Final Project: Progress Report 1

CSE 597

Weiming Hu

Monday, September 24, 2018

Abstract

Analog Ensemble is a highly parallelizable and scalable technique to generate probabilistic forecasts using historical deterministic predictions and the corresponding observations. Since the year of 2013 when it is first brought up, it has been successfully applied to short-term wind and temperature forecasts and also to spatial and temporal down-scaling of other numeric weather prediction models, like the Global Forecast System. However, most of the studies focused on the application of the technique, rather than how to improve and complete the technique. Actually, problems have already emerged when the Analog Ensemble technique is applied to a gridded model output. Because of the intrinsic feature of the search algorithm, the produced forecasts loses the spatial continuity that is usually present in a realistic physical world. To get a better understanding of the problem, a deeper inspection into the ensemble is needed.

This study proposes a matrix approach to analyze the contribution of individual ensemble member to the overall forecast distribution by setting up and solving a linear system. The results can be used to gain better understanding of ensemble members at different locations. The study also has a computational focus that it looks at different solvers and implementation, and tries to optimize the problem solving process.

1 Problem of Interest

1.1 General Overview

Weather forecasts were viewed as intrinsically deterministic before the early 1990s [2]. Although numerical weather models are still routinely run today to generate deterministic values for weather predictions, however, probabilistic forecasts have shown greater advantage in various fields. Forecasts associated with probabilities allow interpreters and data analyzers to build utility functions to derive weather-related economic values, for example, power generation and load prediction, and weather risk assessment. Probabilistic forecasts simply provide end users with more information on the forecast uncertainty, therefore receiving more preference over deterministic predictions.

Ensemble modeling is one of the most popular techniques to generate probabilistic forecasts. Generally, ways to generate probability information can be divided up into either by running a single numerical weather model for multiples times with perturbed initialization, or by using different model physics. This Monte Carlo style simulation addresses the chaotic nature of the atmosphere presented [5] which can be associated with the imperfection in model instantiation and parameter propagation.

Meanwhile, The transformation from deterministic predictions to ensemble forecasts and probabilistic forecasts poses challenges on model verification. How to evaluate a probabilistic forecast then becomes a critical question. Traditional verification metrics, for example, contingency table for categorical predictions and Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) for continuous predictions, are no longer applicable for probabilistic forecast verification. Although post processing can be done to collapse a probabilistic forecast to a deterministic prediction, important information on model uncertainty will be lost after the practice. A set of well-established metrics include but are not limited to the rank histogram, the Continuous Rank Probabilistic Score (CRPS), and the Brier score [3].

The ensemble modeling technique used in this project is the Analog Ensemble (AnEn) technique [1]. AnEn is a data-driven technique to generate probabilistic forecasts using a set of historical deterministic predictions and the corresponding observations or the model analysis field. To generate a future probabilistic forecast, the technique first takes the future deterministic prediction from a numerical weather model and looks for the most similar historical predictions in the same model. The similarity is defined using a weighted multi-variate distance function. After the most similar historical predictions are found, the corresponding observations are taken to form the forecast ensemble. This technique has been successfully applied to surface wind and temperature forecasts, and have been found to be more computationally efficient than conventional ensemble modeling techniques because it does not require additional model simulation.

However, because numerical weather models are never perfect, the ensemble members will never be perfect to the observation as well. Verification of AnEn therefore becomes an important research question. This

project studies a verification technique of spatial ensemble forecasts using a matrix approach. AnEn has been successfully applied to gridded model simulation, but forecast results have shown different level of spatial discontinuity [6]. This is partly due to the practice that AnEn are generated for each spatial grid point, and the spatial correlation is not fully preserved. This verification technique of spatial ensemble forecasts can be used to quantify and diagnose the model prediction skills, for example, under-/over- predictions, as well as the spatial discontinuity of the probabilistic forecasts introduced by the AnEn technique.

