x_{\frac{3}{2}} < \cdots < x_{n-\frac{1}{2}} < x_{n+\frac{1}{2}} = 2\pi$$

则第 j 个区间为: $I_j = [x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}]$, 记每个区间的长度都为 $h = \frac{2\pi}{n}$, $x_{j+1/2}^- = \lim_{x \in I_j, x \rightarrow x_{j+1/2}} x$, $x_{j+1/2}^+ = \lim_{x \in I_{j+1}, x \rightarrow x_{j+1/2}} x$ 。则此时 DG 格式为:

求 $u_h \in V_h^k$, $V_h^k = \{v | v|_{I_j} \in P^k(I_j), \forall j = 1, \dots, n\}$, 使得对 $\forall v_h \in V_h^k, \forall j = 1, \dots, n$, 都有:

$$\int_{I_j} (u_h)_t v_h dx = - \int_{I_j} (u_h)_x (v_h)_x dx + \hat{u}_{x_{j+1/2}} (v_h)_{j+1/2}^- - \hat{u}_{x_{j-1/2}} (v_h)_{j-1/2}^+$$

其中 \hat{u}_x 为对 u_x 进行近似的数值通量。

假设参考单元 $I = [-1, 1]$, 在参考单元上取一组 $P^k(I)$ 的基底 $\{\phi^l\}_{l=0}^k$, 则通过同胚映射 $x = \frac{h}{2}(\xi + 1) + x_{j-1/2}$, 能将 u_h 表示为:

$$u_h(x)|_{I_j} = \sum_{l=0}^k u_j^l(t) \phi^l(\xi), \quad j = 1, \dots, n$$

令 $v_h = \phi^m$, $m = 1, \dots, n$, 则 DG 格式变为:

$$\begin{aligned} \frac{h}{2} \sum_{l=0}^k \left(\int_{I_j} \phi^l \phi^m d\xi \right) \frac{d}{dt} u_j^l &= - \frac{2}{h} \sum_{l=0}^k \left(\int_{I_j} \phi_\xi^l \phi_\xi^m d\xi \right) u_j^l \\ &+ \hat{u}_{x_{j+1/2}} \phi^m(1) - \hat{u}_{x_{j-1/2}} \phi^m(-1), \quad j = 1, \dots, n \end{aligned}$$

在计算中取 $\hat{u}_{x_{j+1/2}} = ((u_x)_{j+1/2}^+ + (u_x)_{j+1/2}^-)/2$, 其中: $(u_x)_{j+1/2}^+ = \frac{2}{h} \sum_{l=0}^k u_{j+1}^l \phi_\xi^l(-1)$, $(u_x)_{j+1/2}^- = \frac{2}{h} \sum_{l=0}^k u_j^l \phi_\xi^l(1)$ 。

在时间方向上使用 SSPRK 进行计算。

3.2 LDG

在 LDG 中引入了辅助变量 q , 方程变为:

$$\begin{cases} q = u_x \\ u_t = q_x \end{cases}$$

假设解空间 V_h^k 不变, 则 LDG 为: 求 $q_h, u_h \in V_h^k$, 使得对 $\forall p_h, v_h \in V_h^K$, $\forall j = 1, \dots, n$, 都有:

$$\begin{cases} \int_{I_j} q_h p_h dx = & - \int_{I_j} u_h (p_h)_x dx + \hat{u}_{j+1/2} (p_h)_{j+1/2}^- - \hat{u}_{j-1/2} (p_h)_{j-1/2}^+ \\ \int_{I_j} (u_h)_t v_h dx = & - \int_{I_j} q_h (v_h)_x dx + \hat{q}_{j+1/2} (v_h)_{j+1/2}^- - \hat{q}_{j-1/2} (v_h)_{j-1/2}^+ \end{cases}$$

