

CFD Project 01 Update

By: Arshia Saffari

This update is actually already used in the original project to solve the 400 by 400 grid. Gauss-Seidel method is now modified to only multiply the potentially non-zero elements of the sparse matrix to their respective unknown values. The performance gain is huge.

```

1: void HeatEquationSolver::createEquationMatrix() {
2:     // 5 elements per row
3:     for (size_t i = 0; i < m_eqMatDims; i++)
4:         m_eqKnownVec.push_back(0);
5:     std::vector<Eigen::Triplet<double> > Vec;
6:     Vec.reserve(m_eqMatDims * 5);
7:     size_t N = m_nodeCount.ycount - 2;
8:     size_t M = m_nodeCount.xcount - 2;
9:     double L = m_computeDomain.xrange.second - m_computeDomain.xrange.first;
10:    double H = m_computeDomain.yrange.second - m_computeDomain.yrange.first;
11:    double gamma = L * (M + 1) / (H * (M + 1));
12:    //for (size_t i = 0; i < M * N; i++)
13:    //    for (size_t j = 0; j < M * N; j++)
14:    //        //m_eqMat.A[i][j] = 0;
15:    //        //m_eqMat.A[i][j] = 0;
16:    for (size_t i = 0; i < M; i++) {
17:        for (size_t j = 0; j < N; j++) {
18:            // for node i,j:
19:            if (j == 0) {
20:                //m_eqMat.b[j * M + i] -= 0 * gamma;
21:                //m_eqMat.A[j * M + i][(j + 1) * M + i] = 1 * gamma;
22:                m_eqKnownVec[j * M + i] -= 0 * gamma;
23:                Vec.push_back(Eigen::Triplet<double>(j * M + i, (j + 1) * M + i, 1 * gamma));
24:            }
25:            else if (j == N - 1) {
26:                //m_eqMat.b[j * M + i] -= 1 * gamma;
27:                //m_eqMat.A[j * M + i][(j - 1) * M + i] = 1 * gamma;
28:                m_eqKnownVec[j * M + i] -= 1 * gamma;
29:                Vec.push_back(Eigen::Triplet<double>(j * M + i, (j - 1) * M + i, 1 * gamma));
30:            }
31:            else {
32:                //m_eqMat.A[j * M + i][(j - 1) * M + i] = 1 * gamma;
33:                //m_eqMat.A[j * M + i][(j + 1) * M + i] = 1 * gamma;
34:                Vec.push_back(Eigen::Triplet<double>(j * M + i, (j - 1) * M + i, 1 * gamma));
35:                Vec.push_back(Eigen::Triplet<double>(j * M + i, (j + 1) * M + i, 1 * gamma));
36:            }
37:            if (i == 0) {
38:                //m_eqMat.b[j * M + i] -= 0;
39:                //m_eqMat.A[j * M + i][j * M + i + 1] = 1;
40:                m_eqKnownVec[j * M + i] -= 0;
41:                Vec.push_back(Eigen::Triplet<double>(j * M + i, j * M + i + 1, 1));
42:            }
43:            else if (i == M - 1) {
44:                //m_eqMat.b[j * M + i] -= 0;
45:                //m_eqMat.A[j * M + i][j * M + i - 1] = 1;
46:                m_eqKnownVec[j * M + i] -= 0;
47:                Vec.push_back(Eigen::Triplet<double>(j * M + i, j * M + i - 1, 1));
48:            }
49:            else {
50:                //m_eqMat.A[j * M + i][j * M + i - 1] = 1;
51:                //m_eqMat.A[j * M + i][j * M + i + 1] = 1;
52:                Vec.push_back(Eigen::Triplet<double>(j * M + i, j * M + i - 1, 1));
53:                Vec.push_back(Eigen::Triplet<double>(j * M + i, j * M + i + 1, 1));
54:            }
55:            //m_eqMat.A[j * M + i][j * M + i] = -4;
56:            Vec.push_back(Eigen::Triplet<double>(j * M + i, j * M + i, -4));
57:        }
58:    }
59:    m_eqSparseMat.resize(N * M, N * M);
60:    m_eqSparseMat.setFromTriplets(Vec.begin(), Vec.end());
61: }

