

```

1: //利用有限體積法求解二維穩態熱擴散方程式
2: #include <iostream>
3: #include <fstream>
4: #include <sstream>
5: #include <cmath>
6: #include <cstdlib>
7: #include <string>
8: #include <vector>
9: #include <iomanip>
10: #define pi 3.14159265358979323846
11:
12: using namespace std;
13:
14: //會用到的參數
15: int nx_data[] = {10, 20, 40, 80};
16: int ny_data[] = {10, 20, 40, 80};
17:
18: vector<int> Nx(nx_data, nx_data + sizeof(nx_data)/sizeof(nx_data[0]));
19: vector<int> Ny(ny_data, ny_data + sizeof(ny_data)/sizeof(ny_data[0]));
20: double FinalL1error[4];
21: int n, NX, NY;
22: double dx, dy;
23: vector<vector<double> > > a;
24: vector<double> b;
25: vector<double> x, x_old;
26: vector<vector<double> > > T;
27: int G, max_G = 100000;
28: double T_left = 10.0;
29: double T_right = 10.0;
30: double T_bottom = 10.0;
31: double L1sum; //計算L1誤差
32: double maxerror; //迭代收斂誤差
33: const double tolerance = 1e-18;
34: bool steadystate;
35:
36: double T_up(int i){
37:     double x_pos = (i - 0.5) * dx; // 使用網格中心點座標
38:     return 10.0 + sin(pi * x_pos);
39: }
40:
41: double T_analytical_fixed(double x_pos, double y_pos){
42:     return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
43: }
44:
45: double T_analytical(int k){
46:     int i, j;
47:     // 修正索引計算
48:     i = ((k-1) % NX) + 1; // i = [1:NX]
49:     j = ((k-1) / NX) + 1; // j = [1:NY]
50:
51:     // 注意：這裡的座標計算要與網格一致
52:     // 對於從1開始的索引，x_pos和y_pos的計算如下：
53:     double x_pos = (i - 0.5) * dx; // 網格中心點
54:     double y_pos = (j - 0.5) * dy; // 網格中心點

```

```

55:
56: return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
57: }
58:
59: //初始化矩陣
60: void initial(vector<vector<double> >& a, vector<double>& b, int n) {
61:     // 初始化所有元素為0
62:     for(int i = 1; i <= n; i++) {
63:         for(int j = 1; j <= n; j++) {
64:             a[i][j] = 0.0;
65:         }
66:         b[i] = 0.0;
67:     }
68:
69:     for(int i = 0; i <= n+1; i++){
70:         x[i] = 0.0;
71:         x_old[i] = 0.0;
72:     }
73:
74:     // 設定邊界條件和係數矩陣
75:     // 四個角點
76:     // 左下角 (1,1)
77:     a[1][1] = 6.0;
78:     a[1][2] = -1.0; // 右鄰點
79:     a[1][1+NX] = -1.0; // 上鄰點
80:     b[1] = 2.0 * (T_left + T_bottom);
81:
82:     // 右下角 (NX,1)
83:     a[NX][NX] = 6.0;
84:     a[NX][NX-1] = -1.0; // 左鄰點
85:     a[NX][NX+NX] = -1.0; // 上鄰點
86:     b[NX] = 2.0 * (T_right + T_bottom);
87:
88:     // 左上角 (1,NY)
89:     a[n-NX+1][n-NX+1] = 6.0;
90:     a[n-NX+1][n-NX+2] = -1.0; // 右鄰點
91:     a[n-NX+1][n-NX+1-NX] = -1.0; // 下鄰點
92:     double T_up_left = 10.0 + sin(pi * (0.5 * dx)); // 左上角的上邊界值
93:     b[n-NX+1] = 2.0 * (T_left + T_up_left);
94:     // 右上角 (NX,NY)
95:     a[n][n] = 6.0;
96:     a[n][n-1] = -1.0; // 左鄰點
97:     a[n][n-NX] = -1.0; // 下鄰點
98:     double T_up_right = 10.0 + sin(pi * ((NX - 0.5) * dx)); //
    右上角的上邊界值
99:     b[n] = 2.0 * (T_right + T_up_right);
100:
101:     // 下邊界 (除角點外)
102:     for(int i = 2; i <= NX-1; i++) {
103:         a[i][i] = 5.0;
104:         a[i][i+1] = -1.0; // 右鄰點
105:         a[i][i-1] = -1.0; // 左鄰點
106:         a[i][i+NX] = -1.0; // 上鄰點
107:         b[i] = 2.0 * T_bottom;

