Assignemnt 1 Conduction

學號：114233515 姓名：陳芃仲

October 29, 2025

Contents

1 Problem description

For a two dimensional conduction equation,

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1.1}$$

and Boundary Conditions are given as follows:

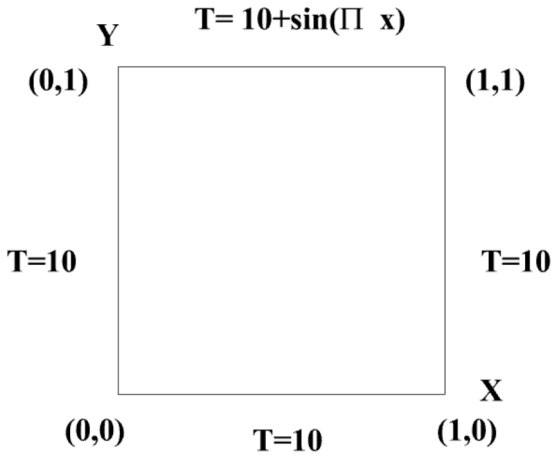

Figure 1: Boundary Conditions

Find the numerial solution of the above equation with the given boundary conditions. And compare the predicted results with analytic solutions . Question was shown below :

(1) indicate also how you solve the linear set of equations, i.e. Gauss Seidel ....

(2) the mesh sizes are set to be 11x11, 21x21, 41x41, and 81x81

(3) compare the predicted and analytic results at y=0.5 and comment on your results.

(4) Show the order of accuracy of the scheme for the second, fourth order schemes and the nine points formula.

2 Introduction of the methodology adopted

To solve the Laplace equation numerially , I use the finite difference method to discrete the eqaution and itteration method to solve the linear algebraic eqautions.

2.1 Numerical Methods

2.1.1 Itteration method

For a algebraic equation system , we can write it as :

$$A\vec{x} = \vec{b} \tag{2.1}$$

where A is the coefficient matrix, $\vec{x}$ is the unknown vector, and $\vec{b}$ is the constant vector. There are severl itteration methods to solve the linear algebraic equations, such as Jacobi method, Gauss-Seidel method and SOR method. In this assigment, I use the Gauss-Seidel method to solve the linear algebraic

equations. The basic idea of Gauss-Seidel method is to use the latest updated values to calculate the next unknowns. The formula is given as follows:

$$x_i^{n+1} = \frac{-\left(\sum_{k=1}^{i-1} a_{ik} x_k^{n+1} + \sum_{j=i+1}^{N} a_{ij} x_j^n\right)}{a_{ii}} \tag{2.2}$$

where x_i^{n+1} is the updated value of the i-th unknown at the (n+1)-th iteration, a_{ij} are the elements of the coefficient matrix A, and N is the total number of unknowns. The iteration continues until the solution converges to a specified tolerance level.and Jacobi method will be shwn below :

$$x_i^{n+1} = \frac{-\left(\sum_{k=1}^{i-1} a_{ik} x_k^{n} + \sum_{j=i+1}^{N} a_{ij} x_j^n\right)}{a_{ii}} \tag{2.3}$$

so we can see that G-S method use the latest value to get the solution for the next step and Jacobi method use the old value to get the next step solution . For the itteration rate , G-S method get more efficent itteration rate than Jacobi method .

2.1.2 Discrete Equation for Laplace Equation

(1) Finite Volume Method

the central difference scheme of the steady two dimensional conduction equation is below : (FVM)

$$\begin{aligned} &A_e \cdot (\frac{k_{i+1,j} + k_{i,j}}{2} \frac{T_{i+1,j} - T_{i,j}}{\Delta x_{Ep}}) - A_w \cdot (\frac{k_{i,j} + k_{i-1,j}}{2} \frac{T_{i,j} - T_{i-1,j}}{\Delta x_{pW}}) \\ &+ A_n \cdot (\frac{k_{i,j+1} + k_{i,j}}{2} \frac{T_{i,j+1} - T_{i,j}}{\Delta y_{Np}}) - A_s \cdot (\frac{k_{i,j} + k_{i,j-1}}{2} \frac{T_{i,j} - T_{i,j-1}}{\Delta y_{pS}}) = 0 \end{aligned} \tag{2.4}$$

and we can simplify it as below :

$$a_p T_p = a_W T_W + a_E T_E + a_S T_S + a_N T_N \tag{2.5}$$

where :

$$\begin{aligned} &T_p = T_{i,j}, a_p = a_W + a_E + a_S + a_N \\ &T_W = T_{i-1,j}, a_W = A_w \frac{k_{i,j} + k_{i-1,j}}{2\Delta x_{pW}} \\ &T_E = T_{i+1,j}, a_E = A_e \frac{k_{i+1,j} + k_{i,j}}{2\Delta x_{Ep}} \\ &T_S = T_{i,j-1}, a_S = A_s \frac{k_{i,j} + k_{i,j-1}}{2\Delta y_{pS}} \\ &T_N = T_{i,j+1}, a_N = A_n \frac{k_{i,j+1} + k_{i,j}}{2\Delta y_{Np}} \end{aligned} \tag{2.6}$$

(2)Four order Central Difference Scheme :

$$\begin{aligned} &\frac{1}{12\Delta x^2}(-T_{i+2,j} + 16T_{i+1,j} - 30T_{i,j} + 16T_{i-1,j} - T_{i-2,j}) + \\ &\frac{1}{12\Delta y^2}(-T_{i,j+2} + 16T_{i,j+1} - 30T_{i,j} + 16T_{i,j-1} - T_{i,j-2}) = 0 \end{aligned} \tag{2.7}$$

and for this scheme , the truncation error will be :

$$TE = -\frac{1}{90}(\Delta x^4 \frac{\partial^6 T}{\partial x^6} + \Delta y^4 \frac{\partial^6 T}{\partial y^6}) \tag{2.8}$$

(3) Nine Point Difference Scheme : The discrete equation is given as below :

$$\begin{aligned} &\frac{1}{6h^2}\left[4(T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1})\right. \\ &\left.+(T_{i+1,j+1} + T_{i-1,j+1} + T_{i+1,j-1} + T_{i-1,j-1}) - 20T_{i,j}\right] = 0 \end{aligned} \tag{2.9}$$

the accuracy of the scheme will be $O(\Delta x^6)$

3 Result and discussion

3.1 Results

3.1.1 Countours : 2-order Central Difference Scheme

the four contours of the temperature distribution for different mesh sizes(11x11 , 21x21 , 41x41 , 81x81) are shown below :

