

```

1: //採用標準四階九點方程求解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: // 解析解
36: double T_analytical_fixed(double x_pos, double y_pos){
37:     return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
38: }
39:
40: double T_analytical(int k){
41:     int i = ((k-1) % NX) + 1;
42:     int j = ((k-1) / NX) + 1;
43:
44:     double x_pos = (i-1) * dx;
45:     double y_pos = (j-1) * dy;
46:
47:     return T_analytical_fixed(x_pos, y_pos);
48: }
49:
50: // 上邊界邊界條件
51: double T_up(int i){
52:     double x_pos = (i-1) * dx;
53:     return 10.0 + sin(pi * x_pos);
54: }

```

```

55:
56:
57: //初始化迭代矩陣
58: void initial(vector<vector<double> >& a, vector<double>& b, int n) {
59:
60:     double H = 1.0 / (6*dx*dx); //用於九點方程
61:     double a1 = -20 ; //本點係數採用a1*H
62:     double a2 = 4.0 ; //直角方向臨計算點a2*H
63:     double a3 = 1.0 ; //斜角方向臨計算點a3*H
64:
65:
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 6th-order 9 points
difference scheme ..." << endl;

```

```

109:   for(int j = 3; j <= NY-2; j++) {
110:       for(int i = 3; i <= NX-2; i++) {
111:           int idx = (j-1)*NX + i;
112:           a[idx][idx] = a1*H ;
113:           a[idx][idx+1] = a2*H ;//(1)右
114:           a[idx][idx+NX] = a2*H ;//(2)上
115:           a[idx][idx-1] = a2*H ;//(3)左
116:           a[idx][idx-NX] = a2*H ;//(4)下
117:           a[idx][idx+1+NX] = a3*H ;//(5)右上
118:           a[idx][idx-1+NX] = a3*H ;//(6)左上
119:           a[idx][idx-1-NX] = a3*H ;//(7)左下
120:           a[idx][idx+1-NX] = a3*H ;//(8)右下
121:           b[idx] = 0.0;
122:       }
123:   }
124:   //各個邊界點採用空間二階精度中心差分格式
125:   double A = -2.0 * ((1.0/(dx*dx))+(1.0/(dy*dy))) ;
126:   double B = (1.0/(dx*dx)) ;
127:   double C = (1.0/(dy*dy)) ;
128:   //左邊界計算點
129:   //處理方式:東西方向採用單邊插分，南北方向採用中心差分
130:   for(int j = 3 ; j <= NY-2 ; j++){
131:       int idx = (j-1)*NX + 2 ; //i = 2 ;
132:       b[idx] = 0.0 ;
133:       a[idx][idx] = A ;
134:       a[idx][idx-1] = B ;
135:       a[idx][idx+1] = B ;
136:       a[idx][idx-NX] = C ;
137:       a[idx][idx+NX] = C ;
138:
139:   }
140:   //右邊界計算點
141:   //處理方式:東西方向採用單邊插分，南北方向採用中心差分
142:   for(int j = 3 ; j <= NY-2 ; j++){
143:       int idx = (j-1)*NX + (NX-1) ; // i = (NX-1)
144:       a[idx][idx] = A ;
145:       a[idx][idx-1] = B ;
146:       a[idx][idx+1] = B ;
147:       a[idx][idx-NX] = C ;
148:       a[idx][idx+NX] = C ;
149:   }
150:   //下邊界計算點
151:   //處理方式:南北方向採用單邊差分，東西方向採用中心差分
152:   for(int i = 3 ; i <= NX-2 ; i++){
153:       int idx = (2-1)*NX + i ; // j = 2
154:       b[idx] = 0.0 ;
155:       a[idx][idx] = A ;
156:       a[idx][idx-1] = B ;
157:       a[idx][idx+1] = B ;
158:       a[idx][idx-NX] = C ;
159:       a[idx][idx+NX] = C ;
160:   }
161:   //上邊界計算點
162:   //處理方式:南北方向採用單邊差分，東西方向採用中心差分

