```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
```

```
 return sin(pi * x_pos) * (sinh(pi * y_pos) / sinh(pi)) + 10.0;
}

//初始化矩陣
void initial(vector<vector<double> >& a, vector<double>& b, int n) {
 // 初始化所有元素為0
 for(int i = 1; i <= n; i++) {
 for(int j = 1; j <= n; j++) {
 a[i][j] = 0.0;
 }
 b[i] = 0.0;
 }

 for(int i = 0; i <= n+1; i++){
 x[i] = 0.0;
 x_old[i] = 0.0;
 }

 // 設定邊界條件和係數矩陣
 // 四個角點
 // 左下角 (1,1)
 a[1][1] = 6.0;
 a[1][2] = -1.0; // 右鄰點
 a[1][1+NX] = -1.0; // 上鄰點
 b[1] = 2.0 * (T_left + T_bottom);

 // 右下角 (NX,1)
 a[NX][NX] = 6.0;
 a[NX][NX-1] = -1.0; // 左鄰點
 a[NX][NX+NX] = -1.0; // 上鄰點
 b[NX] = 2.0 * (T_right + T_bottom);

 // 左上角 (1,NY)
 a[n-NX+1][n-NX+1] = 6.0;
 a[n-NX+1][n-NX+2] = -1.0; // 右鄰點
 a[n-NX+1][n-NX+1-NX] = -1.0; // 下鄰點
 double T_up_left = 10.0 + sin(pi * (0.5 * dx));  // 左上角的上邊界值
 b[n-NX+1] = 2.0 * (T_left + T_up_left);
 // 右上角 (NX,NY)
 a[n][n] = 6.0;
 a[n][n-1] = -1.0; // 左鄰點
 a[n][n-NX] = -1.0; // 下鄰點
 double T_up_right = 10.0 + sin(pi * ((NX - 0.5) * dx));  // 右上角的上邊界值
b[n] = 2.0 * (T_right + T_up_right);

 // 下邊界 (除角點外)
 for(int i = 2; i <= NX-1; i++) {
 a[i][i] = 5.0;
 a[i][i+1] = -1.0; // 右鄰點
 a[i][i-1] = -1.0; // 左鄰點
 a[i][i+NX] = -1.0; // 上鄰點
 b[i] = 2.0 * T_bottom;
```

```
}

// 上邊界 (除角點外)
for(int i = 2; i <= NX-1; i++) {
int idx = n - NX + i;
a[idx][idx] = 5.0;
a[idx][idx+1] = -1.0; // 右鄰點
a[idx][idx-1] = -1.0; // 左鄰點
a[idx][idx-NX] = -1.0; // 下鄰點
double T_up_val = 10.0 + sin(pi * ((i - 0.5) * dx));  // 使用網格中心點
b[idx] = 2.0 * T_up_val;
}

// 左邊界 (除角點外)
for(int j = 2; j <= NY-1; j++){
int idx = (j-1) * NX + 1;
a[idx][idx] = 5.0;
a[idx][idx+1] = -1.0; // 右鄰點
a[idx][idx+NX] = -1.0; // 上鄰點
a[idx][idx-NX] = -1.0; // 下鄰點
b[idx] = 2.0 * T_left;
}

// 右邊界 (除角點外)
for(int j = 2; j <= NY-1; j++){
int idx = (j-1) * NX + NX;
a[idx][idx] = 5.0;
a[idx][idx-1] = -1.0; // 左鄰點
a[idx][idx-NX] = -1.0; // 下鄰點
a[idx][idx+NX] = -1.0; // 上鄰點
b[idx] = 2.0 * T_right;
}

// 內點
for(int j = 2; j <= NY-1; j++) {
for(int i = 2; i <= NX-1; i++) {
int idx = (j-1)*NX + i;
a[idx][idx] = 4.0;
a[idx][idx+1] = -1.0; // 右鄰點
a[idx][idx-1] = -1.0; // 左鄰點
a[idx][idx+NX] = -1.0; // 上鄰點
a[idx][idx-NX] = -1.0; // 下鄰點
b[idx] = 0.0; // 內點無熱源
}
}
}

void Jacobi(vector<vector<double> >& a, vector<double>& b, vector<double>& x, int n) {
// 先複製當前解到x_old
for(int k = 1; k <= n; k++) {
x_old[k] = x[k];
}

```

```
    // 計算新的解
    for(int k = 1; k <= n; k++) {
    double sum = 0;
    for(int p = 1; p <= n; p++) {
    if(p != k) {
    sum += a[k][p] * x_old[p];
    }
    }
    x[k] = (b[k] - sum) / a[k][k];
    }

    // 計算迭代收斂誤差
    maxerror = 0;
    for(int k = 1; k <= n; k++) {
    double error = fabs(x[k] - x_old[k]);
    if(maxerror < error) {
    maxerror = error;
    }
    }

    // 計算L1誤差（與解析解比較）
    double sum = 0;
    for(int k = 1; k <= n; k++) {
    sum += fabs(x[k] - T_analytical(k));
    }
    L1sum = sum / double(n);
}

void output(int m) {
    // 將一維解轉換為二維溫度場
    for(int j = 1; j <= NY; j++){
    for(int i = 1; i <= NX; i++){
    T[i-1][j-1] = x[(j-1)*NX + i];
    }
    }

    ostringstream name;
    name << "steady_diffusion_2D_" << NX << "x" << NY << "_" << setfill('0') << setw(6) << m << ".vtk";
    ofstream out(name.str().c_str());

    // VTK 文件頭
    out << "# vtk DataFile Version 3.0\n";
    out << "steady_diffusion_2D\n";
    out << "ASCII\n";
    out << "DATASET STRUCTURED_POINTS\n";
    out << "DIMENSIONS " << NX << " " << NY << " 1\n";
    out << "ORIGIN 0.0 0.0 0.0\n";
    out << "SPACING " << dx << " " << dy << " 1.0\n";
    out << "POINT_DATA " << NX * NY << "\n";

    // 輸出溫度場
    out << "SCALARS Temperature double 1\n";
    out << "LOOKUP_TABLE default\n";
```

```cpp
 for(int j = 0; j < NY; j++) {
 for(int i = 0; i < NX; i++) {
 out << scientific << setprecision(6) << T[i][j] << "\n";