设 $u_h(x, t)|_{I_j} = \sum_{l=0}^k u_j^l(t) \phi^l(\xi(x))$, $q_h(x, t)|_{I_j} = \sum_{l=0}^k q_j^l(t) \phi^l(\xi(x))$, 取 $p_h = v_h = \phi^m$, $m = 1, \dots, n$, 则方程为:

$$\begin{cases} \frac{h}{2} \sum_{l=0}^k (\int_I \phi^l \phi^m d\xi) q_j^l = & - \sum_{l=0}^k (\int_I \phi^l (\phi^m)_\xi d\xi) u_j^l \\ & + \hat{u}_{j+1/2} \phi^m(1) - \hat{u}_{j-1/2} \phi^m(-1) \\ \frac{h}{2} \sum_{l=0}^k (\int_I \phi^l \phi^m d\xi) \frac{d}{dt} u_j^l = & - \sum_{l=0}^k (\int_I \phi^l (\phi^m)_\xi d\xi) q_j^l \\ & + \hat{q}_{j+1/2} \phi^m(1) - \hat{q}_{j-1/2} \phi^m(-1) \end{cases}$$

在第 m 次计算时, u_h 的系数矩阵 (u_j^l) 是已知的, 所以可以算出 q_h 的系数矩阵 (q_j^l) , 再用 (q_j^l) 通过 SSPRK 求解下一时刻 u_h 的系数。

数值通量 $\hat{q}_{j+1/2}$, $\hat{u}_{j+1/2}$ 可以选择成:

1. 中心通量: $\hat{q}_{j+1/2} = \frac{1}{2}(q_{j+1/2}^+ + q_{j+1/2}^-)$, $\hat{u}_{j+1/2} = \frac{1}{2}(u_{j+1/2}^+ + u_{j+1/2}^-)$
2. 交错通量: $\hat{q}_{j+1/2} = q_{j+1/2}^+$, $\hat{u}_{j+1/2} = u_{j+1/2}^-$ 。

4 数值结果

4.1 bad DG

利用 bad DG 求解得到的数值结果图像见图 1。其中数值解在第 j 个点的值取为 u_h 在 $x_j = (x_{j-1/2} + x_{j+1/2})/2$ 处的值。

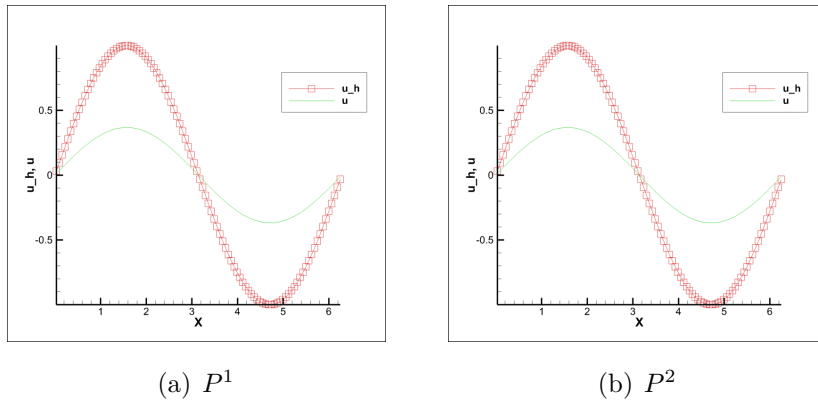


图 1: bad DG at $T = 1$

4.2 LDG

利用 LDG 求解得到的误差表见表 1、表 2，保留三位小数。

表 1: P^1

LDG with central flux				
n	L^2 error	order	L^∞ error	order
20	1.505e-2		1.449e-2	
40	7.424e-3	1.019	7.226e-3	1.004
80	3.699e-3	1.005	3.611e-3	1.009
160	1.848e-3	1.001	1.805e-3	1.000
320	9.239e-4	1.000	9.026e-4	1.000
LDG with alternating flux				
n	L^2 error	order	L^∞ error	order
20	3.922e-3		6.010e-3	
40	9.794e-4	2.002	1.510e-3	1.993
80	2.448e-4	2.000	3.780e-4	1.998
160	6.120e-5	2.000	9.453e-5	2.000
320	1.530e-5	2.000	2.363e-5	2.000

