Sébastien Pacteau January 7, 2013

Writing tests report

The goal of this report is to show that there are different ways to write data in a File using JavaSeis. The currently implemented writing SeisDataContainer function *FileWrite* enables to write datasets from 1D to 4D using *for* loops in Matlab.
One of the inescapable steps of the writing process is to copy data from Matlab to Java memory. This can be done slice by slice or sequentially passing any type of sub-dataset to Java memory. The fact is that, at the end, the whole multidimensional data will have been transiting by Java memory.
The challenge is the next one: on the one hand, we would like to perform the *for* loop on the dimensions as fast as possible, which would require to be done using JavaSeis (Java *for* loop are much faster than Matlab's ones); but, on the other hand, the amount of data copied to Java memory at once, in a step of the sequential copy operation, must not be too large (else, copy time would grow exponentially). The way to write data as fast as possible is therefore not straightforward.
The next writing tests help to have an idea about the advantages and drawbacks of passing large proportions of the total data directly to JavaSeis, thus reducing the number of *for* loops in Matlab. Here, we study the extreme case where the whole dataset is given to JavaSeis at once, that is to say that no *for* loop has to be done in Matlab. This new-implemented writing algorithm can take into account up to 5D data and is written avoiding dimensions permutations, as it was previously the case. That is to say that the dimensions for data representation are simply flipped compared to Matlab's dimensions.

1. Current problem: *for* loop in Matlab

The problem of the current implementation is that for a large number of items, the writing begins to be really slow.

EXAMPLE 1

Let us consider a Matlab multidimensional array containing a number of ***n*** **cubic volumes of 1,000 elements each**. A linear slowing down can be observed with the currently implemented method (cf. Figure 1).

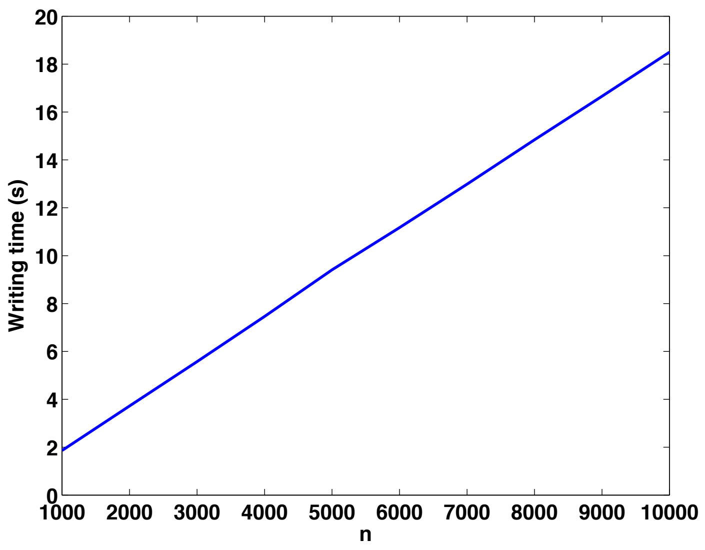


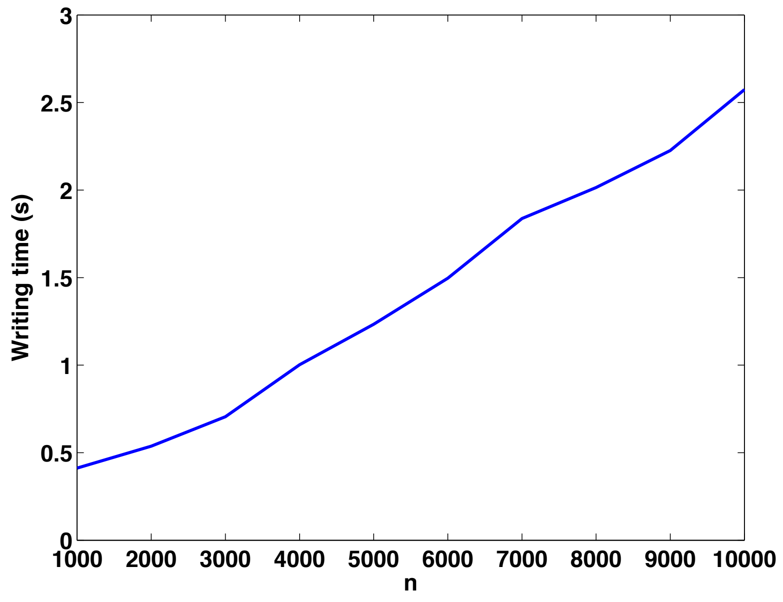
Figure 1: writing time of the currently implemented writing function in function of the number, n, of cubic volumes of size 10x10x10