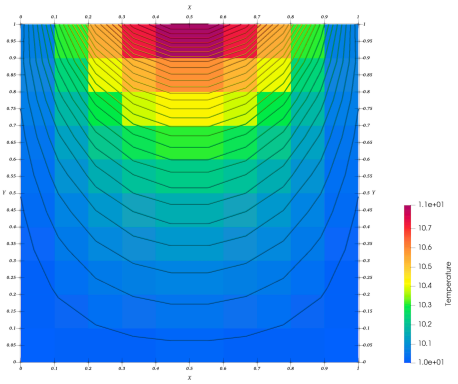

Figure 2: 11×11 mesh temperature contour

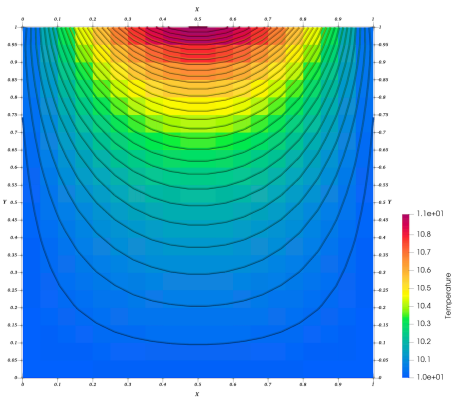

Figure 3: 21×21 mesh temperature contour

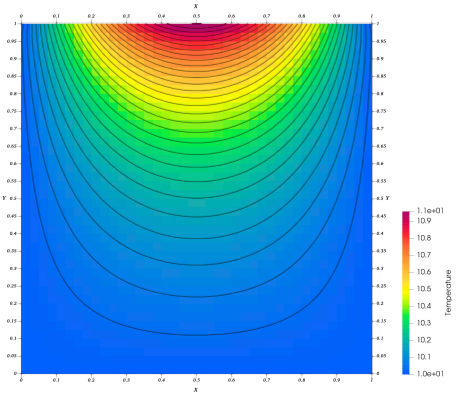

Figure 4: 41×41 mesh temperature contour

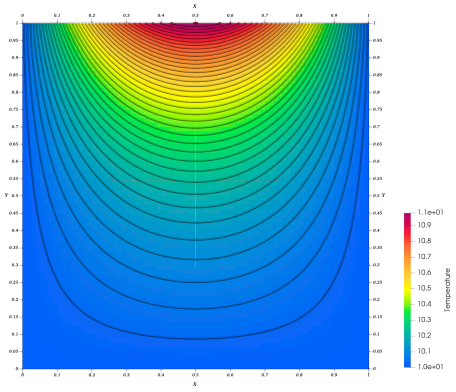

Figure 5: 81×81 mesh temperature contour

3.1.2 Countours : 4-order Central Difference Scheme

the four contours of the temperature distribution for different mesh sizes(11x11 , 21x21 , 41x41 , 81x81) are shown below :

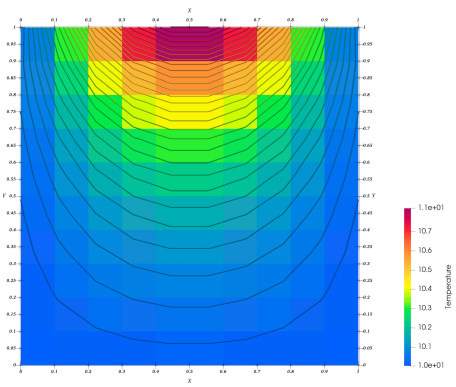

Figure 6: 11×11 mesh temperature contour

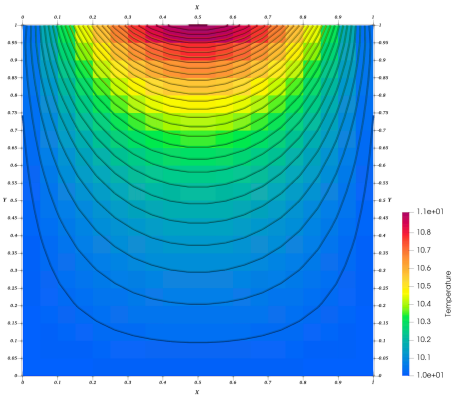

Figure 7: 21×21 mesh temperature contour

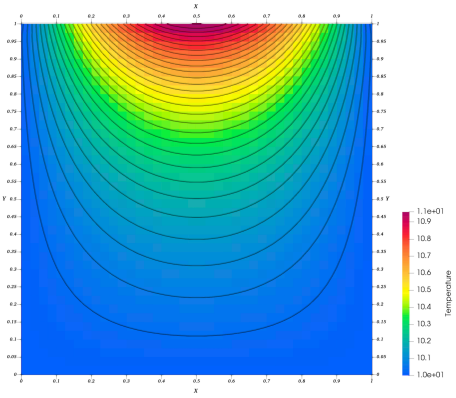

Figure 8: 41×41 mesh temperature contour

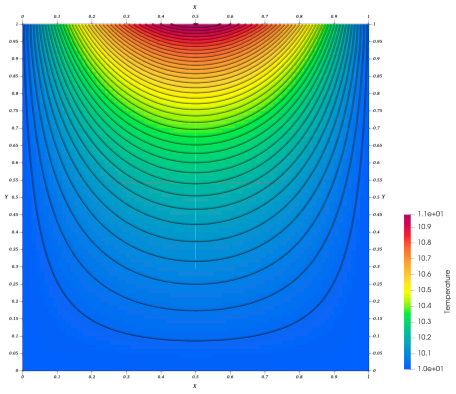

Figure 9: 81×81 mesh temperature contour

3.1.3 Countours :nine points formula scheme

the four contours of the temperature distribution for different mesh sizes(11x11 , 21x21 , 41x41 , 81x81) are shown below :

