

```

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: double cfd[6] = {(10.0/12.0) , (-15.0/12.0) , (-4.0 / 12.0) , (14.0 /
12.0) , (-6.0 / 12.0) , (1.0/12.0) } ;
36: //四階精度差分係數可依序排列做前向差分與後向差分
37:
38: // 解析解
39: double T_analytical_fixed(double x_pos, double y_pos){
40:     return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
41: }
42:
43: double T_analytical(int k){
44:     int i = ((k-1) % NX) + 1;
45:     int j = ((k-1) / NX) + 1;
46:
47:     double x_pos = (i-1) * dx;
48:     double y_pos = (j-1) * dy;
49:
50:     return T_analytical_fixed(x_pos, y_pos);
51: }
52:
53: // 上邊界邊界條件

```

```

54: double T_up(int i){
55:     double x_pos = (i-1) * dx;
56:     return 10.0 + sin(pi * x_pos);
57: }
58:
59:
60: //初始化迭代矩陣
61: void initial(vector<vector<double> >& a, vector<double>& b, int n) {
62:
63:     double H = 1.0 / (6*dx*dx); //用於九點方程
64:     double H2 = 1.0/ (dx*dx) ; //四階前向差分或四階精度後項差分
65:     //1/(6*dx*dx) = 1/(6*dy*dy)
66:     double a1 = -20 ; //本點係數採用a1*H
67:     double a2 = 4.0 ; //直角方向臨計算點a2*H
68:     double a3 = 1.0 ; //斜角方向臨計算點a3*H
69:
70:
71:
72:     // 初始化矩陣
73:     for(int i = 0; i <= n+1; i++) {
74:         for(int j = 0; j <= n+1; j++) {
75:             a[i][j] = 0.0;
76:         }
77:         b[i] = 0.0;
78:         x[i] = 0.0;
79:         x_old[i] = 0.0;
80:     }
81:
82:     cout << "Setting boundary conditions..." << endl;
83:
84:     // 左邊界條件 - 直接賦值
85:     // 左邊界 (i=1)
86:     for(int j = 1; j <= NY; j++){
87:         int idx = (j-1)*NX + 1;
88:         a[idx][idx] = 1.0;
89:         b[idx] = T_left;
90:     }
91:
92:     // 右邊界 (i=NX)
93:     for(int j = 1; j <= NY; j++){
94:         int idx = (j-1)*NX + NX;
95:         a[idx][idx] = 1.0;
96:         b[idx] = T_right;
97:     }
98:
99:     // 下邊界 (j=1)
100:    for(int i = 1; i <= NX; i++){
101:        int idx = i;
102:        a[idx][idx] = 1.0;
103:        b[idx] = T_bottom;
104:    }
105:
106:    // 上邊界 (j=NY)
107:    for(int i = 1; i <= NX; i++){

```

```

108:         int idx = (NY-1)*NX + i;
109:         a[idx][idx] = 1.0 ;
110:         b[idx] = T_up(i);
111:     }
112:
113:     cout << "Setting interior points with consistent 6th-order 9 points
difference shceme ..." << endl;
114:     for(int j = 3; j <= NY-2; j++) {
115:         for(int i = 3; i <= NX-2; i++) {
116:             int idx = (j-1)*NX + i;
117:             a[idx][idx] = a1*H ;
118:             a[idx][idx+1] = a2*H ;//(1)右
119:             a[idx][idx+NX] = a2*H ;//(2)上
120:             a[idx][idx-1] = a2*H ;//(3)左
121:             a[idx][idx-NX] = a2*H ;//(4)下
122:             a[idx][idx+1+NX] = a3*H ;//(5)右上
123:             a[idx][idx-1+NX] = a3*H ;//(6)左上
124:             a[idx][idx-1-NX] = a3*H ;//(7)左下
125:             a[idx][idx+1-NX] = a3*H ;//(8)右下
126:             b[idx] = 0.0;
127:         }
128:     }
129:     double A = -2.0 * ((1.0/(dx*dx))+(1.0/(dy*dy))) ;
130:     double B = (1.0/(dx*dx)) ;
131:     double C = (1.0/(dy*dy)) ;
132:     //左邊界計算點
133:     //處理方式:東西方向採用單邊插分，南北方向採用中心差分
134:     for(int j = 3 ; j <= NY-2 ; j++){
135:         int idx = (j-1)*NX + 2 ; //i = 2 ;
136:         b[idx] = 0.0 ;
137:         int idx_plus = idx-1 ;
138:         for(int ac = 0 ; ac <= 5 ; ac++){
139:             a[idx][idx_plus+ac] = H2*cfd[ac] ;
140:         }
141:         a[idx][idx] += -2.0*C ;
142:         a[idx][idx-NX] = C ;
143:         a[idx][idx+NX] = C ;
144:
145:     }
146:     //右邊界計算點
147:     //處理方式:東西方向採用單邊插分，南北方向採用中心差分
148:     for(int j = 3 ; j <= NY-2 ; j++){
149:         int idx = (j-1)*NX + (NX-1) ; // i = (NX-1)
150:         b[idx] = 0.0 ;
151:         int idx_plus = idx+1 ;
152:         for(int ac = 0 ; ac <= 5 ; ac++){
153:             a[idx][idx_plus-ac] = H2*cfd[ac] ;
154:         }
155:         a[idx][idx] += -2.0*C ;
156:         a[idx][idx-NX] = C ;
157:         a[idx][idx+NX] = C ;
158:     }
159:     //下邊界計算點
160:     //處理方式:南北方向採用單邊差分，東西方向採用中心差分