```

Basically, looking at the matrix generator code there are only 5 elements per row at best. These are located at (i, i) , $(i, i + 1)$, $(i, i - 1)$, $(i, i + M)$, $(i, i - M)$. The parameter M is passed to solver and instead of multiplying each element (which for a 400 by 400 grid is $398 \times 398 = 158404$ elements in each of the 158404 rows of the matrix of which, 158403 elements are multiplied to their respective unknown vector value, summed and subtracted from respective known vector element and divided by $A(i, i)$) only 4 elements (depending on the boundary condition they might still be 0 but it's not worth the effort to identify them) are multiplied. This approach reduces the time considerably.

```

bool HermiticSolver::solveWithSuccessiveRelax() {
    const size_t n = a_mat.rows(); //Iteration limit
    size_t k = 0; //Iteration counter
    double InfNorm = std::numeric_limits<double>::max(); //Just to make sure error > tolerance before the first iteration
    double AbsNorm = std::numeric_limits<double>::max(); //Just to make sure error > tolerance before the first iteration
    double hNorm = std::numeric_limits<double>::max(); //Just to make sure error > tolerance before the first iteration
    double Sum = 0;

    std::vector<double> oldH(n,0); // to store values from last iteration.
    std::vector<double> oldH2(n,0);
    std::vector<double> oldH3(n,0);
    std::cout << "Entering the while loop";
    auto t0 = std::chrono::high_resolution_clock::now();
    while (hNorm > a_tolerance && k < a_maxIterations)
    {
        auto t1 = std::chrono::high_resolution_clock::now();

        for (size_t i = 0; i < n; i++) {
            Sum = 0;
            if ((i%3) + 1 < a_n.size()) {
                Sum += a_mat.coeff(i, i + 1) * a_x[i + 1];
            }
            if ((i%3) - 1 > 0) {
                Sum += a_mat.coeff(i, i - 1) * a_x[i - 1];
            }
            if ((i%3) + n < a_n.size()) {
                Sum += a_mat.coeff(i, i + n) * a_x[i + n];
            }
            if ((i%3) - n > 0) {