```

```

108: }
109:
110: // 上邊界 (除角點外)
111: for(int i = 2; i <= NX-1; i++) {
112:     int idx = n - NX + i;
113:     a[idx][idx] = 5.0;
114:     a[idx][idx+1] = -1.0; // 右鄰點
115:     a[idx][idx-1] = -1.0; // 左鄰點
116:     a[idx][idx-NX] = -1.0; // 下鄰點
117:     double T_up_val = 10.0 + sin(pi * ((i - 0.5) * dx)); // 使用網格中心點
118:     b[idx] = 2.0 * T_up_val;
119: }
120:
121: // 左邊界 (除角點外)
122: for(int j = 2; j <= NY-1; j++){
123:     int idx = (j-1) * NX + 1;
124:     a[idx][idx] = 5.0;
125:     a[idx][idx+1] = -1.0; // 右鄰點
126:     a[idx][idx+NX] = -1.0; // 上鄰點
127:     a[idx][idx-NX] = -1.0; // 下鄰點
128:     b[idx] = 2.0 * T_left;
129: }
130:
131: // 右邊界 (除角點外)
132: for(int j = 2; j <= NY-1; j++){
133:     int idx = (j-1) * NX + NX;
134:     a[idx][idx] = 5.0;
135:     a[idx][idx-1] = -1.0; // 左鄰點
136:     a[idx][idx-NX] = -1.0; // 下鄰點
137:     a[idx][idx+NX] = -1.0; // 上鄰點
138:     b[idx] = 2.0 * T_right;
139: }
140:
141: // 內點
142: for(int j = 2; j <= NY-1; j++) {
143:     for(int i = 2; i <= NX-1; i++) {
144:         int idx = (j-1)*NX + i;
145:         a[idx][idx] = 4.0;
146:         a[idx][idx+1] = -1.0; // 右鄰點
147:         a[idx][idx-1] = -1.0; // 左鄰點
148:         a[idx][idx+NX] = -1.0; // 上鄰點
149:         a[idx][idx-NX] = -1.0; // 下鄰點
150:         b[idx] = 0.0; // 內點無熱源
151:     }
152: }
153: }
154:
155: void Jacobi(vector<vector<double> >& a, vector<double>& b,
             vector<double>& x, int n) {
156:     // 先複製當前解到x_old
157:     for(int k = 1; k <= n; k++) {
158:         x_old[k] = x[k];
159:     }
160:

```

```

161: // 計算新的解
162: for(int k = 1; k <= n; k++) {
163:     double sum = 0;
164:     for(int p = 1; p <= n; p++) {
165:         if(p != k) {
166:             sum += a[k][p] * x_old[p];
167:         }
168:     }
169:     x[k] = (b[k] - sum) / a[k][k];
170: }
171:
172: // 計算迭代收斂誤差
173: maxerror = 0;
174: for(int k = 1; k <= n; k++) {
175:     double error = fabs(x[k] - x_old[k]);
176:     if(maxerror < error) {
177:         maxerror = error;
178:     }
179: }
180:
181: // 計算L1誤差 (與解析解比較)
182: double sum = 0;
183: for(int k = 1; k <= n; k++) {
184:     sum += fabs(x[k] - T_analytical(k));
185: }
186: L1sum = sum / double(n);
187: }
188:
189: void output(int m) {
190:     // 將一維解轉換為二維溫度場
191:     for(int j = 1; j <= NY; j++){
192:         for(int i = 1; i <= NX; i++){
193:             T[i-1][j-1] = x[(j-1)*NX + i];
194:         }
195:     }
196:
197:     stringstream name;
198:     name << "steady_diffusion_2D_" << NX << "x" << NY << "_" <<
        setfill('0') << setw(6) << m << ".vtk";
199:     ofstream out(name.str().c_str());
200:
201:     // VTK 文件頭
202:     out << "# vtk DataFile Version 3.0\n";
203:     out << "steady_diffusion_2D\n";
204:     out << "ASCII\n";
205:     out << "DATASET STRUCTURED_POINTS\n";
206:     out << "DIMENSIONS " << NX << " " << NY << " 1\n";
207:     out << "ORIGIN 0.0 0.0 0.0\n";
208:     out << "SPACING " << dx << " " << dy << " 1.0\n";
209:     out << "POINT_DATA " << NX * NY << "\n";
210:
211:     // 輸出溫度場
212:     out << "SCALARS Temperature double 1\n";
213:     out << "LOOKUP_TABLE default\n";

