MCMC Storage Guide with Simulation Result

This report is prepared for the statistical seismology team to help them identify the most efficient method for storing m numbers of model parameters $\theta^{(i)} \in \mathbb{R}^p$ for the MCMC. In this report, we conduct two simulation experiments with different values of m, while keeping $p = 10$, to assess the computational cost of four methods: data-frame row-wise, data-frame column-wise, matrix row-wise, and matrix column-wise.

The simulation study using $m = 10^4$

To ensure minimal computational time, our approach begins with the testing of lower m values to gauge the performance of the four methodologies. We've designed four distinct functions, each representing one of the methods, that highlight their operations and compute the associated computational costs. In all cases, we assign identical values to $\theta = (\theta_1, \theta_2, \ldots, \theta_p)$ as a means of controlling the variables. These methods are evaluated 10 times, with the first graph of Figure 1 showcasing the average and median times for each method. Furthermore, Table 1 presents the running time statistics (in ms) for each method executed 10 times. From our analysis, the top two methods that effectively minimise the mean computational time are matrix_column_wise and matrix_row_wise.

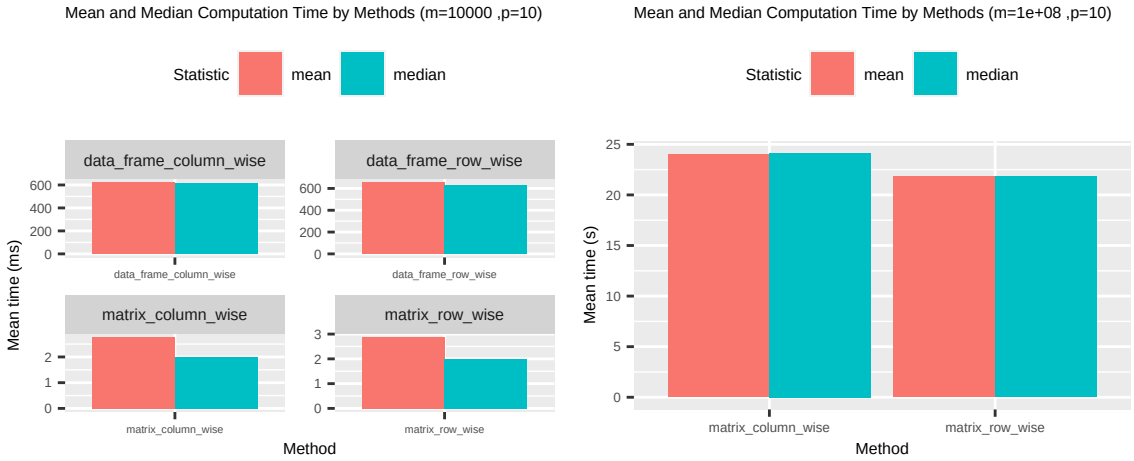

Figure 1: The average and median running time of the mathods selected

Table 1: Statistics of the running time in millisecond for each method

Method	min	lq	mean	median	uq	max	neval
data_frame_row_wise	584.2681	616.3664	652.9419	626.6956	694.6772	831.8609	10
data_frame_column_wise	585.6123	604.7819	620.0235	613.2986	620.3459	689.2730	10
matrix_row_wise	1.7781	1.8322	2.8744	1.9883	3.7858	6.3634	10
matrix_column_wise	1.8561	1.9103	2.7675	2.0019	2.2949	7.0105	10

We conducted an in-depth analysis of the computation time breakdown for each method using the `profvis` package. Our results highlight that the increased computation times for both row-wise and column-wise data-frame methods are chiefly attributable to the extended duration needed to assign values to the data frame. This is in contrast with the comparatively swift assignment to the matrix. Furthermore, the functions used to transpose the data frame/matrix and to convert a matrix into a data frame proved highly efficient, requiring only a minimal portion of the overall processing time.

Further Simulation Study with $m = 10^8$

We proceed with our simulation study by focusing on the top-two-performing methods, specifically matrix_column_wise and matrix_row_wise. In this phase, we simulate the assignment process with $m = 10^8$ and $p = 10$, utilising only these two superior methods. The resulting average and median times for both methods in 3 simulations are visualised in the second graph of Figure 1. Upon inspection, it's evident that the computation time significantly increases compared to when using $m = 10^4$. However, the matrix_row_wise method consistently shows the least computational time.

Recommended Approach

Drawing on the insights garnered from our simulation study, we advocate for the use of the row-wise matrix method to store θ values. This process entails the creation of an $m \times p$ matrix initially filled with NA values. Fill each row in a sequential order and subsequently convert the filled matrix into a data frame.