表 2: P^2

LDG with central flux				
n	L^2 error	order	L^∞ error	order
20	6.442e-5		9.650e-5	
40	7.983e-6	3.013	1.195e-5	3.014
80	9.962e-7	3.002	1.506e-6	2.988
160	1.245e-7	3.000	1.896e-7	2.990
320	1.565e-8	2.992	2.386e-8	2.990
LDG with alternating flux				
n	L^2 error	order	L^∞ error	order
20	9.866e-5		1.886e-4	
40	1.233e-5	3.000	2.374e-5	2.990
80	1.542e-6	2.999	2.989e-6	2.990
160	1.934e-7	2.995	3.910e-7	2.934
320	2.434e-8	2.990	5.120e-8	2.933

5 分析

图 1显示当使用 bad DG 时，得到的数值结果和真解并不相符，产生这种问题的原因是这个格式并不稳定。所以 bad DG 并不是求解对流扩散方程的好方法。使用 LDG 能得到符合真解的结果。LDG 的结果会受到通量函数的选取的影响。当使用中心通量时，在 $k = 1$ 的情况下使用中心通量时 L^1 和 L^∞ 范数下的收敛阶为 1 阶，但是使用交错通量时收敛阶是 2 阶。当 $k = 2$ 时无论哪个收敛阶都保持在 3 阶左右。

6 代码

本次报告程序使用 C++ 编译。

1 #include <iostream>

2 #include <cmath>

3 #include <fstream>

```

4      using namespace std;
5
6
7      //*****部分参数*****//
8      const int n = 320;    //划分单元个数
9      const int k = 2;      //多项式最高次项次数
10
11     const double pi = 3.1415926;
12     const double h = 2 * pi / n;    //空间步长
13     double dt = 1e-5;    //时间步长
14     double p[n+1];    //节点位置,  $p_j = j * h = x_{\{j+1/2\}}$ ,  $j = 0, 1, \dots, n$ 
15
16     const double lobattopoint[5] = {-1.0,
17                                     -0.6546536707079771, 0, 0.6546536707079771,
18                                     1.0};
19
20     const double lobattoco[5] = {0.1,
21                                 0.5444444444444444, 0.7111111111111111,
22                                 0.5444444444444444, 0.1};
23
24     //*****参数定义完毕*****//
25
26     //*****函数声明*****//
27     double u_0(double x);    //初值
28     double u_exact(double x, double t);    //真解
29     double phi(int l, double x);    //参考单元基函数
30
31     double phix(int l, double x);    //基函数一阶导
32     double** initial();    //计算初始时刻  $u_j$ 
33     double fluxu(double ul, double ur);    //数值通量计算
34     double fluxq(double ql, double qr);