1.2 Scientific Question and Merit

The goal of this project is to study a verification technique for spatial ensemble forecasts generated from the AnEn and to develop an efficient solver for the problem. Building upon the work done by [6], the project aims to improve the understanding of the contribution of each ensemble member from the AnEn. The knowledge plays an important role when the AnEn is applied to higher resolution model output and when probability information should be kept in the results for further references. Although Sperati and etc. have proposed Schaake Shuffle as a useful technique to reconstruct the spatial continuity, whether this method produces the optimal condition is unknown, and the actual practice of Schaake Shuffle still involves a lot with randomness, therefore the result quality not being guaranteed.

Conventionally, all ensemble members are used to generate distribution and probability information without inspecting which member might be more helpful in approximating the "true" distribution. Ensemble members are usually assigned with equal weights when generating probabilistic information. However, different ensemble members should vary in its predictive ability. For example, places that are closer to each other should have a similar trend in air temperature than other places that are further away. The proposed method is therefore designed to quantify the differences among various ensemble members.

1.3 Linear System Set-up

The problem can be represented by the following equation:

$$\begin{bmatrix} — & F_1^n & — \\ — & F_2^n & — \\ \vdots & \vdots & \vdots \\ — & F_i^n & — \\ \vdots & \vdots & \vdots \\ — & F_m^n & — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} O_1 \\ O_2 \\ \vdots \\ O_i \\ \vdots \\ O_m \end{bmatrix}$$

where,

1. m is the number of spatial grid points;
2. n is the number of members in each ensemble;
3. F_i^n is the ith ensemble forecast vector with n members;
4. $\{x_1, x_2, ..., x_n\}$ is the verification metric vector for ensemble forecasts;
5. O_i is the ith observation;

On the left side of the equation, matrix F is the spatial forecast ensemble. Each line represent a forecast ensemble at a spatial location, or at a spatial grid point. The number of grid points is determined by the spatial resolution of the underlying numeric weather model being used which reaches to 262,792 grid points in the case of North American Mesoscale Forecast System (NAM) with a 11-km spatial resolution. Each value in a row is a ensemble member value. The number of the ensemble members can range from tens to

hundreds depending on the size of the historical predictions. Vector $\vec{O}$ is the corresponding observations or model analysis. Vector $\vec{x}$ is the verification metric of the spatial forecast ensemble. It is assumed that observations come from a certain unknown distribution, and the forecast ensemble should be a sufficiently "good" approximation to the realistic unknown distribution. If the ensemble is a perfect ensemble, each ensemble member should be equally good comparing to the other ensemble members in a spatial context. Therefore, the coefficient $\vec{x}$ should place similar weights to all the ensemble members, rather than some ensemble members having a higher co-efficiency than others. In reality, because bias exists and error propagates, the actual forecast ensemble might be under- or over- predicting. And the coefficient $\vec{x}$ can be used to characterize it.

The matrix F should be a dense matrix because the most part of the matrix will be filled up with values. There will be cases that null values might exist, but it should be not be a very common situation. The matrix F might be a "tall" matrix in a sense that the number of spatial grid points will largely surpass the number of forecast ensemble members.

The size of the matrix varies dramatically based on the size of ensembles and the number of grid points. In the case of NAM, there are 262 792 grid points in total with a 11 km resolution. The size of ensembles is typically decided by the size of historical forecasts used in to generate the ensemble. A good practice is the square root of the size of historical forecasts. Therefore, the size of ensembles is estimated to be between dozens to hundreds. Although the dense matrix can be made square, it will be a dense narrow matrix in most cases.