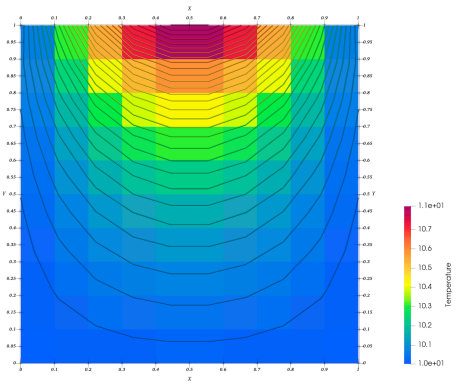

Figure 10: 11×11 mesh temperature contour

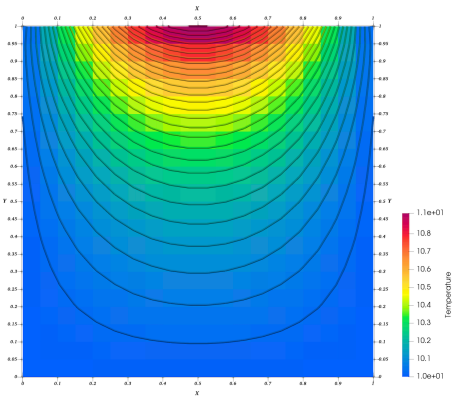

Figure 11: 21×21 mesh temperature contour

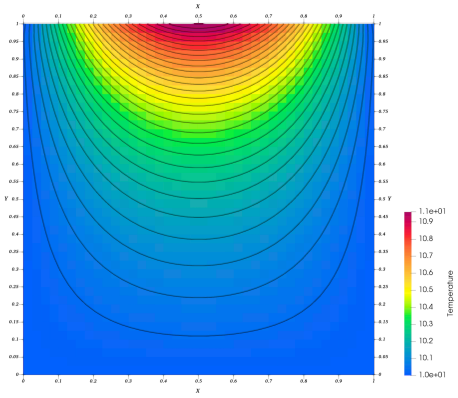

Figure 12: 41×41 mesh temperature contour

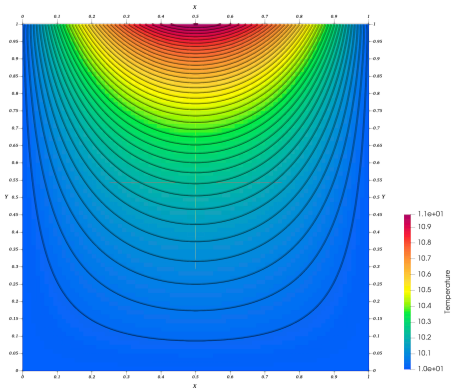

Figure 13: 81×81 mesh temperature contour

3.1.4 Centerline Temperature : Grid Size 11x11

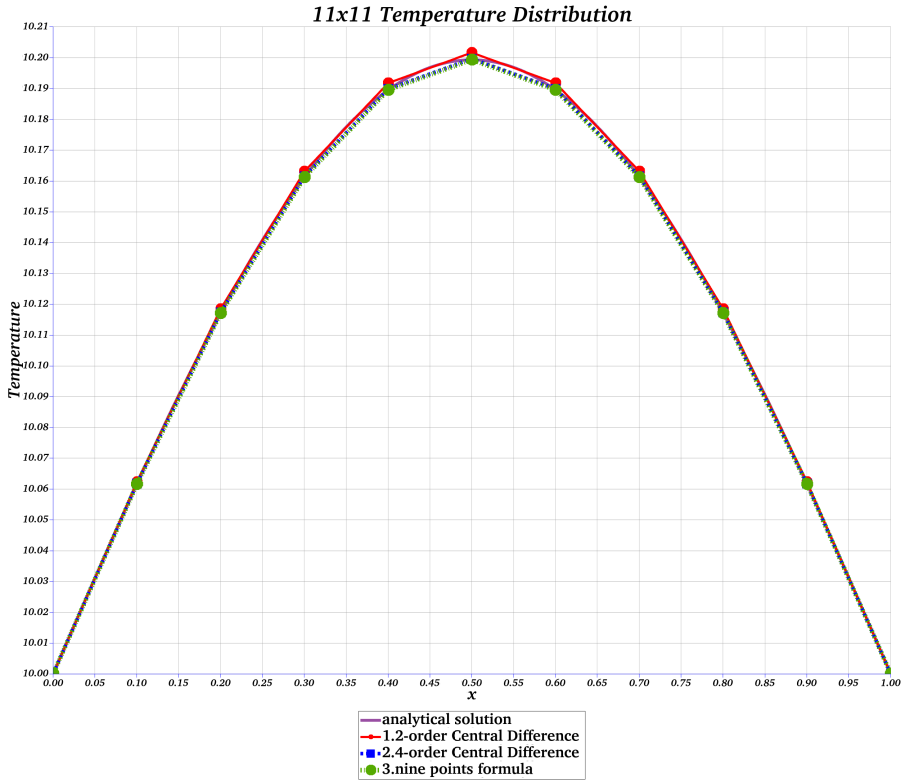

Figure 14: Centerline Temperature Distribution at y=0.5 (11x11 mesh)

3.1.5 Centerline Temperature : Grid Size 21x21

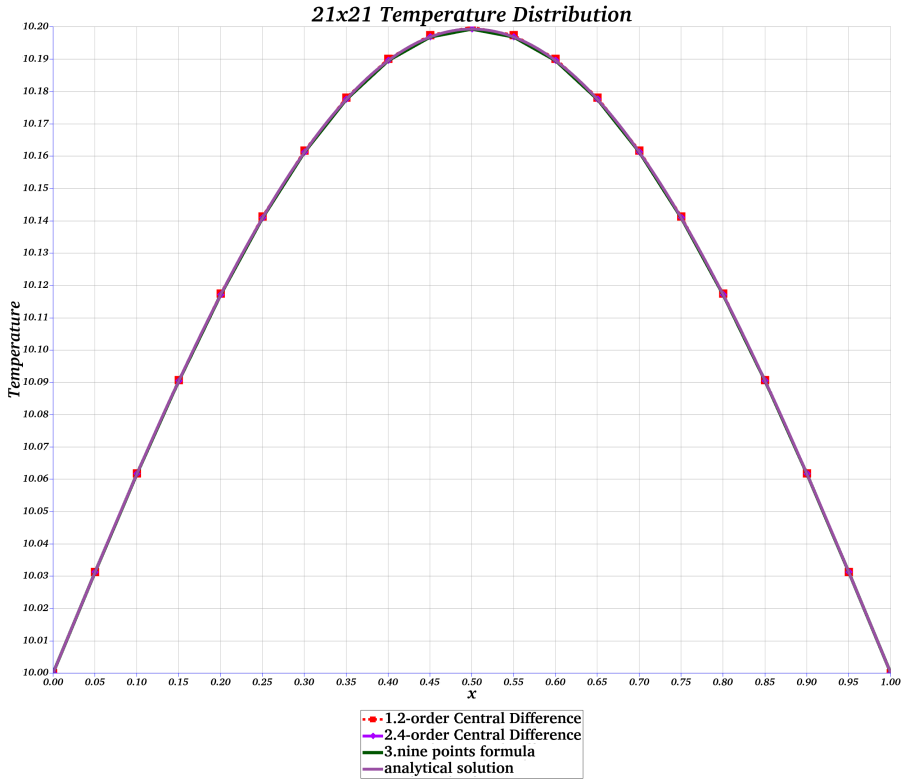

Figure 15: Centerline Temperature Distribution at y=0.5 (21x21 mesh)

3.1.6 Centerline Temperature : Grid Size 41x41

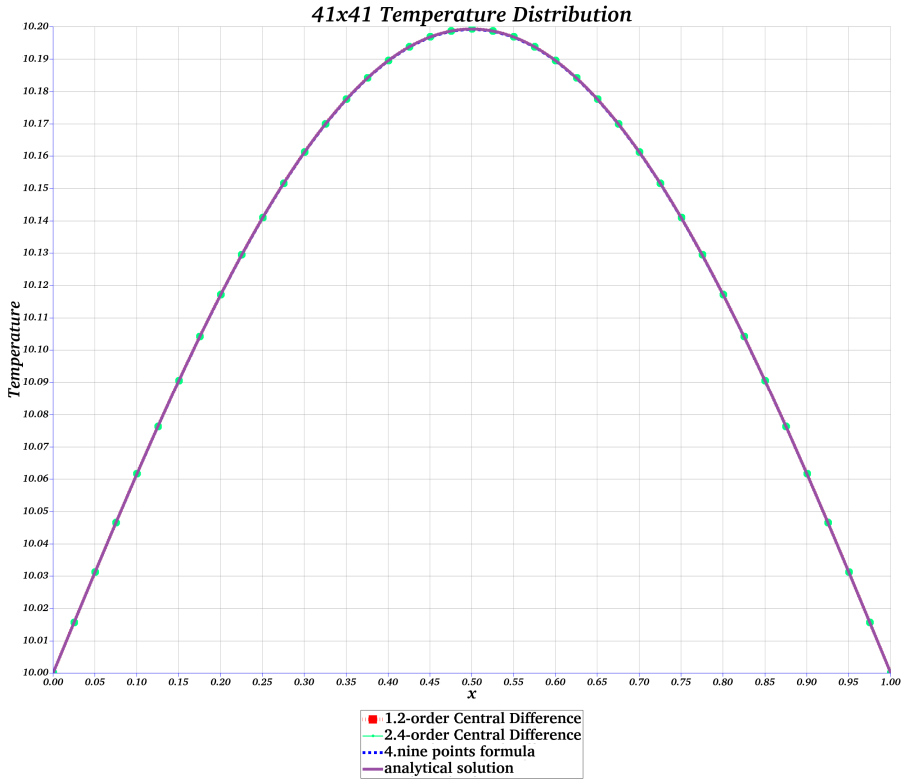

Figure 16: Centerline Temperature Distribution at y=0.5 (41x41 mesh)

3.1.7 Centerline Temperature : Grid Size 81x81

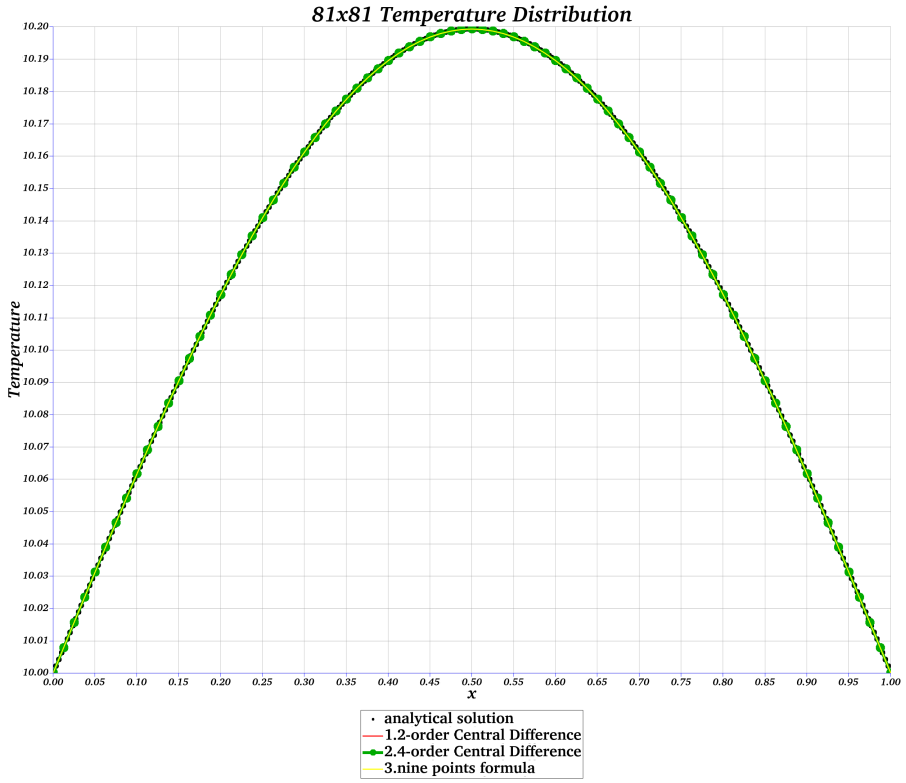

Figure 17: Centerline Temperature Distribution at y=0.5 (81x81 mesh)