```

```

214: for(int j = 0; j < NY; j++) {
215: for(int i = 0; i < NX; i++) {
216: out << scientific << setprecision(6) << T[i][j] << "\n";
217: }
218: }
219:
220: out.close();
221: cout << "VTK document output: " << name.str() << endl;
222: }
223:
224: // 添加到你的程式碼中的新函數
225:
226: void output_gnuplot_data() {
227:     // 輸出數據檔案 (.dat) //一共四組數據
228:     ofstream data_file("grid_convergence_data.dat");
229:     data_file << "# Grid_Size dx log(dx) L1_Error log(L1_Error)" << endl;
230:
231:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
232:         double dx_value = 1.0 / Nx[grid_idx]; // 對應你的 dx 計算方式
233:         double log_dx = log(dx_value);
234:         double log_error = log(Finall1error[grid_idx]);
235:
236:         data_file << Nx[grid_idx] << "\t"
237:             << scientific << setprecision(6) << dx_value << "\t"
238:             << scientific << setprecision(6) << log_dx << "\t"
239:             << scientific << setprecision(6) <<
Finall1error[grid_idx] << "\t"
240:             << scientific << setprecision(6) << log_error << endl;
241:     }
242:     data_file.close();
243:     cout << "Data file output: grid_convergence_data.dat" << endl;
244:
245:     // 計算線性回歸的斜率和截距
246:     double sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
247:     int n_points = 4;
248:
249:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
250:         double x = log(1.0 / Nx[grid_idx]); // log(dx)
251:         double y = log(Finall1error[grid_idx]); // log(error)
252:
253:         sum_x += x;
254:         sum_y += y;
255:         sum_xy += x * y;
256:         sum_x2 += x * x;
257:     }
258:
259:     double slope = (n_points * sum_xy - sum_x * sum_y) / (n_points *
sum_x2 - sum_x * sum_x);
260:     double intercept = (sum_y - slope * sum_x) / n_points;
261:
262:     // 輸出 gnuplot 腳本
263:     ofstream gnuplot_file("plot_convergence.plt");
264:     gnuplot_file << "# Gnuplot script for grid convergence analysis" <<
endl;