2 Solvers

2.1 Direct Solver

2.1.1 Justification

There are many ways of solving this linear problem. Three common methods are shown below:

There are three popular ways to solve a linear system: 1)Cramer's Rule; 2) Gaussian Elimination; 3)LU Decomposition. The Cramer's Rule is the most inefficient method out of the three. For a linear system with n equations and n unknowns, the Cramer's Rule requires to solve $n + 1$ different determinants assuming all the determinants exist. However, calculating determinants should be avoided in all efforts because it is a very computationally expensive task whose computational complexity is estimated to be polynomial[4]. Gaussian Elimination and LU Decomposition are usually integrated to solve multiple linear systems with higher efficiency. Calculating determinants can be avoided in these methods by using forward elimination and backward substitution. LU Decomposition involves one more step, the forward substitution, than Gaussian Elimination, leading to the result that LU Decomposition might take longer time to solve one linear system than Gaussian Elimination. However, Gaussian Elimination modifies the coefficient matrix and the right-hand vector internally within each loop, making it impossible to save the transformation and the factorization information of the coefficient matrix. When presented with problems of multiple linear systems where the coefficient matrix does not change and the right-hand vector changes, LU Decomposition has the advantage to save the factorization and reuse the factor matrices for different right-hand vectors. Complexity-wise, Gaussian Elimination is $\mathcal{O}(nm^3)$ and LU Decomposition is $\mathcal{O}(m^3 + nm^2)$ where n is the number of linear systems to solve and m is the size of the problem.

Considering that linear systems will have different coefficient matrices and presumably not many benefits can be gained from using the LU Decomposition method, Gaussian Elimination is selected as the solver in this project. Considering that Gaussian Elimination only works with square matrices but the coefficient matrices dealt with here are usually non-square matrices, the direct solver method is extended to Normal Equation technique where the non-square matrix can be transformed to be square. The Normal Equation can also be referred to as the Ordinary Least Square Regression when there are more equations than the number of unknowns.

The Normal Equation can be expressed as the following: For a linear system $Ax = b$ where A is the

coefficient matrix, b is the output vector, and x is the unknown, we have a solution $x = A^t \times (A \times A^t)^{-1} \times b$, where A^t is the transpose of the matrix A.

2.1.2 Optimization Flags

A list of optimization flags for consideration includes: 1) $-O0$; 2) $-O1$; 2) $-O2$; 3) $-O3$; t) $-Ofast$. To decide the best optimization flag out of the five, the direct solver has been compiled and tested with different optimization flags independently. The solvers have been tested on solving a 500-by-500 coefficient matrix on a MacBook Air. The machine model is an early 2015 with 2 2.2. GHz Inter i7 cores and one 8 GB 1 600 MHz DDR3 memory card. The programs are compiled using the Clang compiler, and each test has been repeated for 10 times for statistical significance. The correctness has been verified for all tests in advance.

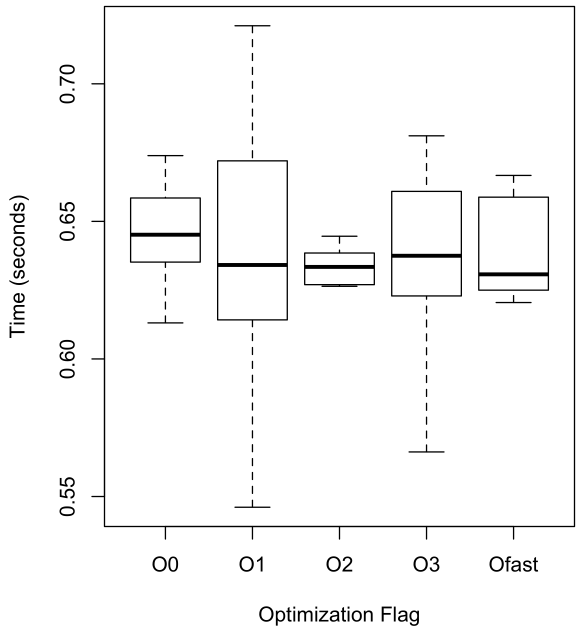

Figure 1: Execution time of the direct solver with different optimization flags.

Figure 1 shows the execution time for the direct solver with different optimization flags. The optimization flag $-Ofast$ generates the lowest averaged execution time. However, the averaged execution time generated by using the flag $-O2$ is very close to to the lowest time, and its spread is smaller than using $-Ofast$. As mentioned in the class materials, $-Ofast$ is the most aggressive flag for code optimization, and it rips of correctness check in precision which may leads to wrong results. Therefore, $-O2$ is chosen to be the optimization flag for the direct solver.