3.2 Disscussion on the order of accuracy

3.2.1 Gernal Accuracy Analysis

To determine the order of accuracy of the numerical schemes used in this study, we analyze the error behavior as the grid is refined. The error (E) is calculated as the difference between the numerical solution and the analytical solution at specific points along the centerline (y=0.5). By plotting the logarithm of the error against the logarithm of the grid spacing (h), we can determine the order of accuracy (p) from the slope of the resulting line, as described by the relationship:

$$E \propto h^p \tag{3.1}$$

where E is the error, h is the grid spacing, and p is the order of accuracy. The slope of the line in the log-log plot gives us the value of p. From the analysis, we find that the second-order central difference scheme exhibits an order of accuracy close to 2, indicating that the error decreases quadratically as the grid is refined.

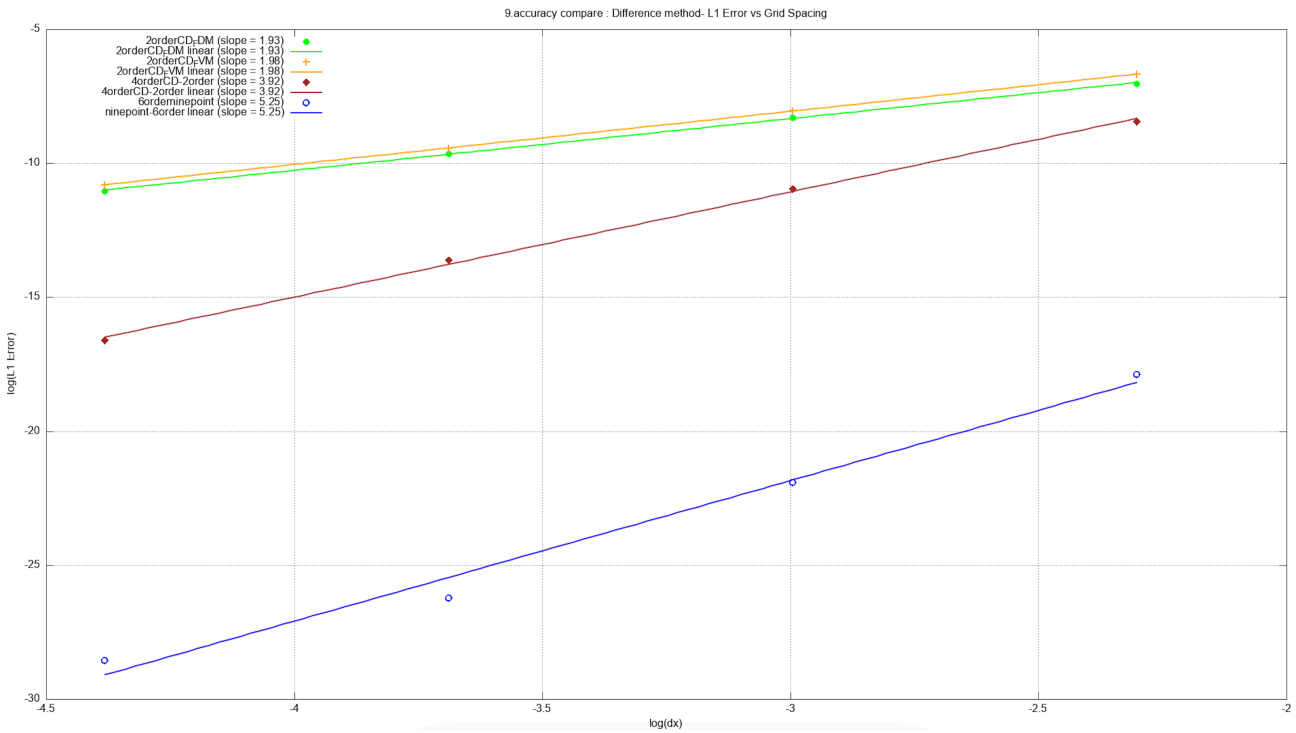

Figure 18: compare with different scheme for accuracy analysis

for the figure 17 , we can see that the slope of the second-order scheme is about 2 (green is FDM and yellow is FVM) , and the slope of the four order scheme is 3.92(red line) and the slope od the nine point formula is 5.25(blue line), so we can conclude that the order of accuracy for the second-order scheme is 2, for the four-order scheme is about 4, and for the nine-point formula is about 6.

3.2.2 Boundary treatment Analysis for nine point formula

In the nine point fourth-order scheme, the boundary treatment plays a crucial role in maintaining the overall accuracy of the numerical solution. The scheme requires information from neighboring points, including diagonal neighbors, which can be challenging at the boundaries of the computational domain. To address this, we employ ghost points or extrapolation techniques to estimate the values at these boundary points. This ensures that the finite difference approximations remain valid and that the order of accuracy is preserved. Proper boundary treatment is essential to avoid introducing significant errors that could degrade the solution's accuracy, especially in higher-order schemes like the nine-point formula. So I use three types of boundary tratment for this scheme :

As the figure 19 shows

(1) Red line : use second order Central Difference Scheme for boundary and corner points.

(2) Purple line : use four order Forward/Backward Diffeence Scheme for Boundary and second order Centeral Difference Scheme for corner points.

(3) Blue line : use the original set : nine point formula for boundary and corner points and put the boundary condition in the source term.

4 Conclusion

From the figure 18 and 19 , we can see that the red line has the worst accuracy because the second order scheme is used for boundary and corner points, which reduces the overall accuracy of the solution. The

