

```

1: //完全修正版：有限差分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: using namespace std;
15: int nx_data[] = {11, 21, 41, 81};
16: int ny_data[] = {11, 21, 41, 81};
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 A, B, C;
32: double L1sum;
33: double maxerror;
34: const double tolerance = 1e-8;
35: bool steadystate;
36:
37: // 上邊界條件
38: double T_up(int i){
39:     double x_pos = (i-1) * dx;
40:     return 10.0 + sin(pi * x_pos);
41: }
42:
43: double T_analytical_fixed(double x_pos, double y_pos){
44:     return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
45: }
46:
47: double T_analytical(int k){
48:     int i, j;
49:     i = ((k-1) % NX) + 1; // i = [1:NX]
50:     j = ((k-1) / NX) + 1; // j = [1:NY]
51:
52:     double x_pos = (i-1) * dx;
53:     double y_pos = (j-1) * dy;
54:

```

```

55:     return T_analytical_fixed(x_pos, y_pos);
56: }
57:
58: //完全修正的初始化矩陣函數
59: void initial(vector<vector<double> >& a, vector<double>& b, int n) {
60:     // 計算係數
61:     A = 2*((1.0/(dx*dx) + 1.0/(dy*dy)));
62:     B = -1.0/(dx*dx);
63:     C = -1.0/(dy*dy);
64:
65:     // 初始化所有元素為0
66:     for(int i = 0; i <= n+1; i++) {
67:         for(int j = 0; j <= n+1; j++) {
68:             a[i][j] = 0.0;
69:         }
70:         b[i] = 0.0;
71:         x[i] = 0.0;
72:         x_old[i] = 0.0;
73:     }
74:
75:     cout << "Setting boundary conditions..." << endl;
76:
77:     // 步驟1：設置所有邊界點
78:     // 下邊界 (j=1)
79:     for(int i = 1; i <= NX; i++){
80:         int idx = i;
81:         a[idx][idx] = 1.0;
82:         b[idx] = T_bottom;
83:     }
84:
85:     // 上邊界 (j=NY) - 修正：添加右端項設置
86:     for(int i = 1; i <= NX; i++){
87:         int idx = (NY-1)*NX + i;
88:         a[idx][idx] = 1.0;
89:         b[idx] = T_up(i); // <-- 這是關鍵修正！
90:     }
91:
92:     // 左邊界 (i=1, j=2 to NY-1)
93:     for(int j = 2; j <= NY-1; j++){
94:         int idx = (j-1)*NX + 1;
95:         a[idx][idx] = 1.0;
96:         b[idx] = T_left;
97:     }
98:
99:     // 右邊界 (i=NX, j=2 to NY-1)
100:    for(int j = 2; j <= NY-1; j++){
101:        int idx = (j-1)*NX + NX;
102:        a[idx][idx] = 1.0;
103:        b[idx] = T_right;
104:    }
105:
106:    cout << "Setting interior points..." << endl;
107:
108:    // 步驟2：設置所有內點 (i=2 to NX-1, j=2 to NY-1)

```

```

109:     for(int j = 2; j <= NY-1; j++) {
110:         for(int i = 2; i <= NX-1; i++) {
111:             int idx = (j-1)*NX + i;
112:
113:             // 檢查是否已經是邊界點 (應該不會，但安全起見)
114:             if(a[idx][idx] != 0.0) continue;
115:
116:             // 設置五點差分格式
117:             a[idx][idx] = A;
118:             a[idx][idx+1] = B; // 右鄰點 (i+1,j)
119:             a[idx][idx-1] = B; // 左鄰點 (i-1,j)
120:             a[idx][idx+NX] = C; // 上鄰點 (i,j+1)
121:             a[idx][idx-NX] = C; // 下鄰點 (i,j-1)
122:             b[idx] = 0.0; // 無熱源
123:         }
124:     }
125:
126:     cout << "Matrix initialization completed." << endl;
127:     cout << "Total equations: " << n << endl;
128:     cout << "Boundary points: " << 2*NX + 2*(NY-2) << endl;
129:     cout << "Interior points: " << (NX-2)*(NY-2) << endl;
130: }
131:
132: void SOR(vector<vector<double> >& a, vector<double>& b, vector<double>&
x, int n) {
133:     // 先複製當前解到x_old
134:     for(int k = 1; k <= n; k++) {
135:         x_old[k] = x[k];
136:     }
137:
138:     // 計算新的解
139:     for(int k = 1; k <= n; k++) {
140:         double sum = 0;
141:         for(int p = 1; p <= n; p++) {
142:             if(p != k) {
143:                 sum += a[k][p] * x[p];
144:             }
145:         }
146:         x[k] = (b[k] - sum) / a[k][k];
147:         x[k] = x_old[k] + 1.5 * (x[k] - x_old[k]); // SOR
148:     }
149:
150:     // 計算迭代收斂誤差
151:     maxerror = 0;
152:     for(int k = 1; k <= n; k++) {
153:         double error = fabs(x[k] - x_old[k]);
154:         if(maxerror < error) {
155:             maxerror = error;
156:         }
157:     }
158:
159:     // 計算L1誤差 (與解析解比較)
160:     double sum = 0;
161:     for(int k = 1; k <= n; k++) {