2.1.3 Profiling

When compiled with the macro definition *-DPROFILE_TIME*, the profiling code will be added to the program and the extra standard output will be provided when the program terminates.

Time The following output is for time profiling of the code. The direct solver is used to solve a 500-by-500 coefficient matrix. The program runs for 0.531 seconds in wall time. The first section of the standard output shows the forward and backward elimination time used by the matrix inverse function.

```
-----------------------------------
Time profiling for the matrix inverse function:
Forward elimination: 0.04851s (66.59%)
Backward elimination: 0.02434s (33.41%)
Inverse function total: 0.07286s (1)
-----------------------------------
Total time for the direct method: 0.531s
```

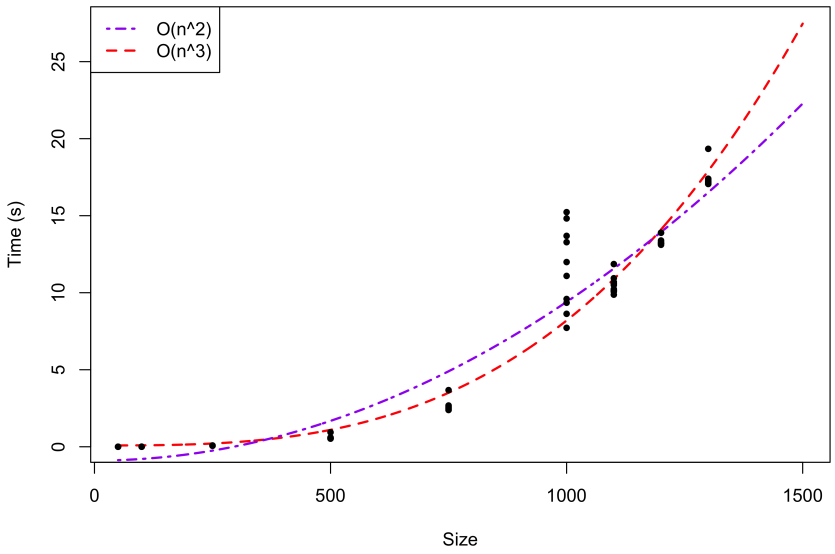

Figure 2: Execution time projection for the direct solver.

Figure 2 shows the execution time projection for the direct solver. Each test has been repeated for 10 times, therefore the vertical points shown on the figure. Two regression lines are calculated and plotted on the figure to show the comparison to different algorithm complexity. According to the regression lines, the complexity of the direct solver lies between $O(n^3)$ and $O(n^2)$.