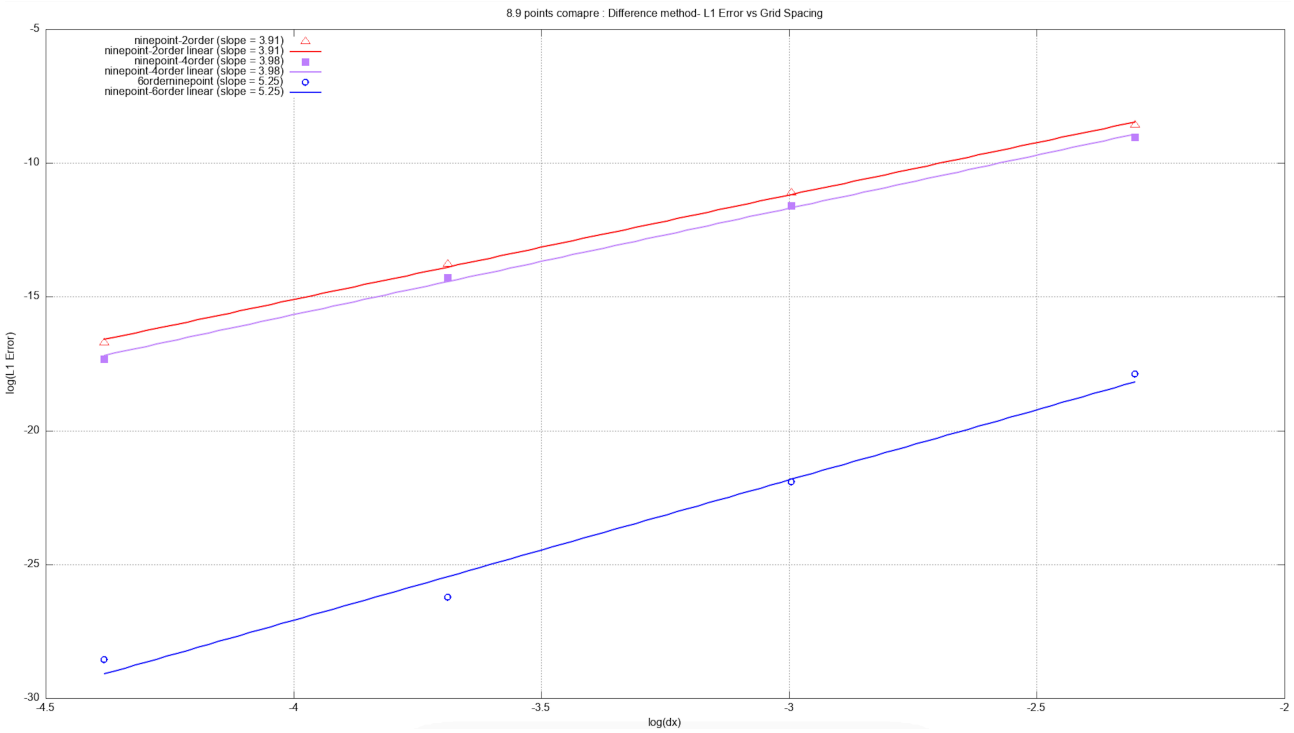

Figure 19: Boundary treatment for nine point formula

purple line shows improved accuracy by using a fourth-order scheme for the boundary points, but still relies on second-order approximations at the corners. The blue line demonstrates the best accuracy, as it consistently applies the nine-point formula throughout the domain, including the boundaries and corners, while incorporating boundary conditions into the source term. This approach effectively maintains the high order of accuracy inherent in the nine-point scheme, resulting in a more precise numerical solution.

5 List of Programs

5.1 2-order Central Difference Scheme (FDM)

```cpp
//完全修正版：有限差分法求解二維穩態熱擴散方程式
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
#include <cstdlib>
#include <string>
#include <vector>
#include <iomanip>
#define pi 3.14159265358979323846

using namespace std;

using namespace std;
int nx_data[] = {11, 21, 41, 81};
int ny_data[] = {11, 21, 41, 81};

vector<int> Nx(nx_data, nx_data + sizeof(nx_data)/sizeof(nx_data[0]));
vector<int> Ny(ny_data, ny_data + sizeof(ny_data)/sizeof(ny_data[0]));
double FinalL1error[4];
int n, NX, NY;
double dx, dy;
vector<vector<double> > a;
vector<double> b;
vector<double> x, x_old;
vector<vector<double> > T;
int G, max_G = 100000;
double T_left = 10.0;
double T_right = 10.0;
double T_bottom = 10.0;
double A, B, C;
double L1sum;
double maxerror;
const double tolerance = 1e-8;
bool steadystate;

// 上邊界條件
double T_up(int i){
    double x_pos = (i-1) * dx;
    return 10.0 + sin(pi * x_pos);
}

double T_analytical_fixed(double x_pos, double y_pos){
    return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
}

double T_analytical(int k){
    int i, j;
    i = ((k-1) % NX) + 1; // i = [1:NX]
    j = ((k-1) / NX) + 1; // j = [1:NY]

    double x_pos = (i-1) * dx;
    double y_pos = (j-1) * dy;

```

```
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 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: }
```

5.2 2-order Central Difference Scheme (FVM)

```cpp
//利用有限體積法求解二維穩態熱擴散方程式
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
#include <cstdlib>
#include <string>
#include <vector>
#include <iomanip>
#define pi 3.14159265358979323846

using namespace std;

//會用到的參數
int nx_data[] = {10, 20, 40, 80};
int ny_data[] = {10, 20, 40, 80};

vector<int> Nx(nx_data, nx_data + sizeof(nx_data)/sizeof(nx_data[0]));
vector<int> Ny(ny_data, ny_data + sizeof(ny_data)/sizeof(ny_data[0]));
double FinalL1error[4];
int n, NX, NY;
double dx, dy;
vector<vector<double> > a;
vector<double> b;
vector<double> x, x_old;
vector<vector<double> > T;
int G, max_G = 100000;
double T_left = 10.0;
double T_right = 10.0;
double T_bottom = 10.0;
double L1sum; //計算L1誤差
double maxerror; // 迭代收斂誤差
const double tolerance = 1e-18;
bool steadystate;

double T_up(int i){
 double x_pos = (i - 0.5) * dx;  // 使用網格中心點座標
 return 10.0 + sin(pi * x_pos);
}

double T_analytical_fixed(double x_pos, double y_pos){
 return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
}

double T_analytical(int k){
 int i, j;
 // 修正索引計算
 i = ((k-1) % NX) + 1; // i = [1:NX]
 j = ((k-1) / NX) + 1; // j = [1:NY]

 // 注意：這裡的座標計算要與網格一致
 // 對於從1開始的索引，x_pos和y_pos的計算如下：
 double x_pos = (i - 0.5) * dx;  // 網格中心點
 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:  ostringstream 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() 函數的最後，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: }
```

5.3 4-order Central Difference Scheme

```cpp
//四階精度差分格式求解Laplace方程
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
#include <cstdlib>
#include <string>
#include <vector>
#include <iomanip>
#define pi 3.14159265358979323846

using namespace std;

int nx_data[] = {11, 21, 41, 81};
int ny_data[] = {11, 21, 41, 81};

vector<int> Nx(nx_data, nx_data + sizeof(nx_data)/sizeof(nx_data[0]));
vector<int> Ny(ny_data, ny_data + sizeof(ny_data)/sizeof(ny_data[0]));
double FinalL1error[4];
int n, NX, NY;
double dx, dy;
vector<vector<double> > a;
vector<double> b;
vector<double> x, x_old;
vector<vector<double> > T;
int G, max_G = 100000;
double T_left = 10.0;
double T_right = 10.0;
double T_bottom = 10.0;
double L1sum;
double maxerror;
const double tolerance = 1e-10; // 迭代收斂判據
bool steadystate;

// 上邊界邊界條件
double T_up(int i){
    double x_pos = (i-1) * dx;
    return 10.0 + sin(pi * x_pos);
}

// 解析解
double T_analytical_fixed(double x_pos, double y_pos){
    return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
}

double T_analytical(int k){
    int i = ((k-1) % NX) + 1;
    int j = ((k-1) / NX) + 1;

    double x_pos = (i-1) * dx;
    double y_pos = (j-1) * dy;

    return T_analytical_fixed(x_pos, y_pos);
}
```

```
 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?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:     // 自適應松弛因子
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:     ostringstream 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: }
```

5.4 nine point formula Scheme with 2-order Boundary Treatment

5.5 nine point formula Scheme with 4-order Boundary Treatment

```cpp
//採用標準四階九點方程求解Laplace方程
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
#include <cstdlib>
#include <string>
#include <vector>
#include <iomanip>
#define pi 3.14159265358979323846

using namespace std;

int nx_data[] = {11, 21, 41, 81};
int ny_data[] = {11, 21, 41, 81};

vector<int> Nx(nx_data, nx_data + sizeof(nx_data)/sizeof(nx_data[0]));
vector<int> Ny(ny_data, ny_data + sizeof(ny_data)/sizeof(ny_data[0]));
double FinalL1error[4];
int n, NX, NY;
double dx, dy;
vector<vector<double> > a;
vector<double> b;
vector<double> x, x_old;
vector<vector<double> > T;
int G, max_G = 100000;
double T_left = 10.0;
double T_right = 10.0;
double T_bottom = 10.0;
double L1sum;
double maxerror;
const double tolerance = 1e-10; // 迭代收斂判據
bool steadystate;

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) } ;
//四階精度差分係數可依序排列做前向差分與後向差分

// 解析解
double T_analytical_fixed(double x_pos, double y_pos){
    return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
}

double T_analytical(int k){
    int i = ((k-1) % NX) + 1;
    int j = ((k-1) / NX) + 1;

    double x_pos = (i-1) * dx;
    double y_pos = (j-1) * dy;

    return T_analytical_fixed(x_pos, y_pos);
}

// 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:     // 自適應松弛因子
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: }
```