```

```

163:     for(int i = 3 ; i <= NX-2 ; i++){
164:         int idx = (NY-2)*NX + i ; // j = (NY-2)
165:         b[idx] = 0.0 ;
166:         a[idx][idx] = A ;
167:         a[idx][idx-1] = B ;
168:         a[idx][idx+1] = B ;
169:         a[idx][idx-NX] = C ;
170:         a[idx][idx+NX] = C ;
171:     }
172:
173:     //左下角點
174:     int p1 = (2-1)*NX + 2 ;
175:     b[p1] = 0.0 ;
176:     a[p1][p1] = A ;
177:     a[p1][p1-1] = B ;
178:     a[p1][p1+1] = B ;
179:     a[p1][p1-NX] = C ;
180:     a[p1][p1+NX] = C ;
181:     //右下角點
182:     int p2 = (2-1)*NX + (NX-1) ;
183:     b[p2] = 0.0 ;
184:     a[p2][p2] = A ;
185:     a[p2][p2-1] = B ;
186:     a[p2][p2+1] = B ;
187:     a[p2][p2-NX] = C ;
188:     a[p2][p2+NX] = C ;
189:     //左上角點
190:     int p3 = (NY-2)*NX + 2 ;
191:     b[p3] = 0.0 ;
192:     a[p3][p3] = A ; // 修正：應該是 a[p3][p3] 而不是 a[p3][p3-1]
193:     a[p3][p3+1] = B ;
194:     a[p3][p3-1] = B ; // 這行是正確的
195:     a[p3][p3-NX] = C ;
196:     a[p3][p3+NX] = C ;
197:     int p4 = (NY-2)*NX + (NX-1) ;
198:     b[p4] = 0.0 ;
199:     a[p4][p4] = A ;
200:     a[p4][p4-1] = B ;
201:     a[p4][p4+1] = B ;
202:     a[p4][p4-NX] = C ;
203:     a[p4][p4+NX] = C ;
204:     cout << "Matrix initialization completed." << endl ;
205:     cout << "Total equations: " << n << endl ;
206:     cout << "for 9 points difference scheme(2-order bound) ,Interior
points: " << (NX-4)*(NY-4) << endl ;
207: }
208:
209: //超鬆弛迭代法_自適應性鬆弛因子
210: void SOR(vector<vector<double> >& a, vector<double>& b, vector<double>&
x, int n) {
211:     // 自適應鬆弛因子
212:     double omega ;
213:     if(NX <= 21) omega = 0.5 ;
214:     else if(NX <= 41) omega = 0.8 ;

```

```

215:     else omega = 1.2;
216:
217:     //保存舊的解
218:     for(int k = 1; k <= n; k++) {
219:         x_old[k] = x[k];
220:     }
221:
222:     // SOR迭代
223:     for(int k = 1; k <= n; k++) {
224:         if(fabs(a[k][k]) < 1e-15) continue; // 跳過奇異矩陣
225:
226:         double sum = 0;
227:         for(int p = 1; p <= n; p++) {
228:             if(p != k) {
229:                 sum += a[k][p] * x[p];
230:             }
231:         }
232:         double x_new = (b[k] - sum) / a[k][k];
233:         x[k] = x_old[k] + omega * (x_new - x_old[k]);
234:     }
235:
236:     // 計算最大收斂誤差
237:     maxerror = 0;
238:     for(int k = 1; k <= n; k++) {
239:         double error = fabs(x[k] - x_old[k]);
240:         if(maxerror < error) {
241:             maxerror = error;
242:         }
243:     }
244:
245:     // 計算L1誤差
246:     double sum = 0;
247:     for(int k = 1; k <= n; k++) {
248:         sum += fabs(x[k] - T_analytical(k));
249:     }
250:     L1sum = sum / double(n);
251: }
252:
253:
254: void output(int m) {
255:     for(int j = 1; j <= NY; j++){
256:         for(int i = 1; i <= NX; i++){
257:             T[i-1][j-1] = x[(j-1)*NX + i];
258:         }
259:     }
260:
261:     ostringstream name;
262:     name << "9 points difference shceme(2-order bound) " << NX << "x" <<
NY << "_" << setfill('0') << setw(6) << m << ".vtk";
263:     ofstream out(name.str().c_str());
264:
265:     out << "# vtk DataFile Version 3.0\n";
266:     out << "9 points difference shceme(2-order bound)_\n";
267:     out << "ASCII\n";