Space The following output comes from the Valgrind utility provided by the ICS cluster. The leak check functionality is turned on. The direct solver is used to solve a 100-by-100 coefficient matrix. The input file for the matrix is 64 KB, and there is 1 505 KB of memory allocated. The message also shows that there is no possible memory leaks. A conservative projection of the memory needed is that it will require about 20 to 30 times of the size of the input file during the calculation.

```
==18676== Memcheck, a memory error detector
==18676== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==18676== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==18676== Command: ./directSolver.O2 ../data/A_100.csv ../data/b_100.csv 0
==18676==
-----------------------------------
Time profiling for the matrix inverse function:
Forward elimination: 0.02s (66.67%)
Backward elimination: 0.01s (33.33%)
Inverse function total: 0.03s (1)
```

```
-----------------------------------
Total time for the direct method: 0.14s
==18676==
==18676== HEAP SUMMARY:
==18676==     in use at exit: 0 bytes in 0 blocks
==18676==   total heap usage: 11,242 allocs, 11,242 frees, 1,505,606 bytes allocated
==18676==
==18676== All heap blocks were freed -- no leaks are possible
==18676==
==18676== For counts of detected and suppressed errors, rerun with: -v
==18676== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```

2.2 Iterative Solver

2.2.1 Justification

Popular iterative solvers come from 1) the Jacobi method and 2) the Gauss-Seidel method. They uses the similar iteration scheme. The scheme can be expressed as $x_{k+1} = M^{-1} \times (b - N \times x_k)$, where x is the solution vector, b is the right-hand-side vector, and M and N matrices are defined so that $A = M + N$. The difference of Jacobi and Gauss-Seidel methods lies in how they define the decomposition matrices M and N. Jacobi method simply define them using the identity matrix. This is a stable approach that the algorithm convergence is guaranteed. However, the convergence rate is very low in practice. The Gauss-Seidel method defines the decomposition using the upper and lower strict triangular parts of the matrix A. This method has a faster convergence.

Although Gauss-Seidel method is in practice more efficient, both methods are implemented in the iterative solver. However, in this section, profiling and projection are only done for Gauss-Seidel method.

2.2.2 Optimization Flags

With the similar approach to the previous section, same tests have been carried out for programs compiled with different optimization flags. A 500-by-500 coefficient matrix is used to test programs. Each test has been repeated for 10 times for statistical significance.

Different from the case of the direct solver, $-O2$ works best with the iterative solver where it generates the lowest averaged execution time and the smallest spread. Therefore, this optimization flag is chosen.

2.2.3 Profiling

Time The following output is for time profiling of the code. The Gauss-Seidel iterative method is used to solve a 500-by-500 coefficient matrix. The program terminates after 0.1015 second, and it runs for 17 iterations. All-1 initialization is used for this example test. The first section of the standard output shows the forward and the backward elimination time used by the matrix inverse function.

```
-----------------------------------
Time profiling for the matrix inverse function:
Forward elimination: 0.04726s (65.94%)
Backward elimination: 0.02441s (34.06%)
Inverse function total: 0.07167s (1)
-----------------------------------
Iteration 1 residual: 3.294e+07
Iteration 2 residual: 3.897e+06
Iteration 3 residual: 1.096e+06
Iteration 4 residual: 1.873e+05
Iteration 5 residual: 4.608e+04
```

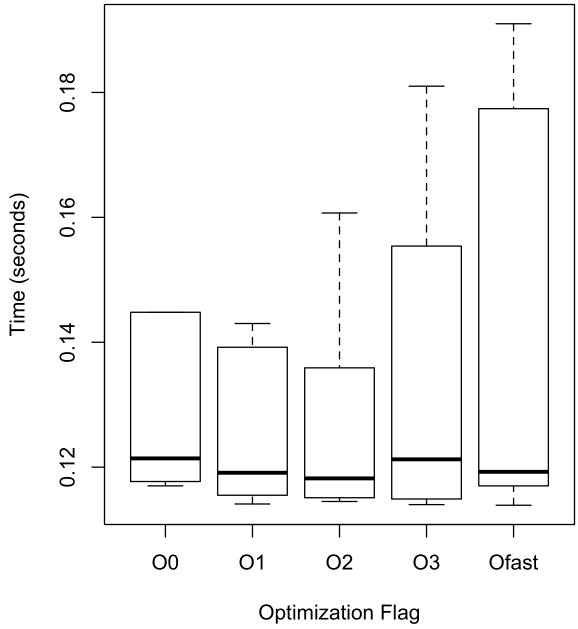

Figure 3: Execution time of the Gauss-Seidel iterative solver with different optimization flags.