5.6 nine point formula Scheme with nine point Boundary Treatment

```cpp
//採用標準六階九點方程求解Laplace方程
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
#include <cstdlib>
#include <string>
#include <vector>
#include <iomanip>
#define pi 3.14159265358979323846

using namespace std;

int nx_data[] = {11, 21, 41, 81};
int ny_data[] = {11, 21, 41, 81};

vector<int> Nx(nx_data, nx_data + sizeof(nx_data)/sizeof(nx_data[0]));
vector<int> Ny(ny_data, ny_data + sizeof(ny_data)/sizeof(ny_data[0]));
double FinalL1error[4];
int n, NX, NY;
double dx, dy;
vector<vector<double> > a;
vector<double> b;
vector<double> x, x_old;
vector<vector<double> > T;
int G, max_G = 100000;
double T_left = 10.0;
double T_right = 10.0;
double T_bottom = 10.0;
double L1sum;
double maxerror;
const double tolerance = 1.422e-14; // 迭代收斂判據
bool steadystate;


//四階精度差分係數可依序排列做前向差分與後向差分

// 解析解
double T_analytical_fixed(double x_pos, double y_pos){
    return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
}

double T_analytical(int k){
    int i = ((k-1) % NX) + 1;
    int j = ((k-1) / NX) + 1;

    double x_pos = (i-1) * dx;
    double y_pos = (j-1) * dy;

    return T_analytical_fixed(x_pos, y_pos);
}

// 上邊界邊界條件
double T_up(int i){
    double x_pos = (i-1) * dx;
    return 10.0 + sin(pi * x_pos);
}


//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:     //1/(6*dx*dx) = 1/(6*dy*dy)
65:     double a1 = -20 ; //本點係數採用a1*H
66:     double a2 = 4.0 ; //直角方向臨計算點a2*H
67:     double a3 = 1.0 ; //斜角方向臨計算點a3*H
68: 
69: 
70: 
71:     // 初始化矩陣
72:     for(int i = 0; i <= n+1; i++) {
73:         for(int j = 0; j <= n+1; j++) {
74:             a[i][j] = 0.0;
75:         }
76:         b[i] = 0.0;
77:         x[i] = 0.0;
78:         x_old[i] = 0.0;
79:     }
80: 
81:     cout << "Setting boundary conditions..." << endl;
82: 
83:     // 左邊界條件 - 直接賦值
84:     // 左邊界 (i=1)
85:     for(int j = 1; j <= NY; j++){
86:         int idx = (j-1)*NX + 1;
87:         a[idx][idx] = 1.0;
88:         b[idx] = T_left;
89:     }
90: 
91:     // 右邊界 (i=NX)
92:     for(int j = 1; j <= NY; j++){
93:         int idx = (j-1)*NX + NX;
94:         a[idx][idx] = 1.0;
95:         b[idx] = T_right;
96:     }
97: 
98:     // 下邊界 (j=1)
99:     for(int i = 1; i <= NX; i++){
100:         int idx = i;
101:         a[idx][idx] = 1.0;
102:         b[idx] = T_bottom;
103:     }
104: 
105:     // 上邊界 (j=NY)
106:     for(int i = 1; i <= NX; i++){
107:         int idx = (NY-1)*NX + i;
108:         a[idx][idx] = 1.0 ;
109:         b[idx] = T_up(i);
110:     }
111: 
112:     cout << "Setting interior points with consistent 6th-order 9 points
    difference shceme ..." << endl;
113:     for(int j = 3; j <= NY-2; j++) {
114:         for(int i = 3; i <= NX-2; i++) {
115:             int idx = (j-1)*NX + i;
116:             a[idx][idx] = a1*H ;
117:             a[idx][idx+1] = a2*H ;//(1)右
118:             a[idx][idx+NX] = a2*H ;//(2)上
119:             a[idx][idx-1] = a2*H ;//(3)左
```

```
120:             a[idx][idx-NX] = a2*H ;//(4)下
121:             a[idx][idx+1+NX] = a3*H ;//(5)右上
122:             a[idx][idx-1+NX] = a3*H ;//(6)左上
123:             a[idx][idx-1-NX] = a3*H ;//(7)左下
124:             a[idx][idx+1-NX] = a3*H ;//(8)右下
125:             b[idx] = 0.0;
126:         }
127:     }
128:
129:     //左邊界計算點
130:     //處理方式:東西方向採用單邊插分，南北方向採用中心差分
131:     for(int j = 3 ; j <= NY-2 ; j++){
132:         int idx = (j-1)*NX + 2 ; //i = 2 ;
133:         a[idx][idx] = a1*H ;
134:         a[idx][idx+1] = a2*H ;//(1)右
135:         a[idx][idx+NX] = a2*H ;//(2)上
136:         a[idx][idx-1] = 0.0 ;//(3)左
137:         a[idx][idx-NX] = a2*H ;//(4)下
138:         a[idx][idx+1+NX] = a3*H ;//(5)右上
139:         a[idx][idx-1+NX] = 0.0 ;//(6)左上
140:         a[idx][idx-1-NX] = 0.0;//(7)左下
141:         a[idx][idx+1-NX] = a3*H ;//(8)右下
142:         b[idx] = -a2*H*T_left -a3*H*T_left -a3*H*T_left ;
143:     }
144:     //右邊界計算點
145:     //處理方式:東西方向採用單邊插分，南北方向採用中心差分
146:     for(int j = 3 ; j <= NY-2 ; j++){
147:         int idx = (j-1)*NX + (NX-1) ; // i = (NX-1)
148:         a[idx][idx] = a1*H ;
149:         a[idx][idx+1] = 0.0 ;//(1)右
150:         a[idx][idx+NX] = a2*H ;//(2)上
151:         a[idx][idx-1] = a2*H ;//(3)左
152:         a[idx][idx-NX] = a2*H ;//(4)下
153:         a[idx][idx+1+NX] = 0.0 ;//(5)右上
154:         a[idx][idx-1+NX] = a3*H ;//(6)左上
155:         a[idx][idx-1-NX] = a3*H ;//(7)左下
156:         a[idx][idx+1-NX] = 0.0 ;//(8)右下
157:         b[idx] = -a2*H*T_right -a3*H*T_right -a3*H*T_right ;
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:         a[idx][idx] = a1*H ;
164:         a[idx][idx+1] = a2*H ;//(1)右
165:         a[idx][idx+NX] = a2*H ;//(2)上
166:         a[idx][idx-1] = a2*H ;//(3)左
167:         a[idx][idx-NX] = 0.0 ;//(4)下
168:         a[idx][idx+1+NX] = a3*H ;//(5)右上
169:         a[idx][idx-1+NX] = a3*H ;//(6)左上
170:         a[idx][idx-1-NX] = 0.0 ;//(7)左下
171:         a[idx][idx+1-NX] = 0.0  ;//(8)右下
172:         b[idx] = -a2*H*T_bottom -a3*H*T_bottom -a3*H*T_bottom ;
173:     }
174:     //上邊界計算點
175:     //處理方式:南北方向採用單邊差分，東西方向採用中心差分
176:     for(int i = 3 ; i <= NX-2 ; i++){
177:         int idx = (NY-2)*NX + i ; // j = (NY-2)
178:         a[idx][idx] = a1*H ;
179:         a[idx][idx+1] = a2*H ;//(1)右
```

```
180:         a[idx][idx+NX] = 0.0 ;//(2)上
181:         a[idx][idx-1] = a2*H ;//(3)左
182:         a[idx][idx-NX] = a2*H ;//(4)下
183:         a[idx][idx+1+NX] = 0.0 ;//(5)右上
184:         a[idx][idx-1+NX] = 0.0 ;//(6)左上
185:         a[idx][idx-1-NX] = a3*H ;//(7)左下
186:         a[idx][idx+1-NX] = a3*H ;//(8)右下
187:         b[idx] = -a2*H*T_up(i) -a3*H*T_up(i-1) -a3*H*T_up(i+1) ;
188:     }
189:
190:     //左下角點
191:     int p1 = (2-1)*NX + 2 ;
192:     a[p1][p1] = a1*H ;
193:     a[p1][p1+1] = a2*H ;//(1)右
194:     a[p1][p1+NX] = a2*H  ;//(2)上
195:     a[p1][p1-1] = 0.0  ;//(3)左
196:     a[p1][p1-NX] = 0.0 ;//(4)下
197:     a[p1][p1+1+NX] = a3*H ;//(5)右上
198:     a[p1][p1-1+NX] = 0.0 ;//(6)左上
199:     a[p1][p1-1-NX] = 0.0  ;//(7)左下
200:     a[p1][p1+1-NX] = 0.0 ;//(8)右下
201:     b[p1] = -a2*H*T_left - a2*H*T_bottom - a3*H*T_left - a3*H*T_left -
    a3*H*T_bottom ;
202:     //右下角點
203:     int p2 = (2-1)*NX + (NX-1) ;
204:     a[p2][p2] = a1*H ;
205:     a[p2][p2+1] = 0.0 ;//(1)右
206:     a[p2][p2+NX] = a2*H ;//(2)上
207:     a[p2][p2-1] = a2*H ;//(3)左
208:     a[p2][p2-NX] = 0.0 ;//(4)下
209:     a[p2][p2+1+NX] = 0.0 ;//(5)右上