2. Solution 1: *for* loop in JavaSeis

One of the solution to avoid this long calculation time due to large number of writing iterations is to directly implement the *for* loop in JavaSeis after having flipped all the dimensions of the Matlab array (in a first time, we decide to keep this currently implemented permutation strategy).

2.1. Results

Figure 2 shows the results of EXAMPLE 1 (cf. §1), this time using the *for* loop in JavaSeis. The improvement is obvious. For example, it takes 2.5 seconds to write a multidimensional array of 10,000 cubes of 1,000 elements each with the new method, versus 18 seconds with the initial method!



2.2. Limitations

Despite those apparent good results, the new method will not systematically be faster than the initial one. In some cases, it may even become slower, notably in the case where large volumes are involved. Here is an illustration:

EXAMPLE 2

Let us consider a Matlab dataset of size $10k \times 10k \times 10k \times 1,\ k \in \mathbb{N}$.

The new implemented function begins to be slower than the current one when the number of elements in the volumes is higher than about 3,375,000 (k=15) as shown on Figure 3.

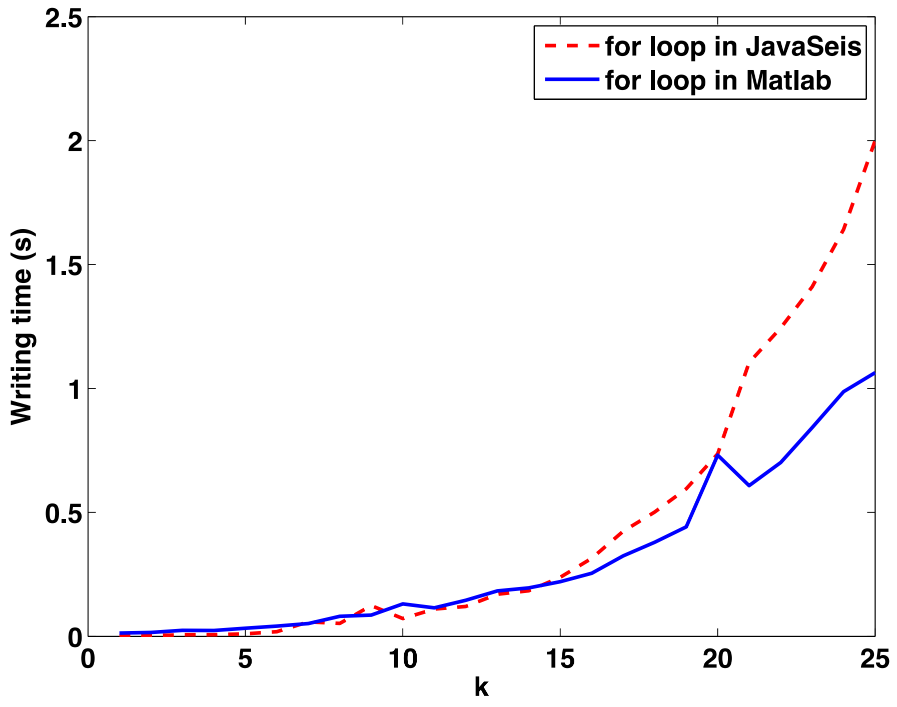


Figure 2: writing time of the current function, using a for loop in Matlab and of the suggested function, using a for loop in JavaSeis. The abscissae are the values of k (cf. EXAMPLE 2 statement).

Copy time

To perform the *for* loop in JavaSeis, one has to copy the Matlab multidimensional array into Java memory space. Analyzing the different times at stake, one can notice that **copy time** is the main limitation of the new implemented function. Copy time is growing more than linearly relatively to the number of elements in the volume.

Permutation time

As for the previous method, the new writing method also has a part of its process allocated to the permutation operation. The time needed to do a multidimensional permutation is necessary a bit shorter than the total time needed to do successive transpositions on the frame in the first method.