```
Iteration 6 residual: 9198
Iteration 7 residual: 1995
Iteration 8 residual: 454.8
Iteration 9 residual: 87.32
Iteration 10 residual: 22.34
Iteration 11 residual: 3.906
Iteration 12 residual: 1.084
Iteration 13 residual: 0.1781
Iteration 14 residual: 0.05145
Iteration 15 residual: 0.008354
Iteration 16 residual: 0.002367
Iteration 17 residual: 0.0004003
Total time for the iterative method: 0.1015s
```

Figure 4 shows the execution time projection for the Gauss-Seidel iterative solver. The figure contains both tests for all-1 and random initialization. The purple points indicate random initialization case and the red plus signs indicate all-1 initialization case. Results suggest that there is not a significant difference in execution time between the two initialization methods. Since two methods perform fairly comparable to each other and the regression lines are very similar to each other, only the regression lines for the random initialization case are shown in the figure. It shows that the algorithm complexity lies between $O(n^3)$ and $O(n^2)$. However, please note the absolute execution time is much smaller than that of the direct method.

Space The following output comes from the Valgrind utility provided by the ICS cluster. The leak check functionality is turned on. The Gauss-Seidel iterative method is used to solve a 100-by-100 coefficient matrix. The input file for the matrix is 64 KB, and there is 1 965 KB of memory allocated. The message also shows that there is no possible memory leaks. A conservative projection of the memory needed is that it will require

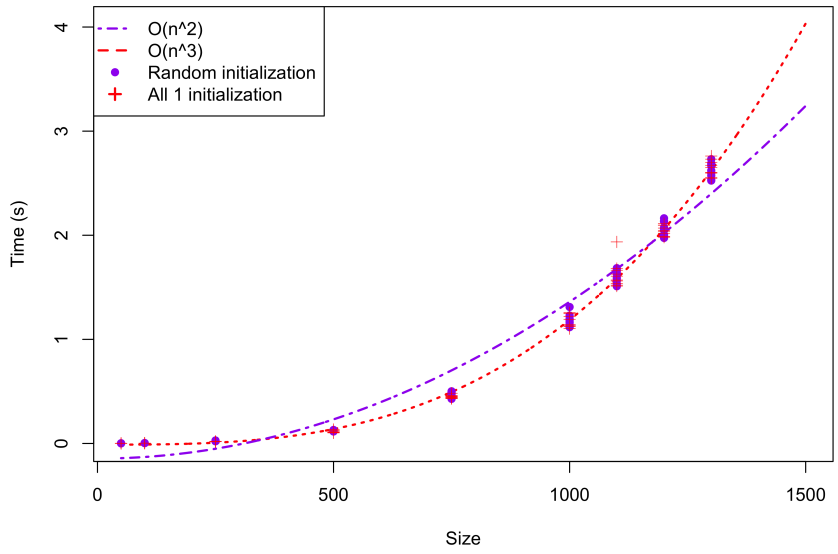

Figure 4: Execution time projection for the Gauss-Seidel iterative method.

about 20 to 30 times of the size of the input file during the calculation.

```
==27773== Memcheck, a memory error detector
==27773== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27773== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27773== Command: ./iterativeSolver.O2 G ../data/A_100.csv ../data/b_100.csv 100 2 0
==27773==
----------------------------------
Time profiling for the matrix inverse function:
Forward elimination: 0.01s (33.33%)
Backward elimination: 0.02s (66.67%)
Inverse function total: 0.03s (1)
----------------------------------
Total time for the iterative method: 0.13s
==27773==
==27773== HEAP SUMMARY:
==27773==     in use at exit: 0 bytes in 0 blocks
==27773==   total heap usage: 27,780 allocs, 27,780 frees, 1,965,632 bytes allocated
==27773==
==27773== All heap blocks were freed -- no leaks are possible
==27773==
==27773== For counts of detected and suppressed errors, rerun with: -v
==27773== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```

3 Solver Comparison

After the above tests and profiling for the directive solver and the Jacobi (results not included in the report) and the Gauss-Seidel iterative solver, a brief summary is provided below:

- The directive solver is much more slower compared with the Gauss-Seidel iterative solver. The difference in execution time can be as large as about 10 times.
- The iterative solver uses a bit more memory than the directive solver in the case of the current implementation. However, this might be further optimized.
- The directive solver works for any type of matrix, include non-diagonally dominant matrices. However, there is an assumption for the iterative solver to work which is that the spectrum radius of the coefficient matrix should be less than 1. This assumption might not be fulfilled in real cases.
- For production scale, the Gauss-Seidel iterative solver should be the most competent out of the three because of the lowest absolute execution time and the stable scale. When the problem size gets bigger, the execution time variability of the program stay rather stable.