```

```

162:         sum += fabs(x[k] - T_analytical(k));
163:     }
164:     L1sum = sum / double(n);
165: }
166:
167: void output(int m) {
168:     // 將一維解轉換為二維溫度場
169:     for(int j = 1; j <= NY; j++){
170:         for(int i = 1; i <= NX; i++){
171:             T[i-1][j-1] = x[(j-1)*NX + i];
172:         }
173:     }
174:
175:     ostringstream name;
176:     name << "FDM_diffusion_2D_" << NX << "x" << NY << "_" <<
    setfill('0') << setw(6) << m << ".vtk";
177:     ofstream out(name.str().c_str());
178:
179:     // VTK 文件頭
180:     out << "# vtk DataFile Version 3.0\n";
181:     out << "steady_diffusion_2D\n";
182:     out << "ASCII\n";
183:     out << "DATASET STRUCTURED_POINTS\n";
184:     out << "DIMENSIONS " << NX << " " << NY << " 1\n";
185:     out << "ORIGIN 0.0 0.0 0.0\n";
186:     out << "SPACING " << dx << " " << dy << " 1.0\n";
187:     out << "POINT_DATA " << NX * NY << "\n";
188:
189:     // 輸出溫度場
190:     out << "SCALARS Temperature double 1\n";
191:     out << "LOOKUP_TABLE default\n";
192:     for(int j = 0; j < NY; j++) {
193:         for(int i = 0; i < NX; i++) {
194:             out << scientific << setprecision(6) << T[i][j] << "\n";
195:         }
196:     }
197:
198:     out.close();
199:     cout << "VTK document output: " << name.str() << endl;
200: }
201:
202: void output_gnuplot_data() {
203:     // 檢查是否有無效的L1誤差
204:     bool valid_data = true;
205:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
206:         if(Finall1error[grid_idx] <= 0 || isnan(Finall1error[grid_idx])
    || isinf(Finall1error[grid_idx])) {
207:             cout << "Warning: Invalid L1 error for grid " << grid_idx <<
    ": " << Finall1error[grid_idx] << endl;
208:             valid_data = false;
209:         }
210:     }
211:
212:     if(!valid_data) {

```

```

213:         cout << "Cannot generate convergence analysis due to invalid
data." << endl;
214:         return;
215:     }
216:
217:     // 輸出數據檔案
218:     ofstream data_file("grid_convergence_data.dat");
219:     data_file << "# Grid_Size dx log(dx) L1_Error log(L1_Error)" << endl;
220:
221:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
222:         double dx_value = 1.0 / (Nx[grid_idx]-1);
223:         double log_dx = log(dx_value);
224:         double log_error = log(FinalL1error[grid_idx]);
225:
226:         data_file << Nx[grid_idx] << "\t"
227:             << scientific << setprecision(6) << dx_value << "\t"
228:             << scientific << setprecision(6) << log_dx << "\t"
229:             << scientific << setprecision(6) <<
FinalL1error[grid_idx] << "\t"
230:             << scientific << setprecision(6) << log_error << endl;
231:     }
232:     data_file.close();
233:     cout << "Data file output: grid_convergence_data.dat" << endl;
234:
235:     // 計算線性回歸
236:     double sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
237:     int n_points = 4;
238:
239:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
240:         double x = log(1.0 / (Nx[grid_idx]-1));
241:         double y = log(FinalL1error[grid_idx]);
242:
243:         sum_x += x;
244:         sum_y += y;
245:         sum_xy += x * y;
246:         sum_x2 += x * x;
247:     }
248:
249:     double slope = (n_points * sum_xy - sum_x * sum_y) / (n_points *
sum_x2 - sum_x * sum_x);
250:     double intercept = (sum_y - slope * sum_x) / n_points;
251:
252:     // 輸出 gnuplot 腳本
253:     ofstream gnuplot_file("plot_convergence.plt");
254:     gnuplot_file << "# Gnuplot script for grid convergence analysis" <<
endl;
255:     gnuplot_file << "set terminal png enhanced size 800,600" << endl;
256:     gnuplot_file << "set output 'grid_convergence.png'" << endl;
257:     gnuplot_file << "set title 'Grid Convergence Analysis - L1 Error vs
Grid Spacing'" << endl;
258:     gnuplot_file << "set xlabel 'log(dx)'" << endl;
259:     gnuplot_file << "set ylabel 'log(L1 Error)'" << endl;
260:     gnuplot_file << "set grid" << endl;
261:     gnuplot_file << "set key left top" << endl;