```

```

265:     gnuplot_file << "set terminal png enhanced size 800,600" << endl;
266:     gnuplot_file << "set output 'grid_convergence.png'" << endl;
267:     gnuplot_file << "set title 'Grid Convergence Analysis - L1 Error vs
Grid Spacing'" << endl;
268:     gnuplot_file << "set xlabel 'log(dx)'" << endl;
269:     gnuplot_file << "set ylabel 'log(L1 Error)'" << endl;
270:     gnuplot_file << "set grid" << endl;
271:     gnuplot_file << "set key left top" << endl;
272:     gnuplot_file << "" << endl;
273:
274:     // 理論2階精度線 (斜率=2)
275:     double x_min = log(1.0 / Nx[3]); // 最小的 log(dx) (對應最細網格)
276:     double x_max = log(1.0 / Nx[0]); // 最大的 log(dx) (對應最粗網格)
277:     double y_ref = log(FinalL1error[1]); // 參考點 (使用第二個點)
278:     double x_ref = log(1.0 / Nx[1]);
279:
280:     gnuplot_file << "# 線性回歸線: y = " << slope << " * x + " <<
intercept << endl;
281:     gnuplot_file << "# 理論2階精度線通過參考點 (" << x_ref << ", " <<
y_ref << ")" << endl;
282:     gnuplot_file << "f(x) = " << slope << " * x + " << intercept << endl;
283:     gnuplot_file << "g(x) = 2.0 * (x - " << x_ref << ") + " << y_ref <<
endl;
284:     gnuplot_file << "" << endl;
285:     //gnuplot運行用到plt.數據
286:     gnuplot_file << "plot 'grid_convergence_data.dat' using 3:5 with
linespoints pt 7 ps 1.5 lw 2 title sprintf('Computed (slope = %.2f)', "
<< slope << "), \" << endl;
287:     gnuplot_file << "      f(x) with lines lw 2 lc rgb 'red' title
sprintf('Linear Fit (slope = %.2f)', " << slope << "), \" << endl;
288:     gnuplot_file << "      g(x) with lines lw 2 lc rgb 'green' dashtype 2
title '2nd Order Theory (slope = 2.0)'" << endl;
289:
290:     gnuplot_file.close();
291:     cout << "Gnuplot script output: plot_convergence.plt" << endl;
    //因為 gnuplot 需要實際的數據點來繪製圖形，即使你有斜率，沒有原始數據點就無法
292:    //所以是的，兩個檔案都必須在同一資料夾！
293:
294:     // 輸出結果摘要
295:     cout << "\n=== Grid Convergence Analysis ===" << endl;
296:     cout << "Linear regression results:" << endl;
297:     cout << "Slope = " << fixed << setprecision(3) << slope << "
(理論值應接近 2.0)" << endl;
298:     cout << "Intercept = " << fixed << setprecision(3) << intercept <<
endl;
299:     cout << "Order of accuracy = " << fixed << setprecision(3) << slope
<< endl;
300:
301:     // 計算相關係數
302:     double mean_x = sum_x / n_points;
303:     double mean_y = sum_y / n_points;
304:     double ss_xx = 0, ss_yy = 0, ss_xy = 0;
305:

```

```

306:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
307:         double x = log(1.0 / Nx[grid_idx]);
308:         double y = log(FinalL1error[grid_idx]);
309:         ss_xx += (x - mean_x) * (x - mean_x);
310:         ss_yy += (y - mean_y) * (y - mean_y);
311:         ss_xy += (x - mean_x) * (y - mean_y);
312:     }
313:
314:     double correlation = ss_xy / sqrt(ss_xx * ss_yy);
315:     cout << "Correlation coefficient R = " << fixed << setprecision(4)
<< correlation << endl;
316:     cout << "R2 = " << fixed << setprecision(4) << correlation *
correlation << endl;
317:
318:     cout << "\nTo generate the plot, run: gnuplot plot_convergence.plt"
<< endl;
319:     cout << "===== " << endl;
320: }
321:
322: // 在 main() 函数的最後，在 output2() 之後添加：
323: // output_gnuplot_data();
324:
325:
326: void output3() {
327:     // 設定解析解的網格大小
328:     int analytical_NX = 80;
329:     int analytical_NY = 80;
330:     double analytical_dx = 1.0 / (analytical_NX);
331:     double analytical_dy = 1.0 / (analytical_NY);
332:
333:     ostringstream name;
334:     name << "Analytical_solution_" << analytical_NX << "x" << analytical_NY
<< "_" << setfill('0') << setw(6) << 0 << ".vtk";
335:     ofstream out(name.str().c_str());
336:
337:     // VTK 文件頭
338:     out << "# vtk DataFile Version 3.0\n";
339:     out << "Analytical_solution_" << analytical_NX << "x" << analytical_NY
<< "\n";
340:     out << "ASCII\n";
341:     out << "DATASET STRUCTURED_POINTS\n";
342:     out << "DIMENSIONS " << analytical_NX << " " << analytical_NY << " 1\n";
343:     out << "ORIGIN 0.0 0.0 0.0\n";
344:     out << "SPACING " << analytical_dx << " " << analytical_dy << " 1.0\n";
345:     out << "POINT_DATA " << analytical_NX * analytical_NY << "\n";
346:
347:     // 輸出解析解溫度場
348:     out << "SCALARS Analytical_Temperature double 1\n";
349:     out << "LOOKUP_TABLE default\n";
350:
351:     // 正確計算解析解
352:     for(int j = 0; j < analytical_NY; j++) {
353:         for(int i = 0; i < analytical_NX; i++) {
354:             double x_pos = (i + 0.5) * analytical_dx; // i從0開始，對應x=0到x=1