```

```

268: out << "DATASET STRUCTURED_POINTS\n";
269: out << "DIMENSIONS " << NX << " " << NY << " 1\n";
270: out << "ORIGIN 0.0 0.0 0.0\n";
271: out << "SPACING " << dx << " " << dy << " 1.0\n";
272: out << "POINT_DATA " << NX * NY << "\n";
273:
274: out << "SCALARS Temperature double 1\n";
275: out << "LOOKUP_TABLE default\n";
276: for(int j = 0; j < NY; j++) {
277:     for(int i = 0; i < NX; i++) {
278:         out << scientific << setprecision(6) << T[i][j] << "\n";
279:     }
280: }
281:
282: out.close();
283: cout << "VTK document output: " << name.str() << endl;
284: }
285:
286: void output_gnuplot_data() {
287:     bool valid_data = true;
288:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
289:         if(Finall1error[grid_idx] <= 0 || isnan(Finall1error[grid_idx])
|| isinf(Finall1error[grid_idx])) {
290:             cout << "Warning: Invalid L1 error for grid " << grid_idx <<
": " << Finall1error[grid_idx] << endl;
291:             valid_data = false;
292:         }
293:     }
294:
295:     if(!valid_data) {
296:         cout << "Cannot generate convergence analysis due to invalid
data." << endl;
297:         return;
298:     }
299:
300:     ofstream data_file("grid_convergence_9 points difference shceme(2-
order bound).dat");
301:     data_file << "# Grid_Size dx log(dx) L1_Error log(L1_Error)" << endl;
302:
303:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
304:         double dx_value = 1.0 / (Nx[grid_idx]-1);
305:         double log_dx = log(dx_value);
306:         double log_error = log(Finall1error[grid_idx]);
307:
308:         data_file << Nx[grid_idx] << "\t"
309:             << scientific << setprecision(6) << dx_value << "\t"
310:             << scientific << setprecision(6) << log_dx << "\t"
311:             << scientific << setprecision(6) <<
Finall1error[grid_idx] << "\t"
312:             << scientific << setprecision(6) << log_error << endl;
313:     }
314:     data_file.close();
315:     cout << "Data file output: grid_convergence_9 points difference
shceme(2-order bound).dat" << endl;

```

```

316:
317:     double sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
318:     int n_points = 4;
319:
320:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
321:         double x = log(1.0 / (Nx[grid_idx]-1));
322:         double y = log(FinalL1error[grid_idx]);
323:
324:         sum_x += x;
325:         sum_y += y;
326:         sum_xy += x * y;
327:         sum_x2 += x * x;
328:     }
329:
330:     double slope = (n_points * sum_xy - sum_x * sum_y) / (n_points *
sum_x2 - sum_x * sum_x);
331:     double intercept = (sum_y - slope * sum_x) / n_points;
332:
333:     ofstream gnuplot_file("plot_convergence_9 points difference shceme(2-
order bound).plt");
334:     gnuplot_file << "set terminal png enhanced size 800,600" << endl;
335:     gnuplot_file << "set output 'grid_convergence_9 points difference
shceme(2-order bound).png'" << endl;
336:     gnuplot_file << "set title 'Improved Grid Convergence Analysis: L1
Error vs Grid Spacing'" << endl;
337:     gnuplot_file << "set xlabel 'log(dx)'" << endl;
338:     gnuplot_file << "set ylabel 'log(L1 Error)'" << endl;
339:     gnuplot_file << "set grid" << endl;
340:     gnuplot_file << "set key left top" << endl;
341:
342:     double x_min = log(1.0 / (Nx[3]-1));
343:     double x_max = log(1.0 / (Nx[0]-1));
344:     double y_ref = log(FinalL1error[1]);
345:     double x_ref = log(1.0 / (Nx[1]-1));
346:
347:     gnuplot_file << "f(x) = " << slope << " * x + " << intercept << endl;
348:     gnuplot_file << "g(x) = 4.0 * (x - " << x_ref << ") + " << y_ref <<
endl;
349:
350:     gnuplot_file << "plot 'grid_convergence_9 points difference shceme(2-
order bound).dat' using 3:5 with linespoints pt 7 ps 1.5 lw 2 title
sprintf('Improved (slope = %.2f)', " << slope << "), \" << endl;
351:     gnuplot_file << "      f(x) with lines lw 2 lc rgb 'red' title
sprintf('Linear Fit (slope = %.2f)', " << slope << "), \" << endl;
352:     gnuplot_file << "      g(x) with lines lw 2 lc rgb 'green' dashtype 2
title '9 points difference shceme(2-order bound)(slope = 4.0)'" << endl;
353:
354:     gnuplot_file.close();
355:     cout << "Gnuplot script output: plot_convergence_9 points difference
shceme(2-order bound).plt" << endl;
356:
357:     cout << "\n=== Improved Grid Convergence Analysis ===" << endl;
358:     cout << "Linear regression results:" << endl;
359:     cout << "Slope = " << fixed << setprecision(3) << slope << "
(theoretical 4.0)" << endl;