                Sum += a_mat.coeff(i, i - n) * a_x[i - n];
            }
        }

        //endifor k
        // for (size_t j = 0; j < i; j++)
        //     Sum += a_mat.coeff(i, j) * a_x[j];
        // for (size_t j = i + 1; j < n; j++)
        //     Sum += a_mat.coeff(i, j) * a_x[j];
        //endifor j

        //On x = oldH
        oldH[i] = oldH[i]; //oldH2 oldH3
        oldH[i] = oldH[i]; //oldH2 oldH3
        a_x[i] = (a_n[i] - Sum) / a_mat.coeff(i, i);
        a_x[i] = a_n[i] + a_x[i] * (1 - a_n[i]) * oldH[i];
    }

    InfNorm = 0;
    AbsNorm = 0;
    hNorm = 0;

    for (size_t i = 0; i < n; i++) {
        double deltax = abs(oldH[i] - a_x[i]);
        AbsNorm += deltax;
        hNorm += deltax/deltax;
        InfNorm = std::max(InfNorm, deltax);
    }
    hNorm = pow(hNorm, 1.0 / 2.0);

    auto t2 = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> std::millis totalTime = t2 - t1;
    std::chrono::duration<double> std::millis timeFillNew = t2 - t0;

    statistics << k << " ";
    statistics << timeFillNew.count() << " ";
    statistics << hNorm << "\n";

    //endif:cout << "it: " << k << "\n";
    //endif:cout << "filling new: " << InfNorm << "\n";
    //endif:cout << "absolute new: " << AbsNorm << "\n";
    //endif:cout << "relative new: " << hNorm << "\n";
    //endif:cout << "log time vlt" << logTime.count() << "\n";
    k++;
}

double ru = 0;
double su = 0;
for (size_t i = 0; i < n; i++) {
    //Converges
    ru += (a_x[i] - oldH[i]) / (oldH[i] - oldH2[i]);
    su += (oldH[i] - oldH2[i]) / (oldH[i] - oldH3[i]);
}
double q = (log10(ru) / log10(su));
std::ofstream outputStatistic;
std::ofstream filename = "File" + std::to_string(n_u);
Filename += ".txt";
outputStatistic.open(filename, std::ios_base::out);
outputStatistic << statistics.rdbuf();
std::cout << "average" << a_n << "\n";
std::cout << "q" << q << "\n";
outputStatistic.close();

if (k < a_maxIterations)
    return false;
return true;
}

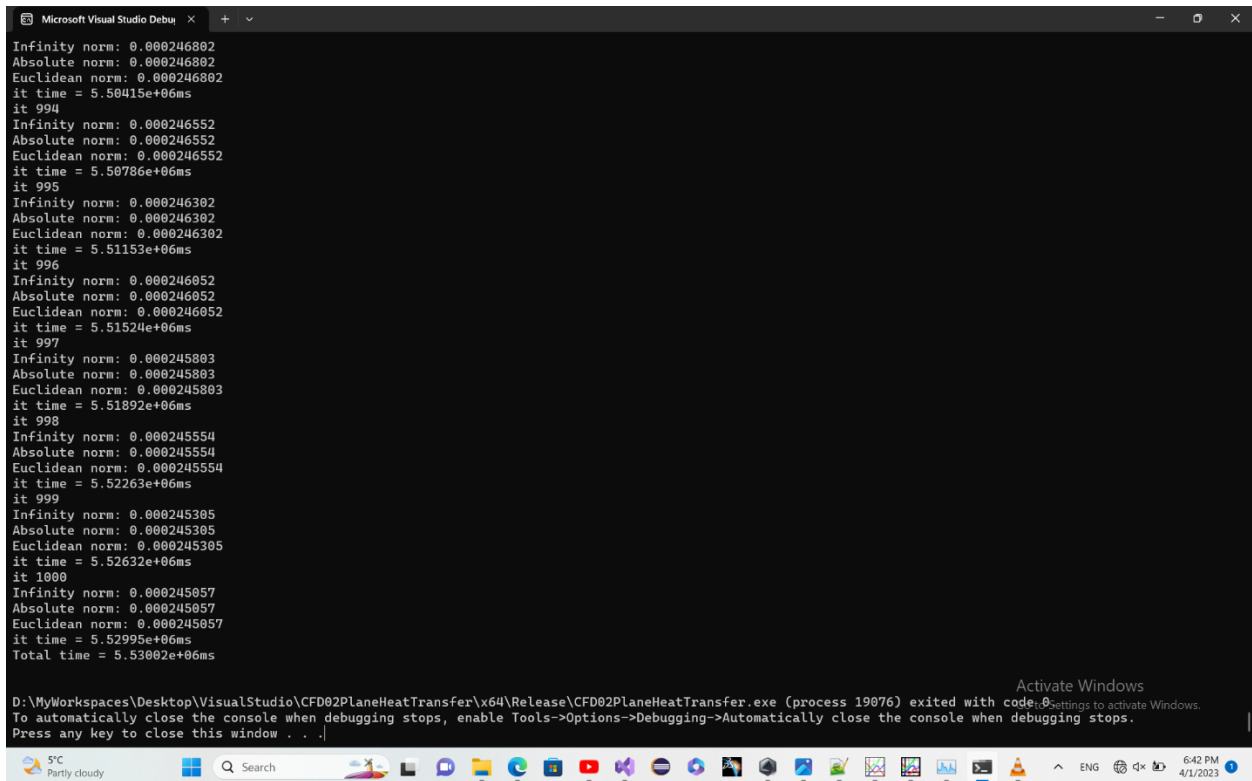
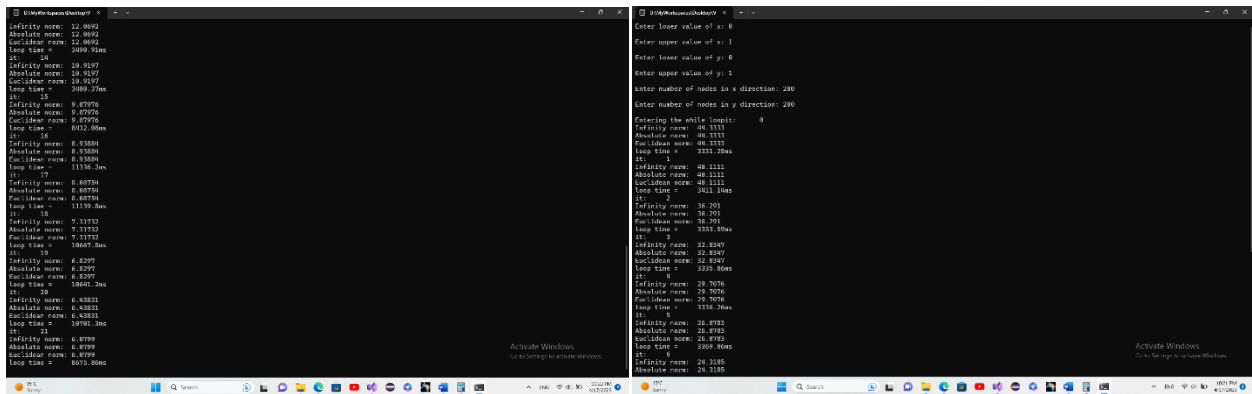
```