```

```

355: double y_pos = (j + 0.5) * analytical_dy; // j從0開始，對應y=0到y=1
356: double analytical_temp = T_analytical_fixed(x_pos, y_pos);
357: out << scientific << setprecision(6) << analytical_temp << "\n";
358: }
359: }
360:
361: out.close();
362: cout << "VTK document output: " << name.str() << endl;
363: }
364:
365:
366: int main() {
367:     for(int grid_idx = 0; grid_idx < Nx.size(); grid_idx++) {
368:         cout << "\n===== " << endl;
369:         cout << "Grid size: " << Nx[grid_idx]+1 << "x" << Ny[grid_idx]+1 <<
            endl;
370:
371:         NX = Nx[grid_idx];
372:         NY = Ny[grid_idx];
373:         n = NX * NY;
374:         // 修正網格間距計算 - 對於均勻網格，從0到1分成NX-1個間隔
375:         dx = 1.0 / (NX );
376:         dy = 1.0 / (NY );
377:
378:         cout << "Grid spacing: dx = " << dx << ", dy = " << dy << endl;
379:
380:         // 重新調整向量大小
381:         a.assign(n+2, vector<double>(n+2, 0.0));
382:         b.assign(n+2, 0.0);
383:         x.assign(n+2, 0.0);
384:         x_old.assign(n+2, 0.0);
385:         T.assign(NX, vector<double>(NY, 0.0));
386:
387:         cout << "Program execution started...." << endl;
388:         steadystate = false;
389:         initial(a, b, n);
390:
391:         for(G = 0; G < max_G; G++) {
392:             Jacobi(a, b, x, n);
393:
394:             if(G % 1000 == 0) { // 每1000次輸出一一次
395:                 cout << "Iteration = " << G;
396:                 cout << ", Convergence error = " << scientific << setprecision(3) <<
                    maxerror;
397:                 cout << ", L1 error = " << scientific << setprecision(3) << L1sum <<
                    endl;
398:
399:                 if(G % 5000 == 0) { // 每5000次輸出VTK文件
400:                     output(G);
401:                 }
402:             }
403:
404:             if(G > 100 && maxerror < tolerance) {
405:                 steadystate = true;

```



```

406: cout << "Steady state reached, temperature field converged!!" << endl;
407: cout << "Final iteration: " << G << ", Convergence error: " << maxerror
<< endl;
408: cout << "Final L1 error: " << L1sum << endl;
409: FinalL1error[grid_idx] = L1sum;
410: break;
411: }
412: }
413:
414: if(!steadystate) {
415: cout << "Maximum iteration reached, but steady state not achieved!" <<
endl;
416: cout << "Final convergence error: " << maxerror << endl;
417: cout << "Final L1 error: " << L1sum << endl;
418: FinalL1error[grid_idx] = L1sum;
419: }
420:
421: output(G);
422: cout << "Grid size " << NX << "x" << NY << " computation completed" <<
endl;
423: cout << "=====" << endl;
424: }
425: output_gnuplot_data();
426: output3();
427: cout << "\nAll computations completed!" << endl;
428:
429: return 0;
430: }

```