```

```
28     void getq(double ut);
29     double** L(double** ut);    //用于计算RK的函
    数,  $u_t = F(u)$ 
30     double** RK22(double** un, double dtn);    //2
    步二阶RK
31     double** RK33(double** un, double dtn);
32     double** RK(int k, double** un, double dtn);
33     //*****声明完毕*****//
34
35
36     int main()
37     {
38         int i, j, l;
39         double t, temp1, temp2, norm1, norm2, xi;
40         double T = 1;
41         double** u1 = new double* [n];
42         double** u2 = new double* [n];
43         for (i=0; i<n; i++)
44         {
45             u1[i] = new double [k+1];
46             u2[i] = new double [k+1];
47         }
48         for (j=0; j<=n; j++)
49         {
50             p[j] = j * h;
51         }
52
53         u1 = initial();
54         t = 0;
55         while(t<T)
56         {
```



```
57         if ( t + dt >T)
58         {
59             dt = T - t;
60         }
61         t = t + dt;
62         u2 = RK(k,u1,dt);
63         u1 = u2;
64         cout<<"t="<<t<<endl;
65     }
66
67     //计算误差
68     norm1 = 0;
69     norm2 = 0;
70     for ( j=1; j<=n; j++)
71     {
72         for ( i=0; i<5; i++)
73         {
74             xi = lobattopoint[i];
75             temp1 = 0;
76             for ( l=0; l<=k; l++)
77             {
78                 temp1 = temp1 + u1[j-1][l] * phi(l,xi);
79             }
80
81             temp2 = h * (xi + 1) / 2. + p[j-1];
82             temp1 = u_exact(temp2,T) - temp1;
83
84             if (abs(temp1) > norm2)
85             {
86                 norm2 = abs(temp1);
87             }
```

```

88
89     temp1 = temp1 * temp1;
90     norm1 = norm1 + lobattoco[i] * temp1;
91 }
92
93 }
94 norm1 = norm1 * h / 2.;
95 norm1 = sqrt(norm1);
96 cout<<"L2="<<norm1<<endl<<"Linf="<<norm2<<endl
    ;//*/
97
98 //开始画图
99 {
100     const char* fn = "DGLecture\\homework5\\LDG.
        plt";
101     remove(fn);
102     fstream f, f1;
103     f.open(fn, ios::out | ios::app);
104     f<<"VARIABLES="<<"X"<<"", "<<"u_h"<<"", "<<"u"<<
        endl;
105     for (j=1; j<=n; j++)
106     {
107         temp1 = 0;
108         for (l=0; l<=k; l++)
109         {
110             temp1 = temp1 + u1[j-1][l] * phi(l,0);
111         }
112         f<<"\t"<<(p[j-1] + p[j])/2.0<<"\t"<<temp1<<"\t
            "<<u_exact((p[j-1] + p[j])/2.0,T)<<endl;
113
114     }

```

```
115         f.close();
116     }
117
118     for (i=0; i<n; i++)
119     {
120         delete [] u1[i];
121         delete [] u2[i];
122     }
123     delete [] u1;
124     delete [] u2;
125
126     system("pause");
127 }
128
129 //*****函数定义*****//
130
131 double u_0(double x)
132 {
133     return sin(x);
134 }
135
136 double u_exact(double x, double t)
137 {
138     double ans;
139     ans = exp(-t) * sin(x);
140     return ans;
141 }
142
143 double phi(int l, double x)
144 {
145     if (l==0)
```

```
146     {
147         return 1;
148     }
149     else if (l == 1)
150     {
151         return x;
152     }
153     else if (l == 2)
154     {
155         return (3*x*x - 1)/2;
156     }
157     else{
158         return 0;
159     }
160 }
161
162 double** initial()
163 {
164     double ans, temp;
165     int j, l, m;
166     double** ut = new double* [n];
167     double* Bt = new double [n];
168     for (j=0; j<n; j++)
169     {
170         ut[j] = new double [k+1];
171     }
172
173     for (j=1; j<=n; j++)
174     {
175         for (m=0; m<=k; m++)
176         {
```

```

177     ans = 0;
178     for (l=0; l<5; l++)
179     {
180         temp = h * (lobattopoint[l] + 1)/2 + p[j-1];
181         ans = ans + lobattoco[l] * u_0(temp) * phi(m,
182             lobattopoint[l]);
183     }
184     ans = ans / 2;
185     Bt[m] = ans;
186 }
187
188 double A[3][3] = {{1,0,0},{0,3,0},{0,0,5}};
189 for (m=0; m<=k ;m++)
190 {
191     ut[j-1][m] = 0;
192     for (l=0; l<=k; l++)
193     {
194         ut[j-1][m] = ut[j-1][m] + A[m][l] * Bt[l];
195     }
196 }
197 delete [] Bt;
198 return ut;
199 }
200
201 double fluxu(double ul, double ur)
202 {
203     double ans;
204     //average flux
205     //ans = 0.5 * (ul + ur);
206 }