The new solver codes.

Performance comparison:

Old solver:

A simple 200 by 200 matrix takes a minimum of about 3331.28ms per iteration which grows up significantly at times even as much as 11139.8ms.



It took about 1.5 hours to complete with a accuracy of 1e-4.

New code:

Replacing the old method:

```
//for (size_t j = 0; j < i; j++)
// Sum += m_sMat.coeff(i, j) * m_x[j];
//for (size_t j = i + 1; j < n; j++)
// Sum += m_sMat.coeff(i, j) * m_x[j];
```

With newer one

```

if ((int)i + 1 < m_x.size()) {
    Sum += m_sMat.coeff(i, i + 1) * m_x[i + 1];
}
if ((int)i - 1 > 0) {
    Sum += m_sMat.coeff(i, i - 1) * m_x[i - 1];
}
if ((int)i + M < m_x.size()) {
    Sum += m_sMat.coeff(i, i + M) * m_x[i + M];
}
if ((int)i - M > 0) {
    Sum += m_sMat.coeff(i, i - M) * m_x[i - M];
}

```

This loop takes about 0.5 to 0.7ms to complete on iteration and takes about 6 seconds to complete 2526 iterations which is much more than what was previously possible and is much more accurate with the error being less than 1e-6. The bottleneck at this point is consul print.

```

Microsoft Visual Studio Debug
