Final Project: Progress Report 1

CSE 597

Wenliang Sun

Monday, September 24, 2018

Abstract

The convection-diffusion equation is a combination of the diffusion and convection equations, and describes physical phenomena where particles, energy, or other physical quantities are transferred inside a physical system due to two processes: diffusion and convection. Depending on context, the same equation can be called the advectiondiffusion equation, driftdiffusion equation, or scalar transport equation.

In this project, we leverage a program to solve a 2D convection-diffusion equation. We have two different solvers: direct solver and iterative solver. We use LU decomposition and Jacobi methods to implement our project. After that, we compare their performance.

1 Problem of Interest

The mathematical models that reflect the laws governing the movement of soil water and salt are mostly nonlinear partial differential equations. Before the 1960s, analytical methods were used to study the analytical or semi-analytical solutions of equations, but the analytical methods were only applicable to the problem of soil water movement under simple conditions. In order to study soil water and salt transport under more complex conditions, the most effective method at present is to use numerical calculation methods. In recent years, due to the rapid development of industry, the industrial heavy metal emissions have increased, and heavy metals containing polluting gases, polluting liquids and industrial residues have had a huge impact on the environment. The root of the problem lies in finding the source of pollution, and the treatment of the source of pollution can solve the fundamental problem. The transport of heavy metal pollutants in the soil can be attributed to the migration of soil solute. The solution to the transport model of soil solute is to solve the convection-diffusion equation.

The effective numerical solution to the convection-diffusion problem has always been an important research content in computational mathematics. The numerical methods for solving the convection-diffusion equation are mainly finite difference method (FDM), finite element method (FEM), finite volume method (FVM), and finite analytical method. (FAM), Boundary Element Method (BEM), Spectral Method (SM). Actually, there are two common methods to solve this equation: the discontinuity-capturing crosswind-dissipation for the finite element solution of the convection-diffusion equation, a single cell high order scheme for the convectiondiffusion equation with variable coefficients and etc. We don't discuss too much about these methods in this report.

In our project, we are going to solve a 2D convection-diffusion problem. As a very important branch of partial differential equations, convection-diffusion equations have been widely used in many fields, such as fluid mechanics, gas dynamics, etc. Because convective diffusion equations are difficult to obtain analytical solutions through analytical methods, so through various numerical methods to solve the convection-diffusion equation plays an important role in numerical analysis. Our project solves the steady 2D convection-diffusion problem: $\frac{\partial \rho u_i \phi}{\partial x_i} = \frac{\partial(\tau \frac{\partial \phi}{\partial x_i})}{\partial x_i} + q_\phi$.

1.1 Numerical Set-up

For the 2D convection diffusion equation, we assume: x∈[0,1] and y∈[0,1] and $\rho = 100$, $\tau = 0.1$, $q_\phi = 0$, $u_x = x$, and $u_y = -y$. The boundary conditions are ϕ=0 on y = 1, $\phi = \phi$(y) = 1-y on x=0, $\frac{\partial \phi}{\partial x} = 0$ on x = 1, and $\frac{\partial \phi}{\partial y}$ on y = 0.

According the above conditions, we can discretize the problem. We use a small size matrix as an simple example. And then divide the area into a 5*5 matrix as figure 1.

