

```

1: //四階精度差分格式求解Laplace方程
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: int nx_data[] = {11, 21, 41, 81};
15: int ny_data[] = {11, 21, 41, 81};
16:
17: vector<int> Nx(nx_data, nx_data + sizeof(nx_data)/sizeof(nx_data[0]));
18: vector<int> Ny(ny_data, ny_data + sizeof(ny_data)/sizeof(ny_data[0]));
19: double FinalL1error[4];
20: int n, NX, NY;
21: double dx, dy;
22: vector<vector<double> > > a;
23: vector<double> b;
24: vector<double> x, x_old;
25: vector<vector<double> > > T;
26: int G, max_G = 100000;
27: double T_left = 10.0;
28: double T_right = 10.0;
29: double T_bottom = 10.0;
30: double L1sum;
31: double maxerror;
32: const double tolerance = 1e-10; // 迭代收斂判據
33: bool steadystate;
34:
35: // 上邊界邊界條件
36: double T_up(int i){
37:     double x_pos = (i-1) * dx;
38:     return 10.0 + sin(pi * x_pos);
39: }
40:
41: // 解析解
42: double T_analytical_fixed(double x_pos, double y_pos){
43:     return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
44: }
45:
46: double T_analytical(int k){
47:     int i = ((k-1) % NX) + 1;
48:     int j = ((k-1) / NX) + 1;
49:
50:     double x_pos = (i-1) * dx;
51:     double y_pos = (j-1) * dy;
52:
53:     return T_analytical_fixed(x_pos, y_pos);
54: }

```

```

55:
56: //初始化迭代矩陣
57: void initial(vector<vector<double> >& a, vector<double>& b, int n) {
58:
59:     double A = 1.0 / (12.0 * dx * dx);
60:     double B = 1.0 / (12.0 * dy * dy);
61:
62:     //二階偏導數的四階精度中心差分的係數
63:     double c_center = 30.0; // 中心係數
64:     double c_near = -16.0; // 相鄰係數
65:     double c_far = 1.0; // 遠方係數
66:
67:     // 初始化矩陣
68:     for(int i = 0; i <= n+1; i++) {
69:         for(int j = 0; j <= n+1; j++) {
70:             a[i][j] = 0.0;
71:         }
72:         b[i] = 0.0;
73:         x[i] = 0.0;
74:         x_old[i] = 0.0;
75:     }
76:
77:     cout << "Setting boundary conditions..." << endl;
78:
79:     // 左邊界條件 - 直接賦值
80:     // 左邊界 (i=1)
81:     for(int j = 1; j <= NY; j++){
82:         int idx = (j-1)*NX + 1;
83:         a[idx][idx] = 1.0;
84:         b[idx] = T_left;
85:     }
86:
87:     // 右邊界 (i=NX)
88:     for(int j = 1; j <= NY; j++){
89:         int idx = (j-1)*NX + NX;
90:         a[idx][idx] = 1.0;
91:         b[idx] = T_right;
92:     }
93:
94:     // 下邊界 (j=1)
95:     for(int i = 1; i <= NX; i++){
96:         int idx = i;
97:         a[idx][idx] = 1.0;
98:         b[idx] = T_bottom;
99:     }
100:
101:     // 上邊界 (j=NY)
102:     for(int i = 1; i <= NX; i++){
103:         int idx = (NY-1)*NX + i;
104:         a[idx][idx] = 1.0;
105:         b[idx] = T_up(i);
106:     }
107:
108:     cout << "Setting interior points with consistent 4th order
scheme..." << endl;