 }
 }

 out.close();
 cout << "VTK document output: " << name.str() << endl;
}

// 添加到你的程式碼中的新函數

void output_gnuplot_data() {
    // 輸出數據檔案 (.dat) //一共四組數據
    ofstream data_file("grid_convergence_data.dat");
    data_file << "# Grid_Size dx log(dx) L1_Error log(L1_Error)" << endl;

    for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
        double dx_value = 1.0 / Nx[grid_idx];  // 對應你的 dx 計算方式
        double log_dx = log(dx_value);
        double log_error = log(FinalL1error[grid_idx]);

        data_file << Nx[grid_idx] << "\t"
                  << scientific << setprecision(6) << dx_value << "\t"
                  << scientific << setprecision(6) << log_dx << "\t"
                  << scientific << setprecision(6) << FinalL1error[grid_idx] << "\t"
                  << scientific << setprecision(6) << log_error << endl;
    }
    data_file.close();
    cout << "Data file output: grid_convergence_data.dat" << endl;

    // 計算線性回歸的斜率和截距
    double sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
    int n_points = 4;

    for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
        double x = log(1.0 / Nx[grid_idx]);  // log(dx)
        double y = log(FinalL1error[grid_idx]);  // log(error)

        sum_x += x;
        sum_y += y;
        sum_xy += x * y;
        sum_x2 += x * x;
    }

    double slope = (n_points * sum_xy - sum_x * sum_y) / (n_points * sum_x2 - sum_x * sum_x);
    double intercept = (sum_y - slope * sum_x) / n_points;

    // 輸出 gnuplot 腳本
    ofstream gnuplot_file("plot_convergence.plt");
    gnuplot_file << "# Gnuplot script for grid convergence analysis" << endl;
```

```cpp
    gnuplot_file << "set terminal png enhanced size 800,600" << endl;
    gnuplot_file << "set output 'grid_convergence.png'" << endl;
    gnuplot_file << "set title 'Grid Convergence Analysis - L1 Error vs Grid Spacing'" << endl;
    gnuplot_file << "set xlabel 'log(dx)'" << endl;
    gnuplot_file << "set ylabel 'log(L1 Error)'" << endl;
    gnuplot_file << "set grid" << endl;
    gnuplot_file << "set key left top" << endl;
    gnuplot_file << "" << endl;

    // 理論2階精度線 (斜率=2)
    double x_min = log(1.0 / Nx[3]);  // 最小的 log(dx) (對應最細網格)
    double x_max = log(1.0 / Nx[0]);  // 最大的 log(dx) (對應最粗網格)
    double y_ref = log(FinalL1error[1]);  // 參考點 (使用第二個點)
    double x_ref = log(1.0 / Nx[1]);

    gnuplot_file << "# 線性回歸線: y = " << slope << " * x + " << intercept << endl;
    gnuplot_file << "# 理論2階精度線通過參考點 (" << x_ref << ", " << y_ref << ")" << endl;
    gnuplot_file << "f(x) = " << slope << " * x + " << intercept << endl;
    gnuplot_file << "g(x) = 2.0 * (x - " << x_ref << ") + " << y_ref << endl;
    gnuplot_file << "" << endl;
    //gnuplot運行用到plt.數據
    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;
    gnuplot_file << "     f(x) with lines lw 2 lc rgb 'red' title sprintf('Linear Fit (slope = %.2f)', " << slope << "), \\" << endl;
    gnuplot_file << "     g(x) with lines lw 2 lc rgb 'green' dashtype 2 title '2nd Order Theory (slope = 2.0)'" << endl;

    gnuplot_file.close();
    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
//所以是的，兩個檔案都必須在同一資料夾！

    // 輸出結果摘要
    cout << "\n=== Grid Convergence Analysis ===" << endl;
    cout << "Linear regression results:" << endl;
    cout << "Slope = " << fixed << setprecision(3) << slope << " (理論值應接近 2.0)" << endl;
    cout << "Intercept = " << fixed << setprecision(3) << intercept << endl;
    cout << "Order of accuracy = " << fixed << setprecision(3) << slope << endl;

    // 計算相關係數
    double mean_x = sum_x / n_points;
    double mean_y = sum_y / n_points;
    double ss_xx = 0, ss_yy = 0, ss_xy = 0;

```

```
    for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
        double x = log(1.0 / Nx[grid_idx]);