```

```
207      //alternating flux
208      ans = ul;
209
210      return ans;
211  }
212
213  double fluxq(double ql, double qr)
214  {
215      double ans;
216      //average flux
217      //ans = 0.5 * (ql + qr);
218
219      //alternating flux
220      ans = qr;
221
222      return ans;
223  }
224
225  double** getq(double** ut)
226  {
227      int j, l, m, r;
228      double ul, ur;
229      double** q = new double* [n];
230      for (j=0; j<n; j++)
231      {
232          q[j] = new double [k+1];
233      }
234
235      double A[3] = {1,3,5};
236      double B[3][3] = {{0,0,0},{2,0,0},{0,2,0}};
237
```

```
238     for ( j=1; j<=n; j++)
239     {
240     for (m=0; m<=k; m++)
241     {
242     q[j-1][m] = 0;
243
244     for ( l=0; l<=k; l++)
245     {
246     q[j-1][m] = q[j-1][m] - B[m][l] * ut[j-1][l];
247     }
248     //第一个数值通量
249     {
250     ul = 0;
251     ur = 0;
252
253     r = j;
254     if (r == n)
255     {
256     r = 0;
257     }
258     for ( l=0; l<=k; l++)
259     {
260     ul = ul + ut[j-1][l] * phi(l,1);
261     ur = ur + ut[r][l] * phi(l,-1);
262     }
263
264     q[j-1][m] = q[j-1][m] + fluxu(ul, ur) * phi(m
265     ,1);
266     }
267     //第二个数值通量
268     {
```

```
268         ul = 0;
269         ur = 0;
270
271         r = j - 2;
272         if (r == -1)
273         {
274             r = n - 1;
275         }
276         for (l = 0; l <= k; l++)
277         {
278             ul = ul + ut[r][l] * phi(l, 1);
279             ur = ur + ut[j - 1][l] * phi(l, -1);
280         }
281
282         q[j - 1][m] = q[j - 1][m] - fluxu(ul, ur) * phi(m
283             , -1);
284
285         q[j - 1][m] = q[j - 1][m] * A[m] / h;
286     }
287 }
288
289 return q;
290
291 double** L(double** ut)
292 {
293     int i, j, l, m, p;
294     double ql, qr;
295     double** ans = new double* [n];
296     double** q = new double* [n];
297     for (i = 0; i < n; i++)
```



```
298     {
299         ans[i] = new double [k+1];
300         q[i] = new double [k+1];
301     }
302     q = getq(ut);
303
304     double A[3] = {1,3,5};
305     double B[3][3] = {{0,0,0},{2,0,0},{0,2,0}};
306
307     for (j=1; j<=n; j++)
308     {
309         for (m=0; m<=k; m++)
310         {
311             ans[j-1][m] = 0;
312
313             for (l=0; l<=k; l++)
314             {
315                 ans[j-1][m] = ans[j-1][m] - B[m][l] * q[j-1][l
316                     ];
317             }
318
319             //第一个数值通量
320             {
321                 p = j;
322                 if (p == n)
323                 {
324                     p = 0;
325
326                     ql = 0;
327                     qr = 0;
```

```

328
329     for (l=0; l<=k; l++)
330     {
331         ql = ql + q[j-1][l] * phi(l,1);
332         qr = qr + q[p][l] * phi(l,-1);
333     }
334
335     ans[j-1][m] = ans[j-1][m] + fluxq(ql,qr) * phi
        (m,1);
336 }
337 //第二个数值通量
338 {
339     p = j-2;
340     if (p == -1)
341     {
342         p = n-1;
343     }
344
345     ql = 0;
346     qr = 0;
347
348     for (l=0; l<=k; l++)
349     {
350         ql = ql + q[p][l] * phi(l,1);
351         qr = qr + q[j-1][l] * phi(l,-1);
352     }
353
354     ans[j-1][m] = ans[j-1][m] - fluxq(ql,qr) * phi
        (m,-1);
355 }
356 ans[j-1][m] = ans[j-1][m] * A[m] / h;