```

```

109:
110: // ?部點 - 使用精確四階差分格式
111: // 只有真正的內部點 (i=3 to NX-2, j=3 to NY-2) 使用四階格式
112: for(int j = 3; j <= NY-2; j++) {
113:     for(int i = 3; i <= NX-2; i++) {
114:         int idx = (j-1)*NX + i;
115:
116:         // 拉普拉斯算子: (d2/dx2 + d2/dy2)u = 0
117:         // x方向: u_{i-2} - 16*u_{i-1} + 30*u_i - 16*u_{i+1} +
u_{i+2}
118:         // y方向: u_{j-2} - 16*u_{j-1} + 30*u_j - 16*u_{j+1} +
u_{j+2}
119:
120:         a[idx][idx] = A * c_center + B * c_center; // 中心點
121:         a[idx][idx-1] = A * c_near; // 西點
122:         a[idx][idx+1] = A * c_near; // 東點
123:         a[idx][idx-2] = A * c_far; // 西西點
124:         a[idx][idx+2] = A * c_far; // 東東點
125:         a[idx][idx-NX] = B * c_near; // 南點
126:         a[idx][idx+NX] = B * c_near; // 北點
127:         a[idx][idx-2*NX] = B * c_far; // 南南點
128:         a[idx][idx+2*NX] = B * c_far; // 北北點
129:
130:         b[idx] = 0.0;
131:     }
132: }
133:
134: //
135: //第二層邊界點 (i=2, i=NX-1, j=2, j=NY-1) 使用二階中心差分
136: // 這些點無法使用四階格式，因為缺少足夠的鄰近點
137: for(int j = 2; j <= NY-1; j++) {
138:     for(int i = 2; i <= NX-1; i++) {
139:         // 跳過已處理的內部點
140:         if(i >= 3 && i <= NX-2 && j >= 3 && j <= NY-2) continue;
141:
142:         int idx = (j-1)*NX + i;
143:
144:         // 使用二階中心差分
145:         double A2 = 1.0 / (dx * dx);
146:         double B2 = 1.0 / (dy * dy);
147:
148:         a[idx][idx] = -2.0 * A2 - 2.0 * B2;
149:         a[idx][idx-1] = A2; // 西點
150:         a[idx][idx+1] = A2; // 東點
151:         a[idx][idx-NX] = B2; // 南點
152:         a[idx][idx+NX] = B2; // 北點
153:         b[idx] = 0.0;
154:     }
155: }
156:
157: cout << "Matrix initialization completed." << endl;
158: cout << "Total equations: " << n << endl;
159: cout << "Interior 4th-order points: " << (NX-4)*(NY-4) << endl;
160: }

```

```

161:
162: //超鬆弛迭代法_自適應性鬆弛因子
163: void SOR(vector<vector<double> >& a, vector<double>& b, vector<double>&
    x, int n) {
164:     // 自適應鬆弛因子
165:     double omega;
166:     if(NX <= 21) omega = 0.5;
167:     else if(NX <= 41) omega = 0.8;
168:     else omega = 1.2;
169:
170:     //保存舊的解
171:     for(int k = 1; k <= n; k++) {
172:         x_old[k] = x[k];
173:     }
174:
175:     // SOR迭代
176:     for(int k = 1; k <= n; k++) {
177:         if(fabs(a[k][k]) < 1e-15) continue; // 跳過奇異矩陣
178:
179:         double sum = 0;
180:         for(int p = 1; p <= n; p++) {
181:             if(p != k) {
182:                 sum += a[k][p] * x[p];
183:             }
184:         }
185:         double x_new = (b[k] - sum) / a[k][k];
186:         x[k] = x_old[k] + omega * (x_new - x_old[k]);
187:     }
188:
189:     // 計算最大收斂誤差
190:     maxerror = 0;
191:     for(int k = 1; k <= n; k++) {
192:         double error = fabs(x[k] - x_old[k]);
193:         if(maxerror < error) {
194:             maxerror = error;
195:         }
196:     }
197:
198:     // 計算L1誤差
199:     double sum = 0;
200:     for(int k = 1; k <= n; k++) {
201:         sum += fabs(x[k] - T_analytical(k));
202:     }
203:     L1sum = sum / double(n);
204: }
205:
206:
207: void output(int m) {
208:     for(int j = 1; j <= NY; j++){
209:         for(int i = 1; i <= NX; i++){
210:             T[i-1][j-1] = x[(j-1)*NX + i];
211:         }
212:     }
213:

```

```

214:     ostream name;
215:     name << "FDM_diffusion_2D_improved_" << NX << "x" << NY << "_" <<
    setfill('0') << setw(6) << m << ".vtk";
216:     ofstream out(name.str().c_str());
217:
218:     out << "# vtk DataFile Version 3.0\n";
219:     out << "steady_diffusion_2D_improved\n";
220:     out << "ASCII\n";
221:     out << "DATASET STRUCTURED_POINTS\n";
222:     out << "DIMENSIONS " << NX << " " << NY << " 1\n";
223:     out << "ORIGIN 0.0 0.0 0.0\n";
224:     out << "SPACING " << dx << " " << dy << " 1.0\n";
225:     out << "POINT_DATA " << NX * NY << "\n";
226:
227:     out << "SCALARS Temperature double 1\n";
228:     out << "LOOKUP_TABLE default\n";
229:     for(int j = 0; j < NY; j++) {
230:         for(int i = 0; i < NX; i++) {
231:             out << scientific << setprecision(6) << T[i][j] << "\n";
232:         }
233:     }
234:
235:     out.close();
236:     cout << "VTK document output: " << name.str() << endl;
237: }
238:
239: void output_gnuplot_data() {
240:     bool valid_data = true;
241:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
242:         if(Finall1error[grid_idx] <= 0 || isnan(Finall1error[grid_idx])
|| isinf(Finall1error[grid_idx])) {
243:             cout << "Warning: Invalid L1 error for grid " << grid_idx <<
": " << Finall1error[grid_idx] << endl;
244:             valid_data = false;
245:         }
246:     }
247:
248:     if(!valid_data) {
249:         cout << "Cannot generate convergence analysis due to invalid
data." << endl;
250:         return;
251:     }
252:
253:     ofstream data_file("grid_convergence_4order_improved.dat");
254:     data_file << "# Grid_Size dx log(dx) L1_Error log(L1_Error)" << endl;
255:
256:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
257:         double dx_value = 1.0 / (Nx[grid_idx]-1);
258:         double log_dx = log(dx_value);
259:         double log_error = log(Finall1error[grid_idx]);
260:
261:         data_file << Nx[grid_idx] << "\t"
262:             << scientific << setprecision(6) << dx_value << "\t"
263:             << scientific << setprecision(6) << log_dx << "\t"

```

```

264:             << scientific << setprecision(6) <<
    FinalL1error[grid_idx] << "\t"
265:             << scientific << setprecision(6) << log_error << endl;
266:     }
267:     data_file.close();
268:     cout << "Data file output: grid_convergence_4order_improved.dat" <<
endl;
269:
270:     // ?性回?
271:     double sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
272:     int n_points = 4;
273:
274:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
275:         double x = log(1.0 / (Nx[grid_idx]-1));
276:         double y = log(FinalL1error[grid_idx]);
277:
278:         sum_x += x;
279:         sum_y += y;
280:         sum_xy += x * y;
281:         sum_x2 += x * x;
282:     }
283:
284:     double slope = (n_points * sum_xy - sum_x * sum_y) / (n_points *
sum_x2 - sum_x * sum_x);
285:     double intercept = (sum_y - slope * sum_x) / n_points;
286:
287:     ofstream gnuplot_file("plot_convergence_4order_improved.plt");
288:     gnuplot_file << "set terminal png enhanced size 800,600" << endl;
289:     gnuplot_file << "set output 'grid_convergence_4order_improved.png'"
<< endl;
290:     gnuplot_file << "set title 'Improved Grid Convergence Analysis: L1
Error vs Grid Spacing'" << endl;
291:     gnuplot_file << "set xlabel 'log(dx)'" << endl;
292:     gnuplot_file << "set ylabel 'log(L1 Error)'" << endl;
293:     gnuplot_file << "set grid" << endl;
294:     gnuplot_file << "set key left top" << endl;
295:
296:     double x_min = log(1.0 / (Nx[3]-1));
297:     double x_max = log(1.0 / (Nx[0]-1));
298:     double y_ref = log(FinalL1error[1]);
299:     double x_ref = log(1.0 / (Nx[1]-1));
300:
301:     gnuplot_file << "f(x) = " << slope << " * x + " << intercept << endl;
302:     gnuplot_file << "g(x) = 4.0 * (x - " << x_ref << ") + " << y_ref <<
endl;
303:
304:     gnuplot_file << "plot 'grid_convergence_4order_improved.dat' using
3:5 with linespoints pt 7 ps 1.5 lw 2 title sprintf('Improved (slope =
%.2f)', " << slope << ")", "\\ " << endl;
305:     gnuplot_file << "      f(x) with lines lw 2 lc rgb 'red' title
sprintf('Linear Fit (slope = %.2f)', " << slope << ")", "\\ " << endl;
306:     gnuplot_file << "      g(x) with lines lw 2 lc rgb 'green' dashtype 2
title '4th Order Theory (slope = 4.0)'" << endl;
307:

```

```

308:     gnuplot_file.close();
309:     cout << "Gnuplot script output:
plot_convergence_4order_improved.plt" << endl;
310:
311:     cout << "\n=== Improved Grid Convergence Analysis ===" << endl;
312:     cout << "Linear regression results:" << endl;
313:     cout << "Slope = " << fixed << setprecision(3) << slope << " (理?值
4.0)" << endl;
314:     cout << "Order of accuracy = " << fixed << setprecision(3) << slope
<< endl;
315: }
316:
317: int main() {
318:     for(int grid_idx = 0; grid_idx < Nx.size(); grid_idx++) {
319:         cout << "\n===== " << endl;
320:         cout << "Grid size: " << Nx[grid_idx] << "x" << Ny[grid_idx] <<
endl;
321:
322:         NX = Nx[grid_idx];
323:         NY = Ny[grid_idx];
324:         n = NX * NY;
325:         dx = 1.0 / (NX-1);
326:         dy = 1.0 / (NY-1);
327:
328:         cout << "Grid spacing: dx = " << dx << ", dy = " << dy << endl;
329:
330:         a.assign(n+2, vector<double>(n+2, 0.0));
331:         b.assign(n+2, 0.0);
332:         x.assign(n+2, 0.0);
333:         x_old.assign(n+2, 0.0);
334:         T.assign(NX, vector<double>(NY, 0.0));
335:
336:         cout << "Program execution started with improved 4th order
scheme..." << endl;
337:         steadystate = false;
338:         initial(a, b, n); //初始化
339:
340:         for(G = 0; G < max_G; G++) {
341:             SOR(a, b, x, n); // 使用改進的SOR
342:
343:             if(G % 1000 == 0) {
344:                 cout << "Iteration = " << G;
345:                 cout << ", Convergence error = " << scientific <<
setprecision(3) << maxerror;
346:                 cout << ", L1 error = " << scientific << setprecision(3)
<< L1sum << endl;
347:
348:                 if(G % 5000 == 0) {
349:                     output(G);
350:                 }
351:             }
352:
353:             if(G > 100 && maxerror < tolerance) {
354:                 steadystate = true;

```

```

355:             cout << "Steady state reached!" << endl;
356:             cout << "Final iteration: " << G << ", Final convergence
error : " << maxerror << endl;
357:             cout << "Final L1 error: " << L1sum << endl;
358:             Finall1error[grid_idx] = L1sum;
359:             break;
360:         }
361:     }
362:
363:     if(!steadystate) {
364:         cout << "Maximum iteration reached!" << endl;
365:         cout << "Final convergence error: " << maxerror << endl;
366:         cout << "Final L1 error: " << L1sum << endl;
367:         Finall1error[grid_idx] = L1sum;
368:     }
369:
370:     output(G);
371:     cout << "Grid size " << NX << "x" << NY << " computation
completed" << endl;
372:     cout << "===== " << endl;
373: }
374:
375: output_gnuplot_data();
376: cout << "\nAll computations completed!" << endl;
377:
378: return 0;
379: }

```