```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);
}
```

```
//初始化迭代矩陣
void initial(vector<vector<double> >& a, vector<double>& b, int n) {

    double A = 1.0 / (12.0 * dx * dx);
    double B = 1.0 / (12.0 * dy * dy);

    //二階偏導數的四階精度中心差分的係數
    double c_center = 30.0;  // 中心係數
    double c_near = -16.0;   // 相鄰係數
    double c_far = 1.0;      // 遠方係數

    // 初始化矩陣
    for(int i = 0; i <= n+1; i++) {
        for(int j = 0; j <= n+1; j++) {
            a[i][j] = 0.0;
        }
        b[i] = 0.0;
        x[i] = 0.0;
        x_old[i] = 0.0;
    }

    cout << "Setting boundary conditions..." << endl;

    // 左邊界條件 - 直接賦值
    // 左邊界 (i=1)
    for(int j = 1; j <= NY; j++){
        int idx = (j-1)*NX + 1;
        a[idx][idx] = 1.0;
        b[idx] = T_left;
    }

    // 右邊界 (i=NX)
    for(int j = 1; j <= NY; j++){
        int idx = (j-1)*NX + NX;
        a[idx][idx] = 1.0;
        b[idx] = T_right;
    }

    // 下邊界 (j=1)
    for(int i = 1; i <= NX; i++){
        int idx = i;
        a[idx][idx] = 1.0;
        b[idx] = T_bottom;
    }

    // 上?界 (j=NY)
    for(int i = 1; i <= NX; i++){
        int idx = (NY-1)*NX + i;
        a[idx][idx] = 1.0;
        b[idx] = T_up(i);
    }

    cout << "Setting interior points with consistent 4th order scheme..." << endl;
```

```
    // ?部點 - 使用精確四階差分格式
    // 只有真正的內部點 (i=3 to NX-2, j=3 to NY-2) 使用四階格式
    for(int j = 3; j <= NY-2; j++) {
        for(int i = 3; i <= NX-2; i++) {
            int idx = (j-1)*NX + i;

            // 拉普拉斯算子: (d2/dx2 + d2/dy2)u = 0
            // x方向: u_{i-2} - 16*u_{i-1} + 30*u_i - 16*u_{i+1} + u_{i+2}
            // y方向: u_{j-2} - 16*u_{j-1} + 30*u_j - 16*u_{j+1} + u_{j+2}

            a[idx][idx] = A * c_center + B * c_center;        // 中心點
            a[idx][idx-1] = A * c_near;                       // 西點
            a[idx][idx+1] = A * c_near;                       // 東點
            a[idx][idx-2] = A * c_far;                        // 西西點
            a[idx][idx+2] = A * c_far;                        // 東東點
            a[idx][idx-NX] = B * c_near;                      // 南點
            a[idx][idx+NX] = B * c_near;                      // 北點
            a[idx][idx-2*NX] = B * c_far;                     // 南南點
            a[idx][idx+2*NX] = B * c_far;                     // 北北點

            b[idx] = 0.0;
        }
    }

    //
    //第二層邊界點 (i=2, i=NX-1, j=2, j=NY-1) 使用二階中心差分
    // 這些點無法使用四階格式，因為缺少足夠的鄰近點
    for(int j = 2; j <= NY-1; j++) {
        for(int i = 2; i <= NX-1; i++) {
            // 跳過已處理的內部點
            if(i >= 3 && i <= NX-2 && j >= 3 && j <= NY-2) continue;

            int idx = (j-1)*NX + i;

            // 使用二階中心差分
            double A2 = 1.0 / (dx * dx);
            double B2 = 1.0 / (dy * dy);

            a[idx][idx] = -2.0 * A2 - 2.0 * B2;
            a[idx][idx-1] = A2;     // 西點
            a[idx][idx+1] = A2;     // 東點
            a[idx][idx-NX] = B2;    // 南點
            a[idx][idx+NX] = B2;    // 北點
            b[idx] = 0.0;
        }
    }

    cout << "Matrix initialization completed." << endl;
    cout << "Total equations: " << n << endl;
    cout << "Interior 4th-order points: " << (NX-4)*(NY-4) << endl;
}
```

```

//超鬆弛迭代法_自適應性鬆弛因子
void SOR(vector<vector<double> >& a, vector<double>& b, vector<double>& x, int n) {
    // 自適應松弛因子
    double omega;
    if(NX <= 21) omega = 0.5;
    else if(NX <= 41) omega = 0.8;
    else omega = 1.2;

    //保存舊的解
    for(int k = 1; k <= n; k++) {
        x_old[k] = x[k];
    }

    // SOR迭代
    for(int k = 1; k <= n; k++) {
        if(fabs(a[k][k]) < 1e-15) continue; // 跳過奇異矩陣

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

    // 計算最大收斂誤差