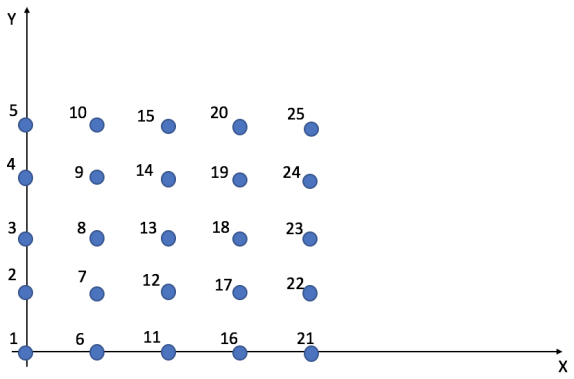

Figure 1: This is an example of a 5*5 matrix

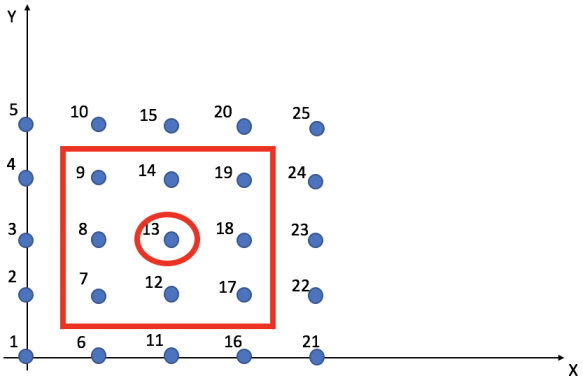

Figure 2: This is an example of an interior node

Firstly, we calculate the interior nodes. In the figure 2, the interior nodes are in the red circle. We use node 13 as an example. The other interior nodes' calculation are same. For the node 13, we leverage the Second-order Central Difference Scheme(CDS2) to solve it. According to the node 8, 18, 12, 14, we can get: $(\rho u_x/2h-\tau/h^2)\phi_{18}+(-\rho u_x/2h-\tau/h^2)\phi_8+4\tau/h^2\phi_{13}+(\rho u_y/2h-\tau/h^2)\phi_{14}+(-\rho u_y/2h-\tau/h^2)\phi_{12}=q_\phi$

In this equation, "h" stand for the distance between two nodes. Through this method, we can calculate all the interior nodes. And we can use the coefficients to build A matrix. There are nine interior nodes, so it could fill in nine lines in A matrix, and the same position of b matrix is "0".

Secondly, we should solve the boundary problem. For this example, there are four boundaries: left, right, up and down. The left boundary is a constant, it is "1-y". So we can fill in the A matrix with "0", and set the node's position in A matrix as "1". The corresponding position in b matrix is "1-y". The up boundary is similar as the left boundary, the difference is that the corresponding position in b matrix is "0".

But the right boundary is different, they are Newman Boundary Condition. We use Second-order Backward Difference Scheme(BDS2) to solve it. This method means that we use the two nodes behind it. We use node 23 as an example. We can get the equation:

$$\frac{\phi_13-4\phi_{18}+3\phi_{23}}{2\Delta x}=f(y)|_{23}$$

Because the $\frac{\partial\phi}{\partial x}=0$ on x = 1, so we can get: $\phi_{13}-4\phi_{18}+3\phi_{23}$=0. And then we fill in A matrix with these coefficients. And the corresponding position of b matrix is "0". In the end, we can generate the A

matrix and b matrix by merging the two parts as above.

2 Solvers

2.1 Direct Solver

In our project, we choose LU decomposition solver as the direct approach. As we learned in class, this type of solver decompose A matrix into L and U matrix. In the real world, there may be lots of calculations to solve the whole problem. So the matrix decomposition method is much more convenient than other methods such as Gauss elimination.

We use the following optimization flags:

- -O2 - Enabling the optimizations while not inducing a significant increasing of compiling time.
- -mavx - Enabling the optimizations for vectorized data to speed-up matrix operations.

According to the experiments, a 100 * 100 matrix takes about 40 seconds for the matrix decomposition. This matrix takes about 100 MB for saving the matrices L and U, and about another 30 MB to save the original matrix A. If the matrix is 5000 * 5000, the matrix could be 2500 times larger than currently constructed. It takes around 100GB memory space. It cannot be accepted. Additionally, the matrix decomposition takes about 200 days!