```

```

262:     gnuplot_file << "" << endl;
263:
264:     // 理論2階精度線 (斜率=2)
265:     double x_min = log(1.0 / (Nx[3]-1)); // 最小的 Log(dx) (對應最細網格)
266:     double x_max = log(1.0 / (Nx[0]-1)); // 最大的 Log(dx) (對應最粗網格)
267:     double y_ref = log(FinalL1error[1]); // 參考點 (使用第二個點)
268:     double x_ref = log(1.0 / (Nx[1]-1));
269:
270:     gnuplot_file << "# 線性回歸線: y = " << slope << " * x + " <<
    intercept << endl;
271:     gnuplot_file << "# 理論2階精度線通過參考點 (" << x_ref << ", " <<
    y_ref << ")" << endl;
272:     gnuplot_file << "f(x) = " << slope << " * x + " << intercept << endl;
273:     gnuplot_file << "g(x) = 2.0 * (x - " << x_ref << ") + " << y_ref <<
    endl;
274:     gnuplot_file << "" << endl;
275:     //gnuplot運行用到plt.數據
276:     gnuplot_file << "plot 'grid_convergence_data.dat' using 3:5 with
    linespoints pt 7 ps 1.5 lw 1 lw 2 title sprintf('Computed (slope = %.2f)', "
    << slope << "), \"\" << endl;
277:     gnuplot_file << "      f(x) with lines lw 2 lc rgb 'red' title
    sprintf('Linear Fit (slope = %.2f)', " << slope << "), \"\" << endl;
278:     gnuplot_file << "      g(x) with lines lw 2 lc rgb 'green' dashtype 2
    title '2nd Order Theory (slope = 2.0)'" << endl;
279:
280:     gnuplot_file.close();
281:     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
282: //所以是的，兩個檔案都必須在同一資料夾！
283:
284:
285:     /// 輸出結果摘要
286:     cout << "\n=== Grid Convergence Analysis ===" << endl;
287:     cout << "Linear regression results:" << endl;
288:     cout << "Slope = " << fixed << setprecision(3) << slope << "
    (理論值應接近 2.0)" << endl;
289:     cout << "Intercept = " << fixed << setprecision(3) << intercept <<
    endl;
290:     cout << "Order of accuracy = " << fixed << setprecision(3) << slope
    << endl;
291:
292:     // 計算相關係數
293:     double mean_x = sum_x / n_points;
294:     double mean_y = sum_y / n_points;
295:     double ss_xx = 0, ss_yy = 0, ss_xy = 0;
296:
297:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
298:         double x = log(1.0 / (Nx[grid_idx]-1));
299:         double y = log(FinalL1error[grid_idx]);
300:         ss_xx += (x - mean_x) * (x - mean_x);
301:         ss_yy += (y - mean_y) * (y - mean_y);
302:         ss_xy += (x - mean_x) * (y - mean_y);
303:     }