    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
    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) {
    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 << "FDM_diffusion_2D_improved_" << NX << "x" << NY << "_" << setfill('0') << setw(6) << m << ".vtk";
    ofstream out(name.str().c_str());

    out << "# vtk DataFile Version 3.0\n";
    out << "steady_diffusion_2D_improved\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";

    out << "SCALARS Temperature double 1\n";
    out << "LOOKUP_TABLE default\n";
    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;
}

void output_gnuplot_data() {
    bool valid_data = true;
    for(int grid_idx = 0; grid_idx < 4; grid_idx++) {
        if(FinalL1error[grid_idx] <= 0 || isnan(FinalL1error[grid_idx]) || isinf(FinalL1error[grid_idx])) {
            cout << "Warning: Invalid L1 error for grid " << grid_idx << ": " << FinalL1error[grid_idx] << endl;
            valid_data = false;
        }
    }

    if(!valid_data) {
        cout << "Cannot generate convergence analysis due to invalid data." << endl;
        return;
    }

    ofstream data_file("grid_convergence_4order_improved.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]-1);
        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_4order_improved.dat" << endl;

    // ?性回?
    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]-1));
        double y = log(FinalL1error[grid_idx]);

        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;

    ofstream gnuplot_file("plot_convergence_4order_improved.plt");
    gnuplot_file << "set terminal png enhanced size 800,600" << endl;
    gnuplot_file << "set output 'grid_convergence_4order_improved.png'" << endl;
    gnuplot_file << "set title 'Improved 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;

    double x_min = log(1.0 / (Nx[3]-1));
    double x_max = log(1.0 / (Nx[0]-1));
    double y_ref = log(FinalL1error[1]);
    double x_ref = log(1.0 / (Nx[1]-1));

    gnuplot_file << "f(x) = " << slope << " * x + " << intercept << endl;
    gnuplot_file << "g(x) = 4.0 * (x - " << x_ref << ") + " << y_ref << endl;

    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;
    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 '4th Order Theory (slope = 4.0)'" << endl;

```

```cpp
    gnuplot_file.close();
    cout << "Gnuplot script output: plot_convergence_4order_improved.plt" << endl;

    cout << "\n=== Improved Grid Convergence Analysis ===" << endl;
    cout << "Linear regression results:" << endl;
    cout << "Slope = " << fixed << setprecision(3) << slope << " (理?值 4.0)" << endl;
    cout << "Order of accuracy = " << fixed << setprecision(3) << slope << endl;
}

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

        NX = Nx[grid_idx];
        NY = Ny[grid_idx];
        n = NX * NY;
        dx = 1.0 / (NX-1);
        dy = 1.0 / (NY-1);

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

        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 with improved 4th order scheme...." << endl;
        steadystate = false;
        initial(a, b, n);  //初始化

        for(G = 0; G < max_G; G++) {
            SOR(a, b, x, n);  // 使用改進的SOR

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

                if(G % 5000 == 0) {
                    output(G);
                }
            }

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

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

        if(!steadystate) {
            cout << "Maximum iteration reached!" << 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();
    cout << "\nAll computations completed!" << endl;

    return 0;
}
```