Euclidean norm: 1.04623e-06
loop time = 0.648ms
it: 2520
Infinity norm: 1.04117e-06
Absolute norm: 1.04117e-06
Euclidean norm: 1.04117e-06
loop time = 0.6685ms
it: 2521
Infinity norm: 1.03614e-06
Absolute norm: 1.03614e-06
Euclidean norm: 1.03614e-06
loop time = 0.717ms
it: 2522
Infinity norm: 1.03113e-06
Absolute norm: 1.03113e-06
Euclidean norm: 1.03113e-06
loop time = 0.7646ms
it: 2523
Infinity norm: 1.02614e-06
Absolute norm: 1.02614e-06
Euclidean norm: 1.02614e-06
loop time = 0.6158ms
it: 2524
Infinity norm: 1.02118e-06
Absolute norm: 1.02118e-06
Euclidean norm: 1.02118e-06
loop time = 0.7047ms
it: 2525
Infinity norm: 1.01624e-06
Absolute norm: 1.01624e-06
Euclidean norm: 1.01624e-06
loop time = 0.7288ms
it: 2526
Infinity norm: 1.01133e-06
Absolute norm: 1.01133e-06
Euclidean norm: 1.01133e-06
loop time = 0.7817ms
omega= 1.9
q= 1
Total time = 5973.11ms

D:\MyWorkspaces\Desktop\VisualStudio\CFD02PlaneHeatTransfer\x64\Release\CFD02PlaneHeatTransfer.exe (process 33220) exited with code 0
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

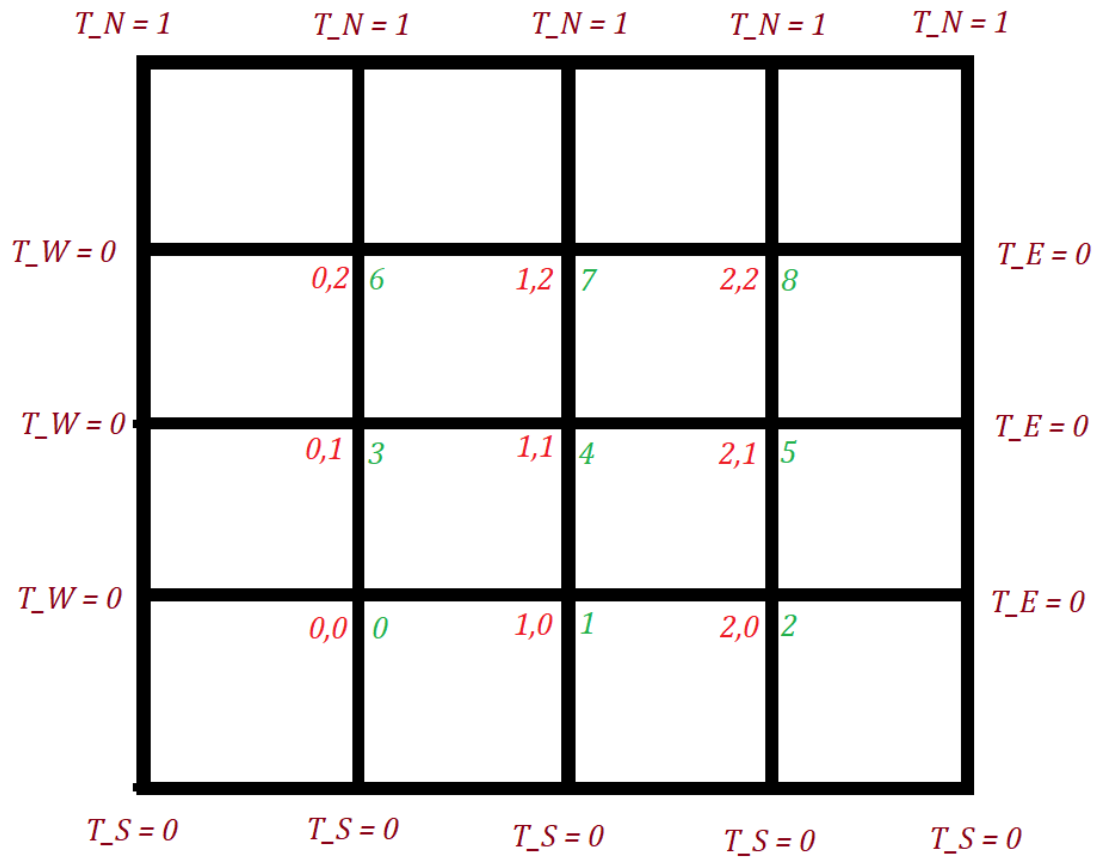
```