```

```
357     }
358
359     }
360     for ( i=0; i<n; i++)
361     {
362         delete [] q[i];
363     }
364     delete [] q;
365     return ans;
366 }
367
368 double** RK22(double** un, double dtn)
369 {
370     int j,l,m;
371     double ul;
372     double** ans = new double* [n];
373     for (j=0; j<n; j++)
374     {
375         ans[j] = new double [k+1];
376     }
377     double** u0 = new double* [n];
378     double** u1 = new double* [n];
379     double** u2 = new double* [n];
380
381     for (j=0; j<n; j++)
382     {
383         u0[j] = new double [k+1];
384         u1[j] = new double [k+1];
385         u2[j] = new double [k+1];
386     }
387
```

```
388     for (j=1; j<=n; j++)
389     {
390     for (l=0; l<=k; l++)
391     {
392     u0[j-1][l] = un[j-1][l];
393     }
394     }
395
396     u1 = L(u0);
397     for (j=1; j<=n; j++)
398     {
399     for (l=0; l<=k; l++)
400     {
401     u1[j-1][l] = u1[j-1][l] * dtn + u0[j-1][l];
402     }
403     }
404
405     u2 = L(u1);
406     for (j=1; j<=n; j++)
407     {
408     for (l=0; l<=k; l++)
409     {
410     u2[j-1][l] = u0[j-1][l] * 0.5 + u1[j-1][l] *
        0.5 + u2[j-1][l] * 0.5 * dtn;
411     ans[j-1][l] = u2[j-1][l];
412     }
413     }
414
415     delete [] u0;
416     delete [] u1;
417     delete [] u2;
```

```
418
419     return ans;
420 }
421
422 double** RK33(double** un, double dtn)
423 {
424     int j, l, m;
425     double ul;
426     double** ans = new double* [n];
427     for (j=0; j<n; j++)
428     {
429         ans[j] = new double [k+1];
430     }
431     double** u0 = new double* [n];
432     double** u1 = new double* [n];
433     double** u2 = new double* [n];
434     double** u3 = new double* [n];
435
436     for (j=0; j<n; j++)
437     {
438         u0[j] = new double [k+1];
439         u1[j] = new double [k+1];
440         u2[j] = new double [k+1];
441         u3[j] = new double [k+1];
442     }
443
444     for (j=1; j<=n; j++)
445     {
446         for (l=0; l<=k; l++)
447         {
448             u0[j-1][l] = un[j-1][l];
```

```
449     }
450     }
451
452     u1 = L(u0);
453     for (j=1; j<=n; j++)
454     {
455         for (l=0; l<=k; l++)
456         {
457             u1[j-1][l] = u1[j-1][l] * dtm + u0[j-1][l];
458         }
459     }
460
461     u2 = L(u1);
462     for (j=1; j<=n; j++)
463     {
464         for (l=0; l<=k; l++)
465         {
466             u2[j-1][l] = u0[j-1][l] * 3 / 4.0 + u1[j-1][l]
467                 / 4 + u2[j-1][l] * dtm / 4.0;
468         }
469     }
470
471     u3 = L(u2);
472     for (j=1; j<=n; j++)
473     {
474         for (l=0; l<=k; l++)
475         {
476             u3[j-1][l] = u0[j-1][l] / 3.0 + 2 * u2[j-1][l]
477                 / 3.0 + 2 * dtm * u3[j-1][l] / 3.0;
478             ans[j-1][l] = u3[j-1][l];
479         }
480     }
```

```
478     }
479
480     delete [] u0;
481     delete [] u1;
482     delete [] u2;
483     delete [] u3;
484
485     return ans;
486 }
487
488 double** RK(int k, double** un, double dtn)
489 {
490     double** u2;
491     if ( k == 1 )
492     {
493         u2 = RK22(un, dtn);
494     }
495     else if (k == 2)
496     {
497         u2 = RK33(un, dtn);
498     }
499     else{
500         ;
501     }
502
503     return u2;
504 }
```