```

```

360:     cout << "Order of accuracy = " << fixed << setprecision(3) << slope
    << endl;
361: }
362:
363: int main() {
364:     for(int grid_idx = 0; grid_idx < Nx.size(); grid_idx++) {
365:         cout << "\n===== " << endl;
366:         cout << "Grid size: " << Nx[grid_idx] << "x" << Ny[grid_idx] <<
endl;
367:
368:         NX = Nx[grid_idx];
369:         NY = Ny[grid_idx];
370:         n = NX * NY;
371:         dx = 1.0 / (NX-1);
372:         dy = 1.0 / (NY-1);
373:
374:         cout << "Grid spacing: dx = " << dx << ", dy = " << dy << endl;
375:
376:         a.assign(n+2, vector<double>(n+2, 0.0));
377:         b.assign(n+2, 0.0);
378:         x.assign(n+2, 0.0);
379:         x_old.assign(n+2, 0.0);
380:         T.assign(NX, vector<double>(NY, 0.0));
381:
382:         cout << "Program execution started with 9 points difference
shceme(2-order bound) ...." << endl;
383:         steadystate = false;
384:         initial(a, b, n); //初始化
385:
386:         for(G = 0; G < max_G; G++) {
387:             SOR(a, b, x, n); // 使用改進的SOR
388:
389:             if(G % 1000 == 0) {
390:                 cout << "Iteration = " << G;
391:                 cout << ", Convergence error = " << scientific <<
setprecision(3) << maxerror;
392:                 cout << ", L1 error = " << scientific << setprecision(3)
<< L1sum << endl;
393:
394:                 if(G % 5000 == 0) {
395:                     output(G);
396:                 }
397:             }
398:
399:             if(G > 100 && maxerror < tolerance) {
400:                 steadystate = true;
401:                 cout << "Steady state reached!" << endl;
402:                 cout << "Final iteration: " << G << ", Final convergence
error : " << maxerror << endl;
403:                 cout << "Final L1 error: " << L1sum << endl;
404:                 Finall1error[grid_idx] = L1sum;
405:                 break;
406:             }
407:         }

```



```

408:
409:     if(!steadystate) {
410:         cout << "Maximum iteration reached!" << endl;
411:         cout << "Final convergence error: " << maxerror << endl;
412:         cout << "Final L1 error: " << L1sum << endl;
413:         FinalL1error[grid_idx] = L1sum;
414:     }
415:
416:     output(G);
417:     cout << "Grid size " << NX << "x" << NY << " computation
completed" << endl;
418:     cout << "===== " << endl;
419: }
420:
421: output_gnuplot_data();
422: cout << "\nAll computations completed!" << endl;
423:
424: return 0;
425: }

```