```

```

161:     for(int i = 3 ; i <= NX-2 ; i++){
162:         int idx = (2-1)*NX + i ; // j = 2
163:         b[idx] = 0.0 ;
164:         int idx_plus = idx - NX ;
165:         for(int ac = 0 ; ac <= 5 ; ac++){
166:             a[idx][idx_plus+NX*ac] = H2*cfd[ac] ;
167:         }
168:         a[idx][idx] += -2.0*B ;
169:         a[idx][idx-1] = B ;
170:         a[idx][idx+1] = B ;
171:     }
172:     //上邊界計算點
173:     //處理方式:南北方向採用單邊差分，東西方向採用中心差分
174:     for(int i = 3 ; i <= NX-2 ; i++){
175:         int idx = (NY-2)*NX + i ; // j = (NY-2)
176:         b[idx] = 0.0 ;
177:         int idx_plus = idx + NX ;
178:         for(int ac = 0 ; ac <= 5 ; ac++){
179:             a[idx][idx_plus-NX*ac] = H2*cfd[ac] ;
180:         }
181:         a[idx][idx-1] = B ;
182:         a[idx][idx] += -2.0*B ;
183:         a[idx][idx-1] = B ;
184:         a[idx][idx+1] = B ;
185:     }
186:
187:     //左下角點
188:     int p1 = (2-1)*NX + 2 ;
189:     b[p1] = 0.0 ;
190:     a[p1][p1] = A ;
191:     a[p1][p1-1] = B ;
192:     a[p1][p1+1] = B ;
193:     a[p1][p1-NX] = C ;
194:     a[p1][p1+NX] = C ;
195:     //右下角點
196:     int p2 = (2-1)*NX + (NX-1) ;
197:     b[p2] = 0.0 ;
198:     a[p2][p2] = A ;
199:     a[p2][p2-1] = B ;
200:     a[p2][p2+1] = B ;
201:     a[p2][p2-NX] = C ;
202:     a[p2][p2+NX] = C ;
203:     //左上角點
204:     int p3 = (NY-2)*NX + 2 ;
205:     b[p3] = 0.0 ;
206:     a[p3][p3] = A ; // 修正：應該是 a[p3][p3] 而不是 a[p3][p3-1]
207:     a[p3][p3+1] = B ;
208:     a[p3][p3-1] = B ; // 這行是正確的
209:     a[p3][p3-NX] = C ;
210:     a[p3][p3+NX] = C ;
211:     int p4 = (NY-2)*NX + (NX-1) ;
212:     b[p4] = 0.0 ;
213:     a[p4][p4] = A ;
214:     a[p4][p4-1] = B ;

```

```

215:     a[p4][p4+1] = B;
216:     a[p4][p4-NX] = C;
217:     a[p4][p4+NX] = C;
218:     cout << "Matrix initialization completed." << endl;
219:     cout << "Total equations: " << n << endl;
220:     cout << " for 9 points difference shceme(4-order bound) ,Interior
points: " << (NX-4)*(NY-4) << endl;
221: }
222:
223: //超鬆弛迭代法_自適應性鬆弛因子
224: void SOR(vector<vector<double> >& a, vector<double>& b, vector<double>&
x, int n) {
225:     // 自適應鬆弛因子
226:     double omega;
227:     if(NX <= 21) omega = 0.5;
228:     else if(NX <= 41) omega = 0.8;
229:     else omega = 1.2;
230:
231:     //保存舊的解
232:     for(int k = 1; k <= n; k++) {
233:         x_old[k] = x[k];
234:     }
235:
236:     // SOR迭代
237:     for(int k = 1; k <= n; k++) {
238:         if(fabs(a[k][k]) < 1e-15) continue; // 跳過奇異矩陣
239:
240:         double sum = 0;
241:         for(int p = 1; p <= n; p++) {
242:             if(p != k) {
243:                 sum += a[k][p] * x[p];
244:             }
245:         }
246:         double x_new = (b[k] - sum) / a[k][k];
247:         x[k] = x_old[k] + omega * (x_new - x_old[k]);
248:     }
249:
250:     // 計算最大收斂誤差
251:     maxerror = 0;
252:     for(int k = 1; k <= n; k++) {
253:         double error = fabs(x[k] - x_old[k]);
254:         if(maxerror < error) {
255:             maxerror = error;
256:         }
257:     }
258:
259:     // 計算L1誤差
260:     double sum = 0;
261:     for(int k = 1; k <= n; k++) {
262:         sum += fabs(x[k] - T_analytical(k));
263:     }
264:     L1sum = sum / double(n);
265: }
266:

```

```

267:
268: void output(int m) {
269:     for(int j = 1; j <= NY; j++){
270:         for(int i = 1; i <= NX; i++){
271:             T[i-1][j-1] = x[(j-1)*NX + i];
272:         }
273:     }
274:
275:     ostringstream name;
276:     name << "9 points difference shceme(4-order bound)_" << NX << "x" <<
NY << "_" << setfill('0') << setw(6) << m << ".vtk";
277:     ofstream out(name.str().c_str());
278:
279:     out << "# vtk DataFile Version 3.0\n";
280:     out << "9 points difference shceme(4-order bound)\n";
281:     out << "ASCII\n";
282:     out << "DATASET STRUCTURED_POINTS\n";
283:     out << "DIMENSIONS " << NX << " " << NY << " 1\n";
284:     out << "ORIGIN 0.0 0.0 0.0\n";
285:     out << "SPACING " << dx << " " << dy << " 1.0\n";
286:     out << "POINT_DATA " << NX * NY << "\n";
287:
288:     out << "SCALARS Temperature double 1\n";
289:     out << "LOOKUP_TABLE default\n";
290:     for(int j = 0; j < NY; j++) {
291:         for(int i = 0; i < NX; i++) {
292:             out << scientific << setprecision(6) << T[i][j] << "\n";
293:         }
294:     }
295:
296:     out.close();
297:     cout << "VTK document output: " << name.str() << endl;
298: }
299:
300: void output_gnuplot_data() {
301:     bool valid_data = true;
302:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
303:         if(Finall1error[grid_idx] <= 0 || isnan(Finall1error[grid_idx])
|| isinf(Finall1error[grid_idx])) {
304:             cout << "Warning: Invalid L1 error for grid " << grid_idx <<
": " << Finall1error[grid_idx] << endl;
305:             valid_data = false;
306:         }
307:     }
308:
309:     if(!valid_data) {
310:         cout << "Cannot generate convergence analysis due to invalid
data." << endl;
311:         return;
312:     }
313:
314:     ofstream data_file("grid_convergence_9 points difference shceme(4-
order bound).dat");
315:     data_file << "# Grid_Size dx log(dx) L1_Error log(L1_Error)" << endl;

```

```

316:
317:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
318:         double dx_value = 1.0 / (Nx[grid_idx]-1);
319:         double log_dx = log(dx_value);
320:         double log_error = log(Finall1error[grid_idx]);
321:
322:         data_file << Nx[grid_idx] << "\t"
323:             << scientific << setprecision(6) << dx_value << "\t"
324:             << scientific << setprecision(6) << log_dx << "\t"
325:             << scientific << setprecision(6) <<
Finall1error[grid_idx] << "\t"
326:             << scientific << setprecision(6) << log_error << endl;
327:     }
328:     data_file.close();
329:     cout << "Data file output: grid_convergence_9 points difference
shceme(4-order bound).dat" << endl;
330:
331:     double sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
332:     int n_points = 4;
333:
334:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
335:         double x = log(1.0 / (Nx[grid_idx]-1));
336:         double y = log(Finall1error[grid_idx]);
337:
338:         sum_x += x;
339:         sum_y += y;
340:         sum_xy += x * y;
341:         sum_x2 += x * x;
342:     }
343:
344:     double slope = (n_points * sum_xy - sum_x * sum_y) / (n_points *
sum_x2 - sum_x * sum_x);
345:     double intercept = (sum_y - slope * sum_x) / n_points;
346:
347:     ofstream gnuplot_file("plot_convergence_9 points difference shceme(4-
order bound).plt");
348:     gnuplot_file << "set terminal png enhanced size 800,600" << endl;
349:     gnuplot_file << "set output 'grid_convergence_9 points difference
shceme(4-order bound).png'" << endl;
350:     gnuplot_file << "set title 'Improved Grid Convergence Analysis: L1
Error vs Grid Spacing'" << endl;
351:     gnuplot_file << "set xlabel 'log(dx)'" << endl;
352:     gnuplot_file << "set ylabel 'log(L1 Error)'" << endl;
353:     gnuplot_file << "set grid" << endl;
354:     gnuplot_file << "set key left top" << endl;
355:
356:     double x_min = log(1.0 / (Nx[3]-1));
357:     double x_max = log(1.0 / (Nx[0]-1));
358:     double y_ref = log(Finall1error[1]);
359:     double x_ref = log(1.0 / (Nx[1]-1));
360:
361:     gnuplot_file << "f(x) = " << slope << " * x + " << intercept << endl;
362:     gnuplot_file << "g(x) = 4.0 * (x - " << x_ref << ") + " << y_ref <<
endl;