210:     a[p2][p2-1+NX] = a3*H ;//(6)左上
211:     a[p2][p2-1-NX] = 0.0 ;//(7)左下
212:     a[p2][p2+1-NX] = 0.0 ;//(8)右下
213:     b[p2] = -a2*H*T_right - a2*H*T_bottom - a3*H*T_right - a3*H*T_right -
    a3*H*T_bottom ;
214:     //左上角點
215:     int p3 = (NY-2)*NX + 2;
216:     a[p3][p3] = a1*H ;
217:     a[p3][p3+1] = a2*H ;//(1)右
218:     a[p3][p3+NX] = 0.0 ;//(2)上
219:     a[p3][p3-1] = 0.0 ;//(3)左
220:     a[p3][p3-NX] = a2*H ;//(4)下
221:     a[p3][p3+1+NX] = 0.0;//(5)右上
222:     a[p3][p3-1+NX] = 0.0 ;//(6)左上
223:     a[p3][p3-1-NX] = 0.0 ;//(7)左下
224:     a[p3][p3+1-NX] = a3*H ;//(8)右下
225:     b[p3] = -a2*H*T_left - a2*H*T_up(2) - a3*H*T_left - a3*H*T_up(1) -
    a3*H*T_up(3) ;
226:     //右上角點
227:     int p4 = (NY-2)*NX + (NX-1);
228:     a[p4][p4] = a1*H ;
229:     a[p4][p4-1] = 0.0 ;//(1)右
230:     a[p4][p4+NX] = 0.0 ;//(2)上
231:     a[p4][p4-1] = a2*H ;//(3)左
232:     a[p4][p4-NX] = a2*H ;//(4)下
233:     a[p4][p4+1+NX] = 0.0 ;//(5)右上
234:     a[p4][p4-1+NX] = 0.0 ;//(6)左上
235:     a[p4][p4-1-NX] = a3*H ;//(7)左下
236:     a[p4][p4+1-NX] = 0.0 ;//(8)右下
```

```
237:     b[p4] = -a2*H*T_right - a2*H*T_up(NX-1) - a3*H*T_right - a3*H*T_up(NX) -
     a3*H*T_up(NX-2) ;
238:     cout << "Matrix initialization completed." << endl;
239:     cout << "Total equations: " << n << endl;
240:     cout << " for 9 points difference shceme(4-order bound) ,Interior  points: "
     << (NX-4)*(NY-4) << endl;
241: }
242:
243: //超鬆弛迭代法_自適應性鬆弛因子
244: void SOR(vector<vector<double> >& a, vector<double>& b, vector<double>& x, int n)
     {
245:     // 自適應松弛因子
246:     double omega;
247:     if(NX <= 21) omega = 0.5;
248:     else if(NX <= 41) omega = 0.8;
249:     else omega = 1.6;
250:     if(G > 10000) omega = 2.0 ; // 大型矩陣使用較大鬆弛因子
251:     //保存舊的解
252:     for(int k = 1; k <= n; k++) {
253:         x_old[k] = x[k];
254:     }
255:
256:     // SOR迭代
257:     for(int k = 1; k <= n; k++) {
258:         if(fabs(a[k][k]) < 1e-15) continue; // 跳過奇異矩陣
259:
260:         double sum = 0;
261:         for(int p = 1; p <= n; p++) {
262:             if(p != k) {
263:                 sum += a[k][p] * x[p];
264:             }
265:         }
266:         double x_new = (b[k] - sum) / a[k][k];
267:         x[k] = x_old[k] + omega * (x_new - x_old[k]);
268:     }
269:
270:     // 計算最大收斂誤差
271:     maxerror = 0;
272:     for(int k = 1; k <= n; k++) {
273:         double error = fabs(x[k] - x_old[k]);
274:         if(maxerror < error) {
275:             maxerror = error;
276:         }
277:     }
278:
279:     // 計算L1誤差
280:     double sum = 0;
281:     for(int k = 1; k <= n; k++) {
282:         sum += fabs(x[k] - T_analytical(k));
283:     }
284:     L1sum = sum / double(n);
285: }
286:
287:
288: void output(int m) {
289:     for(int j = 1; j <= NY; j++){
290:         for(int i = 1; i <= NX; i++){
291:             T[i-1][j-1] = x[(j-1)*NX + i];
292:         }
293:     }
```

```
294:
295:     ostringstream name;
296:     name << "9 points difference shceme(6-order bound)_" << NX << "x" << NY <<
    "_" << setfill('0') << setw(6) << m << ".vtk";
297:     ofstream out(name.str().c_str());
298:
299:     out << "# vtk DataFile Version 3.0\n";
300:     out << "9 points difference shceme(6-order bound)_\n";
301:     out << "ASCII\n";
302:     out << "DATASET STRUCTURED_POINTS\n";
303:     out << "DIMENSIONS " << NX << " " << NY << " 1\n";
304:     out << "ORIGIN 0.0 0.0 0.0\n";
305:     out << "SPACING " << dx << " " << dy << " 1.0\n";
306:     out << "POINT_DATA " << NX * NY << "\n";
307:
308:     out << "SCALARS Temperature double 1\n";
309:     out << "LOOKUP_TABLE default\n";
310:     for(int j = 0; j < NY; j++) {
311:         for(int i = 0; i < NX; i++) {
312:             out << scientific << setprecision(6) << T[i][j] << "\n";
313:         }
314:     }
315:
316:     out.close();
317:     cout << "VTK document output: " << name.str() << endl;
318: }
319:
320: void output_gnuplot_data() {
321:     bool valid_data = true;
322:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
323:         if(FinalL1error[grid_idx] <= 0 || isnan(FinalL1error[grid_idx]) ||
    isinf(FinalL1error[grid_idx])) {
324:             cout << "Warning: Invalid L1 error for grid " << grid_idx << ": " <<
    FinalL1error[grid_idx] << endl;
325:             valid_data = false;
326:         }
327:     }
328:
329:     if(!valid_data) {
330:         cout << "Cannot generate convergence analysis due to invalid data." <<
    endl;
331:         return;
332:     }
333:
334:     ofstream data_file("grid_convergence_9 points difference shceme(6-order
    bound).dat");
335:     data_file << "# Grid_Size dx log(dx) L1_Error log(L1_Error)" << endl;
336:
337:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
338:         double dx_value = 1.0 / (Nx[grid_idx]-1);
339:         double log_dx = log(dx_value);
340:         double log_error = log(FinalL1error[grid_idx]);
341:
342:         data_file << Nx[grid_idx] << "\t"
343:                 << scientific << setprecision(6) << dx_value << "\t"
344:                 << scientific << setprecision(6) << log_dx << "\t"
345:                 << scientific << setprecision(6) << FinalL1error[grid_idx] <<
    "\t"
346:                 << scientific << setprecision(6) << log_error << endl;
347:     }
```

```
348:     data_file.close();
349:     cout << "Data file output: grid_convergence_9 points difference shceme(6-
     order bound).dat" << endl;
350:
351:     double sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
352:     int n_points = 4;
353:
354:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
355:         double x = log(1.0 / (Nx[grid_idx]-1));
356:         double y = log(FinalL1error[grid_idx]);
357:
358:         sum_x += x;
359:         sum_y += y;
360:         sum_xy += x * y;
361:         sum_x2 += x * x;
362:     }
363:
364:     double slope = (n_points * sum_xy - sum_x * sum_y) / (n_points * sum_x2 -
     sum_x * sum_x);
365:     double intercept = (sum_y - slope * sum_x) / n_points;
366:
367:     ofstream gnuplot_file("plot_convergence_9 points difference shceme(6-order
     bound).plt");
368:     gnuplot_file << "set terminal png enhanced size 800,600" << endl;
369:     gnuplot_file << "set output 'grid_convergence_9 points difference shceme(6-
     order bound).png'" << endl;
370:     gnuplot_file << "set title '9 points difference scheme (6-order bound)'" <<