Some more talks about the matrix:

```
Microsoft Visual Studio Debu x + v
Enter lower value of x: 0
Enter upper value of x: 1
Enter lower value of y: 0
Enter upper value of y: 1
Enter number of nodes in x direction: 5
Enter number of nodes in y direction: 5
-4,1,0,1,0,0,0,0,0,0, 0
1,-4,1,0,1,0,0,0,0,0, 0
0,1,-4,0,0,1,0,0,0,0, 0
1,0,0,-4,1,0,1,0,0,0, 0
0,1,0,1,-4,1,0,1,0,0, 0
0,0,1,0,1,-4,0,0,1,0, -1
0,0,0,1,0,0,-4,1,0,0, -1
0,0,0,0,1,0,1,-4,1, -1
0,0,0,0,0,1,0,1,-4, -1
Entering the while loopit: 0
Infinity norm: 60.2126
Absolute norm: 60.2126
Euclidean norm: 60.2126
loop time = 0.0093ms
it: 1
Infinity norm: 54.7818
Absolute norm: 54.7818
Euclidean norm: 54.7818
loop time = 0.0018ms
it: 2
Infinity norm: 30.9315
Absolute norm: 30.9315
Euclidean norm: 30.9315
loop time = 0.0046ms
it: 3
Infinity norm: 25.6787
Absolute norm: 25.6787
Euclidean norm: 25.6787
loop time = 0.0008ms
it: 4
Infinity norm: 24.2588
Absolute norm: 24.2588
Euclidean norm: 24.2588

Activate Windows
Go to Settings to activate Windows.
```

This is the sparse matrix generated by `void HeatEquationSolver::createEquationMatrix()` method for a 5 by 5 grid.



The indexing (2D index in red on bottom left of each node and 1D index on bottom right). Sorry for the bad paint job.

```

-4,1,0,1,0,0,0,0,0,0, 0
1,-4,1,0,1,0,0,0,0,0, 0
0,1,-4,0,0,1,0,0,0,0, 0
1,0,0,-4,1,0,1,0,0,0, 0
0,1,0,1,-4,1,0,1,0,0, 0
0,0,1,0,1,-4,0,0,1,0, 0
0,0,0,1,0,0,-4,1,0,-1
0,0,0,0,1,0,1,-4,1,-1
0,0,0,0,0,1,0,1,-4,-1

```

As expected, the equations are

$$\text{Line 0: } -4T_0 + 1T_1 + 0T_2 + 1T_3 + 0T_4 + 0T_5 + 0T_6 + 0T_7 + 0T_8 = 0$$

$$\text{Line 1: } 1T_0 - 4T_1 + 1T_2 + 0T_3 + 1T_4 + 0T_5 + 0T_6 + 0T_7 + 0T_8 = 0$$

$$\text{Line 2: } 0T_0 + 1T_1 - 4T_2 + 0T_3 + 0T_4 + 1T_5 + 0T_6 + 0T_7 + 0T_8 = 0$$

$$\text{Line 3: } 1T_0 + 0T_1 + 0T_2 - 4T_3 + 1T_4 + 0T_5 + 1T_6 + 0T_7 + 0T_8 = 0$$

$$\text{Line 4: } 0T_0 + 1T_1 + 0T_2 + 1T_3 - 4T_4 + 1T_5 + 0T_6 + 1T_7 + 0T_8 = 0$$

$$\text{Line 5: } 0T_0 + 0T_1 + 1T_2 + 0T_3 + 1T_4 - 4T_5 + 0T_6 + 0T_7 + 1T_8 = 0$$

$$\text{Line 6: } 0T_0 + 0T_1 + 0T_2 + 1T_3 + 0T_4 + 0T_5 - 4T_6 + 1T_7 + 0T_8 = -1$$

$$\text{Line 7: } 0T_0 + 0T_1 + 0T_2 + 0T_3 + 1T_4 + 0T_5 + 1T_6 - 4T_7 + 1T_8 = -1$$

$$\text{Line 8: } 0T_0 + 0T_1 + 0T_2 + 0T_3 + 0T_4 + 1T_5 + 0T_6 + 1T_7 - 4T_8 = -1$$

Which seems true as equations 6,7 and 8 are the only ones with -1 in the known vector (from moving +1 temperature in the north node to the RHS.)

Each line is written for each node and all nodes are unknown temperature, the -4 coefficient starts from line 0 and T0 to line 8 and T8.

And the neighboring nodes have coefficients equal to 1.