```

```

363:
364:     gnuplot_file << "plot 'grid_convergence_9 points difference shceme(4-
order bound).dat' using 3:5 with linespoints pt 7 ps 1.5 lw 2 title
sprintf('Improved (slope = %.2f)', " << slope << "), "\\" << endl;
365:     gnuplot_file << "         f(x) with lines lw 2 lc rgb 'red' title
sprintf('Linear Fit (slope = %.2f)', " << slope << "), "\\" << endl;
366:     gnuplot_file << "         g(x) with lines lw 2 lc rgb 'green' dashtype 2
title '9 points difference shceme(4-order bound)(slope = 4.0)'" << endl;
367:
368:     gnuplot_file.close();
369:     cout << "Gnuplot script output: plot_convergence_9 points difference
shceme(4-order bound).plt" << endl;
370:
371:     cout << "\n=== Improved Grid Convergence Analysis ===" << endl;
372:     cout << "Linear regression results:" << endl;
373:     cout << "Slope = " << fixed << setprecision(3) << slope << "
(theoretical 4.0)" << endl;
374:     cout << "Order of accuracy = " << fixed << setprecision(3) << slope
<< endl;
375: }
376:
377: int main() {
378:     for(int grid_idx = 0; grid_idx < Nx.size(); grid_idx++) {
379:         cout << "\n===== " << endl;
380:         cout << "Grid size: " << Nx[grid_idx] << "x" << Ny[grid_idx] <<
endl;
381:
382:         NX = Nx[grid_idx];
383:         NY = Ny[grid_idx];
384:         n = NX * NY;
385:         dx = 1.0 / (NX-1);
386:         dy = 1.0 / (NY-1);
387:
388:         cout << "Grid spacing: dx = " << dx << ", dy = " << dy << endl;
389:
390:         a.assign(n+2, vector<double>(n+2, 0.0));
391:         b.assign(n+2, 0.0);
392:         x.assign(n+2, 0.0);
393:         x_old.assign(n+2, 0.0);
394:         T.assign(NX, vector<double>(NY, 0.0));
395:
396:         cout << "Program execution started with 9 points difference
shceme(4-order bound) ...." << endl;
397:         steadystate = false;
398:         initial(a, b, n); // 初始化
399:
400:         for(G = 0; G < max_G; G++) {
401:             SOR(a, b, x, n); // 使用改進的SOR
402:
403:             if(G % 1000 == 0) {
404:                 cout << "Iteration = " << G;
405:                 cout << ", Convergence error = " << scientific <<
setprecision(3) << maxerror;
406:                 cout << ", L1 error = " << scientific << setprecision(3)
<< L1sum << endl;

```



```

407:
408:         if(G % 5000 == 0) {
409:             output(G);
410:         }
411:     }
412:
413:     if(G > 100 && maxerror < tolerance) {
414:         steadystate = true;
415:         cout << "Steady state reached!" << endl;
416:         cout << "Final iteration: " << G << ", Final convergence
error : " << maxerror << endl;
417:         cout << "Final L1 error: " << L1sum << endl;
418:         Finall1error[grid_idx] = L1sum;
419:         break;
420:     }
421: }
422:
423: if(!steadystate) {
424:     cout << "Maximum iteration reached!" << endl;
425:     cout << "Final convergence error: " << maxerror << endl;
426:     cout << "Final L1 error: " << L1sum << endl;
427:     Finall1error[grid_idx] = L1sum;
428: }
429:
430:     output(G);
431:     cout << "Grid size " << NX << "x" << NY << " computation
completed" << endl;
432:     cout << "===== " << endl;
433: }
434:
435: output_gnuplot_data();
436: cout << "\nAll computations completed!" << endl;
437:
438: return 0;
439: }

```