```

```

304:
305:     double correlation = ss_xy / sqrt(ss_xx * ss_yy);
306:     cout << "Correlation coefficient R = " << fixed << setprecision(4)
    << correlation << endl;
307:     cout << "R2 = " << fixed << setprecision(4) << correlation *
    correlation << endl;
308:
309:     cout << "\nTo generate the plot, run: gnuplot plot_convergence.plt"
    << endl;
310:     cout << "=====" << endl;
311: }
312: void output3() {
313:     int analytical_NX = 81;
314:     int analytical_NY = 81;
315:     double analytical_dx = 1.0 / (analytical_NX-1);
316:     double analytical_dy = 1.0 / (analytical_NY-1);
317:
318:     ostringstream name;
319:     name << "Analytical_solution_" << analytical_NX << "x" <<
    analytical_NY << "_" << setfill('0') << setw(6) << 0 << ".vtk";
320:     ofstream out(name.str().c_str());
321:
322:     // VTK 文件頭
323:     out << "# vtk DataFile Version 3.0\n";
324:     out << "Analytical_solution_" << analytical_NX << "x" <<
    analytical_NY << "\n";
325:     out << "ASCII\n";
326:     out << "DATASET STRUCTURED_POINTS\n";
327:     out << "DIMENSIONS " << analytical_NX << " " << analytical_NY << "
    1\n";
328:     out << "ORIGIN 0.0 0.0 0.0\n";
329:     out << "SPACING " << analytical_dx << " " << analytical_dy << "
    1.0\n";
330:     out << "POINT_DATA " << analytical_NX * analytical_NY << "\n";
331:
332:     // 輸出解析解溫度場
333:     out << "SCALARS Analytical_Temperature double 1\n";
334:     out << "LOOKUP_TABLE default\n";
335:
336:     for(int j = 0; j < analytical_NY; j++) {
337:         for(int i = 0; i < analytical_NX; i++) {
338:             double x_pos = i * analytical_dx;
339:             double y_pos = j * analytical_dy;
340:             double analytical_temp = T_analytical_fixed(x_pos, y_pos);
341:             out << scientific << setprecision(6) << analytical_temp <<
    "\n";
342:         }
343:     }
344:
345:     out.close();
346:     cout << "VTK document output: " << name.str() << endl;
347: }
348:
349: int main() {

```

```

350:     for(int grid_idx = 0; grid_idx < Nx.size(); grid_idx++) {
351:         cout << "\n===== " << endl;
352:         cout << "Grid size: " << Nx[grid_idx] << "x" << Ny[grid_idx] <<
endl;
353:
354:         NX = Nx[grid_idx];
355:         NY = Ny[grid_idx];
356:         n = NX * NY;
357:         dx = 1.0 / (NX-1);
358:         dy = 1.0 / (NY-1);
359:
360:         cout << "Grid spacing: dx = " << dx << ", dy = " << dy << endl;
361:
362:         // 重新調整向量大小
363:         a.assign(n+2, vector<double>(n+2, 0.0));
364:         b.assign(n+2, 0.0);
365:         x.assign(n+2, 0.0);
366:         x_old.assign(n+2, 0.0);
367:         T.assign(NX, vector<double>(NY, 0.0));
368:
369:         cout << "Program execution started..." << endl;
370:         steadystate = false;
371:         initial(a, b, n);
372:
373:         for(G = 0; G < max_G; G++) {
374:             SOR(a, b, x, n);
375:
376:             if(G % 1000 == 0) {
377:                 cout << "Iteration = " << G;
378:                 cout << ", Convergence error = " << scientific <<
setprecision(3) << maxerror;
379:                 cout << ", L1 error = " << scientific << setprecision(3)
<< L1sum << endl;
380:
381:                 if(G % 5000 == 0) {
382:                     output(G);
383:                 }
384:             }
385:
386:             if(G > 100 && maxerror < tolerance) {
387:                 steadystate = true;
388:                 cout << "Steady state reached!" << endl;
389:                 cout << "Final iteration: " << G << ", Convergence
error: " << maxerror << endl;
390:                 cout << "Final L1 error: " << L1sum << endl;
391:                 Finall1error[grid_idx] = L1sum;
392:                 break;
393:             }
394:         }
395:
396:         if(!steadystate) {
397:             cout << "Maximum iteration reached!" << endl;
398:             cout << "Final convergence error: " << maxerror << endl;
399:             cout << "Final L1 error: " << L1sum << endl;

```



```

400:         FinalL1error[grid_idx] = L1sum;
401:     }
402:
403:     output(G);
404:     cout << "Grid size " << NX << "x" << NY << " computation
completed" << endl;
405:     cout << "=====" << endl;
406: }
407:
408: output_gnuplot_data();
409: output3();
410: cout << "\nAll computations completed!" << endl;
411:
412: return 0;
413: }

```