4 Discussion and Conclusions

A basic overview of what I have done is provided below:

- Implemented the direct solver using Gaussian Elimination and the iterative solver using Jacobi method and Gauss-Seidel method.
- Created test scripts and data sets using R. Data sets contain tests with sizes ranging from 5-by-5 to 1300-by-1300 matrices as input to different solvers.
- Carried out time and space profiling both on MacBook Air and the ACI ICS clusters using code-based profiling functions and the Valgrind utility.
- Wrote analysis and visualization scripts in R.
- Conducted writing of this report.

The current implementation relies heavily on matrix manipulation. Therefore there is a lot of copying and moving the data, causing the rather high memory requirement. This can be solved by providing APIs to directly manipulate the matrix in place, rather than copying and moving data elsewhere.

Appendices

A Acknowledgements

I would like to express my gratitude to the instructors of CSE 597, Dr. Adam Lavely and Dr. Christopher Blanton for the excellent organization and guidance for the course. I would like to also thank my advisor, Prof. Guido Cervone, for mentoring and guiding me.

B Code

Please find the full code and data sets in the GitHub repository. Please find the source code to the report #1 at Overleaf.

Instructions for compiling and reproducing the results can be found in the *README.md* file. I have compiled the codes using the regular nodes (aci-b), and run the tests on computing nodes (aci-i).

Below is a summary of the folders and files in the repository:

- R/ contains the R scripts for data analyses and visualization.
- data/ contains the test data sets and the R script to generate them.
- .gitignore is the file specifying which files should be ignored in Git.
- CMakeLists.txt guides CMake to generate a make file. Please see README.md for detailed usage.
- LICENSE.txt is the MIT license.
- Matrix.cpp is the source file for Matrix library.
- Matrix.h is the header file for Matrix library.
- README.md contains basic information for the repository and detailed information for how to compile and reproduce the results.
- directSolver.cpp is the source file for the direct solver.
- iterativeSolver.cpp is the source file the iterative solver.
- optimization-flags-compile.sh is the bash script to compile solvers with different optimization flags and to organize them in a particular folder.
- optimization-flags-test-direct.sh is the bash script to test the direct solver with different optimization flags.
- optimization-flags-test-gauss.sh is the bash script to test the Gauss-Seidel iterative solver with different optimization flags.
- profiling-time-direct.sh is the bash script to profile the direct solver.
- profiling-time-guass.sh is the bash script to profile the iterative solver.
- testMatrix.cpp is the source file for testing Matrix library.

C Licensing and Publishing

This project and all the codes are protected by the MIT license. I chose this license for its permissive features and brevity in language.

I plan to publish the related work in a meteorology journal or a computing conference.

References

[1] Luca Delle Monache, F Anthony Eckel, Daran L Rife, Badrinath Nagarajan, and Keith Searight. Probabilistic weather prediction with an analog ensemble. *Monthly Weather Review*, 141(10):3498–3516, 2013.

[2] Tilmann Gneiting and Adrian E Raftery. Weather forecasting with ensemble methods. *Science*, 310(5746):248–249, 2005.

[3] Thomas M Hamill. Interpretation of rank histograms for verifying ensemble forecasts. *Monthly Weather Review*, 129(3):550–560, 2001.

[4] Erich Kaltofen and Gilles Villard. On the complexity of computing determinants. *computational complexity*, 13(3-4):91–130, 2005.

[5] Edward Lorenz. Chaos in meteorological forecast. *J. Atmos. Sci*, 20:130–144, 1963.

[6] Simone Sperati, Stefano Alessandrini, and Luca Delle Monache. Gridded probabilistic weather forecasts with an analog ensemble. *Quarterly Journal of the Royal Meteorological Society*, 143(708):2874–2885, 2017.