     endl;
371:     gnuplot_file << "set xlabel 'log(dx)'" << endl;
372:     gnuplot_file << "set ylabel 'log(L1 Error)'" << endl;
373:     gnuplot_file << "set grid" << endl;
374:     gnuplot_file << "set key left top" << endl;
375:
376:     double x_min = log(1.0 / (Nx[3]-1));
377:     double x_max = log(1.0 / (Nx[0]-1));
378:     double y_ref = log(FinalL1error[1]);
379:     double x_ref = log(1.0 / (Nx[1]-1));
380:
381:     gnuplot_file << "f(x) = " << slope << " * x + " << intercept << endl;
382:     gnuplot_file << "g(x) = 6.0 * (x - " << x_ref << ") + " << y_ref << endl;
383:
384:     gnuplot_file << "plot 'grid_convergence_9 points difference shceme(6-order
     bound).dat' using 3:5 with linespoints pt 7 ps 1.5 lw 2 title sprintf('Improved
     (slope = %.2f)', " << slope << "), \\" << endl;
385:     gnuplot_file << "     f(x) with lines lw 2 lc rgb 'red' title sprintf('Linear
     Fit (slope = %.2f)', " << slope << "), \\" << endl;
386:     gnuplot_file << "     g(x) with lines lw 2 lc rgb 'green' dashtype 2 title '9
     points difference shceme(6-order bound)(slope = 6.0)'" << endl;
387:
388:     gnuplot_file.close();
389:     cout << "Gnuplot script output: plot_convergence_9 points difference shceme(6-
     order bound).plt" << endl;
390:
391:     cout << "Gnuplot script output: plot_convergence_9 points difference shceme(6-
     order bound).plt" << endl;
392:     cout << "\n=== Improved Grid Convergence Analysis ===" << endl;
393:     cout << "Linear regression results:" << endl;
394:     cout << "Slope = " << fixed << setprecision(3) << slope << " (theoretical
     4.0)" << endl;
395:     cout << "Order of accuracy = " << fixed << setprecision(3) << slope << endl;
```

```
396: }
397: void output_gnuplot_data2() {
398:     double sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
399:     int n_points = 4;
400:
401:     for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
402:         double x = log(1.0 / (Nx[grid_idx]-1));
403:         double y = log(FinalL1error[grid_idx]);
404:
405:         sum_x += x;
406:         sum_y += y;
407:         sum_xy += x * y;
408:         sum_x2 += x * x;
409:     }
410:
411:     double slope = (n_points * sum_xy - sum_x * sum_y) / (n_points * sum_x2 -
    sum_x * sum_x);
412:     double intercept = (sum_y - slope * sum_x) / n_points;
413:     ofstream gnuplot_file("plot_convergence_9 points difference shceme(6-order
    bound)2.plt");
414:     gnuplot_file << "set terminal png enhanced size 800,600" << endl;
415:     gnuplot_file << "set output 'grid_convergence_9 points difference shceme(6-
    order bound)2.png'" << endl;
416:     gnuplot_file << "set title '9 points difference scheme (6-order bound)2'" <<
    endl;
417:     gnuplot_file << "set xlabel 'log(dx)'" << endl;
418:     gnuplot_file << "set ylabel 'log(L1 Error)'" << endl;
419:     gnuplot_file << "set grid" << endl;
420:     gnuplot_file << "set key left top" << endl;
421:
422:     double x_min = log(1.0 / (Nx[3]-1));
423:     double x_max = log(1.0 / (Nx[0]-1));
424:     double y_ref = log(FinalL1error[1]);
425:     double x_ref = log(1.0 / (Nx[1]-1));
426:
427:     gnuplot_file << "f(x) = " << slope << " * x + " << intercept << endl;
428:
429:     gnuplot_file << "plot 'grid_convergence_9 points difference shceme(6-order
    bound).dat' using 3:5 with linespoints pt 7 ps 1.5 lw 2 title sprintf('Improved
    (slope = %.2f)', " << slope << "), \\" << endl;
430:     gnuplot_file << "     f(x) with lines lw 2 lc rgb 'red' title sprintf('Linear
    Fit (slope = %.2f)', " << slope << "), \\" << endl;
431:
432:     gnuplot_file.close();
433:     cout << "Gnuplot script output: plot_convergence_9 points difference shceme(6-
    order bound)2.plt" << endl;
434:
435:
436: }
437:
438:
439:
440: int main() {
441:     for(int grid_idx = 0; grid_idx < Nx.size(); grid_idx++) {
442:         cout << "\n======================================" << endl;
443:         cout << "Grid size: " << Nx[grid_idx] << "x" << Ny[grid_idx] << endl;
444:
445:         NX = Nx[grid_idx];
446:         NY = Ny[grid_idx];
447:         n = NX * NY;
```

```
448:         dx = 1.0 / (NX-1);
449:         dy = 1.0 / (NY-1);
450:
451:         cout << "Grid spacing: dx = " << dx << ", dy = " << dy << endl;
452:
453:         a.assign(n+2, vector<double>(n+2, 0.0));
454:         b.assign(n+2, 0.0);
455:         x.assign(n+2, 0.0);
456:         x_old.assign(n+2, 0.0);
457:         T.assign(NX, vector<double>(NY, 0.0));
458:
459:         cout << "Program execution started with  9 points difference shceme(4-
    order bound) ...." << endl;
460:         steadystate = false;
461:         initial(a, b, n);  //初始化
462:
463:         for(G = 0; G < max_G; G++) {
464:             SOR(a, b, x, n);  // 使用改進的SOR
465:
466:             if(G % 1000 == 0) {
467:                 cout << "Iteration = " << G;
468:                 cout << ", Convergence error = " << scientific << setprecision(3)
    << maxerror;
469:                 cout << ", L1 error = " << scientific << setprecision(3) << L1sum
    << endl;
470:
471:                 if(G % 5000 == 0) {
472:                     output(G);
473:                 }
474:             }
475:
476:             if(G > 100 && maxerror < tolerance) {
477:                 steadystate = true;
478:                 cout << "Steady state reached!" << endl;
479:                 cout << "Final iteration: " << G << ", Final convergence error :
    " << maxerror << endl;
480:                 cout << "Final L1 error: " << L1sum << endl;
481:                 FinalL1error[grid_idx] = L1sum;
482:                 break;
483:             }
484:         }
485:
486:         if(!steadystate) {
487:             cout << "Maximum iteration reached!" << endl;
488:             cout << "Final convergence error: " << maxerror << endl;
489:             cout << "Final L1 error: " << L1sum << endl;
490:             FinalL1error[grid_idx] = L1sum;
491:         }
492:
493:         output(G);
494:         cout << "Grid size " << NX << "x" << NY << " computation completed" <<
    endl;
495:         cout << "=====================================" << endl;
496:     }
497:
498:     output_gnuplot_data();
499:     output_gnuplot_data2() ;
500:     cout << "\nAll computations completed!" << endl;
501:
502:     return 0;
```

```
503: }
```