2.2 Iterative Solver

We use Jacobi method as iterative solver in our project. According to the Dr. Adam's code, we implement our Jacobi solver. Generally speaking, the Jacobi solver is based on the convergence. It also could be used in the future parallel implementations. In my test case, the two solvers seems like have similar time consumption. But according to the algorithm complexity, the iterative solver must be better than the direct solver when the problem scale is huge. For the memory problem, because my the scale of problem is small, so it just costs several kilobytes.

For the above test cases, the iterative solver takes less time than the direct solver. For the random, good guess, all zeros cases, the convergence is satisfying, and guessed initial value is slightly better than zero initial value, which is then better than random initial value. The result is in accordance with the expectation. Because the Jacobi method is too fast, we cannot measure the exact value. Therefore we get the result is around 20 ms. Memory allocated in this procedure is hard to track with the in-program tracking method. The memory allocation for a production problem is still considerably small, for the size of array still grows in O(N), and it around takes 3 MB for a 5000 * 5000 size problem.

3 Solver Comparison

For our certain problem, the iterative method has a better performance. The convergence of iterative methods varies widely among different problems. And the Jacobi iterative method are used to solve large sparse matrices.

For production problems, they are very likely to be constrained by time consumption of direct solver on first run. Therefore before the memory exhausts, it is very likely that the computation time becomes unacceptable. For iterative solver, the major constraint is time (iteration cycle numbers), for the memory consumption is of the same order of magnitude of the consumption for saving an input field. To make the solvers parallel, the partial-pivoting LU-decomposition method might need modifications in order to run on different cores or even nodes.

4 Discussion and Conclusions

In this project, we describe the basic idea of the 2D convection-diffusion equation, and then solve the equation by direct solver and iterative solver. We also have discussed the method of discretization, the construction of A matrix and b matrix and the comparison of these two methods.

We also test the performance of these two solvers. For this problem, the result shows that the Jacobi solver is better than the other one on time and spatial consumption. According to our experiment, we can see the direct solver waste lots of time on matrix decomposition. Additionally, for a large size problem, the direct solver is not a good choice. For the Jacobi solver, our result shows its efficient and effective. It also can be modified to fit the parallel computation methods. Therefore, the direct solvers time consumption increase rapidly and it also occupies large memory. However, the Jacobi solvers memory consumption is adequate, the performance depends on the numbers of iterations.

Appendices

A Acknowledgements

My mathematics and C abilities are not very well. The last time I solved PDE was about 6 years ago. And I use Java several years, my program are translated from Java to C. So this project would not have been possible without the support of Dr. Adam Lavely, Dr. Christopher Blanton and my classmate Yueze Tan. I am especially indebted to my friends Tianyuan Wei and Xingzhao Yun. Tianyuan Wei is an Earth and Mineral Science student, he gave me his class notes and taught me how to solve the PDE step by step; Xingzhao Yun is good at C and CPP, he taught me the basic syntax. They worked actively to provide me the help to complete the Ax = b problem.

B Code

- LU.cpp: Using LU method to solve the equations. According to the comments, modify the input before you use it.
- Jacobi.cpp: Using Jacobi method to solve the equations. According to the comments, modify the input before you use it.
- GetMatrix.cpp: Using this class to generate matrix A and b. According to the comments, modify the input before you use it.
- Our code run on the log-on node in ACI.

  All files are on our github, the address is https://github.com/William0617/CSE597HW1. These above files can be compiled by the command "gcc filename.cpp -o filename". And all the output will be showed on the terminal.

C Licensing and Publishing

We select GNU license for this project. Because the derivatives of all materials using this agreement, whether modified or reproduced, must also use the GNU Free Documentation Agreement certificate. Materials using this agreement may be used for commercial purposes, but any person willing to comply with the agreement must be allowed to further modify or distribute the material under the agreement. The detailed information of the license could be seen in the license.txt and gpl.txt in the root folder of this project.

Currently, the report is published in GitHub website. Because GitHub is a managed platform for open source and proprietary software projects. We hope more and more people see our works and propose some suggestion to help us make our project better.