        double y = log(FinalL1error[grid_idx]);
        ss_xx += (x - mean_x) * (x - mean_x);
        ss_yy += (y - mean_y) * (y - mean_y);
        ss_xy += (x - mean_x) * (y - mean_y);
    }

    double correlation = ss_xy / sqrt(ss_xx * ss_yy);
    cout << "Correlation coefficient R = " << fixed << setprecision(4) << correlation << endl;
    cout << "R2 = " << fixed << setprecision(4) << correlation * correlation << endl;

    cout << "\nTo generate the plot, run: gnuplot plot_convergence.plt" << endl;
    cout << "====================================" << endl;
}

// 在 main() 函數的最後，在 output2() 之後添加：
// output_gnuplot_data();


void output3() {
 // 設定解析解的網格大小
 int analytical_NX = 80;
 int analytical_NY = 80;
 double analytical_dx = 1.0 / (analytical_NX);
 double analytical_dy = 1.0 / (analytical_NY);

 ostringstream name;
 name << "Analytical_solution_" << analytical_NX << "x" << analytical_NY << "_" << setfill('0') << setw(6) << 0 << ".vtk";
 ofstream out(name.str().c_str());

 // VTK 文件頭
 out << "# vtk DataFile Version 3.0\n";
 out << "Analytical_solution_" << analytical_NX << "x" << analytical_NY << "\n";
 out << "ASCII\n";
 out << "DATASET STRUCTURED_POINTS\n";
 out << "DIMENSIONS " << analytical_NX << " " << analytical_NY << " 1\n";
 out << "ORIGIN 0.0 0.0 0.0\n";
 out << "SPACING " << analytical_dx << " " << analytical_dy << " 1.0\n";
 out << "POINT_DATA " << analytical_NX * analytical_NY << "\n";

 // 輸出解析解溫度場
 out << "SCALARS Analytical_Temperature double 1\n";
 out << "LOOKUP_TABLE default\n";

 // 正確計算解析解
 for(int j = 0; j < analytical_NY; j++) {
 for(int i = 0; i < analytical_NX; i++) {
 double x_pos = (i + 0.5) * analytical_dx; // i從0開始，對應x=0到x=1
```

```
 double y_pos = (j + 0.5) * analytical_dy; // j從0開始，對應y=0到y=1
 double analytical_temp = T_analytical_fixed(x_pos, y_pos);
 out << scientific << setprecision(6) << analytical_temp << "\n";
 }
 }

 out.close();
 cout << "VTK document output: " << name.str() << endl;
}


int main() {
 for(int grid_idx = 0; grid_idx < Nx.size(); grid_idx++) {
 cout << "\n========================================" << endl;
 cout << "Grid size: " << Nx[grid_idx]+1 << "x" << Ny[grid_idx]+1 << endl;

 NX = Nx[grid_idx];
 NY = Ny[grid_idx];
 n = NX * NY;
 // 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
 dx = 1.0 / (NX );
 dy = 1.0 / (NY );

 cout << "Grid spacing: dx = " << dx << ", dy = " << dy << endl;

 // 重新調整向量大小
 a.assign(n+2, vector<double>(n+2, 0.0));
 b.assign(n+2, 0.0);
 x.assign(n+2, 0.0);
 x_old.assign(n+2, 0.0);
 T.assign(NX, vector<double>(NY, 0.0));

 cout << "Program execution started...." << endl;
 steadystate = false;
 initial(a, b, n);

 for(G = 0; G < max_G; G++) {
 Jacobi(a, b, x, n);

 if(G % 1000 == 0) { // 每1000次輸出一次
 cout << "Iteration = " << G;
 cout << ", Convergence error = " << scientific << setprecision(3) << maxerror;
 cout << ", L1 error = " << scientific << setprecision(3) << L1sum << endl;

 if(G % 5000 == 0) { // 每5000次輸出VTK文件
 output(G);
 }
 }

 if(G > 100 && maxerror < tolerance) {
 steadystate = true;
```

```
cout << "Steady state reached, temperature field converged!!" << endl;
cout << "Final iteration: " << G << ", Convergence error: " << maxerror << endl;
cout << "Final L1 error: " << L1sum << endl;
FinalL1error[grid_idx] = L1sum;
break;
}
}

if(!steadystate) {
cout << "Maximum iteration reached, but steady state not achieved!" << endl;
cout << "Final convergence error: " << maxerror << endl;
cout << "Final L1 error: " << L1sum << endl;
FinalL1error[grid_idx] = L1sum;
}

output(G);
cout << "Grid size " << NX << "x" << NY << " computation completed" << endl;
cout << "=====================================" << endl;
}
output_gnuplot_data();
output3();
cout << "\nAll computations completed!" << endl;

return 0;
}
```