3. Solution 2: use of flipped dimensions

As there is no doubt that the *for* loop in JavaSeis is faster than the one in Matlab and could lead to a great improvement in writing time, let us try to improve the new writing function. A good improvement can be done avoiding doing the permutation in Matlab.

For that, one may first think about doing the permutation directly in JavSeis. This is not a good idea since, in that case, the Matlab multidimensional array would first have to be copied into a JavaSeis multiarray before being permuted using JavaSeis *Transpose* class methods. That would engender an additional **copy time**.

A better solution to avoid permutation time would be to choose to flip dimensions in JavaSeis so as not to have to do any permutation at all. Thus, a Matlab multidimensional array of dimensions (x,y,z) will for example be stored as a JavaSeis object of dimensions (z,y,x).

Really good results, shown on Figure 3 and Figure 4, are obtained for the EXAMPLE 1 (cf. §1) with this new method:

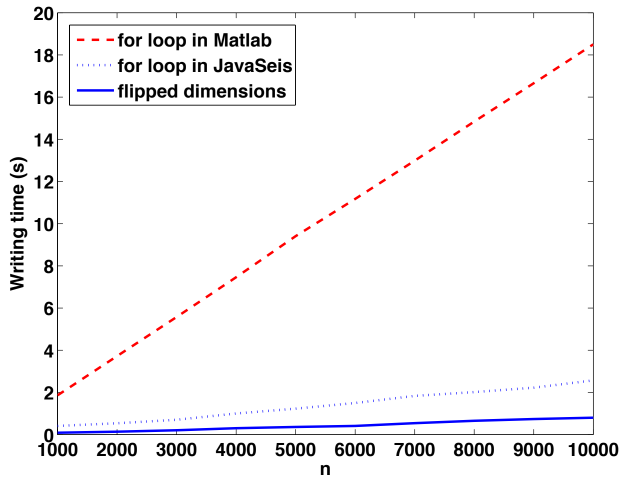


Figure 3: comparative curve to show the improvement obtained using the for loop in JavaSeis with flipped dimensions convention. The abscissae, n, are the number of volumes.

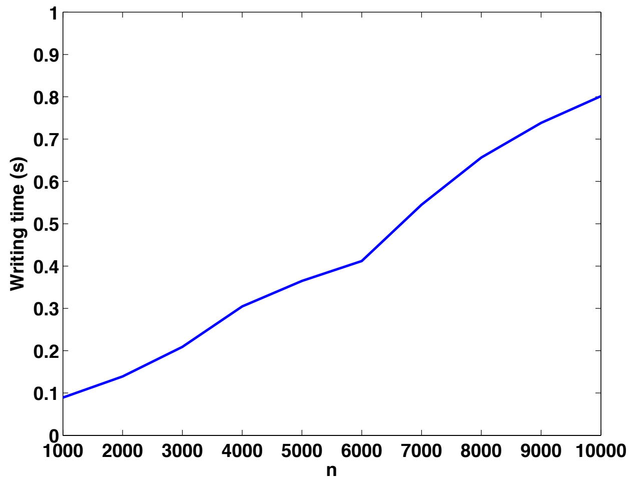


Figure 4: detail of the writing time of the improved "flipped dimensions" method in function of the number of volumes, n.

The suggested method is definitely faster than the initial one in this case. Moreover, the interesting aspect is that the total writing time increases slower than for the first method. Here, when 1000 additional volumes of 1000 items are added, the writing time is augmented of about 0.07 seconds versus 1.65 seconds with the initial method.

Applying this new implementation with flipped dimensions to EXAMPLE 2, one may notice that a little improvement has been done concerning the point (size of the data) from which the writing time begins to strongly increase (cf. Figure 5).

However, this memory issue will have to be corrected, wisely defining which *for* loops have to be done in Matlab or in JavaSeis and what is the maximum size of the data to provide to JavaSeis a once. An optimization strategy could also be defined seeing in which order should the different dimensions be processed. It could sometimes be more efficient doing a *for* loop in Matlab on a given dimension first.
Eventually, an efficient algorithm should be implemented in such a way that the total writing time is always the fastest, considering the best way to minimize the sum of processing time (the *for* loop) and copy time.

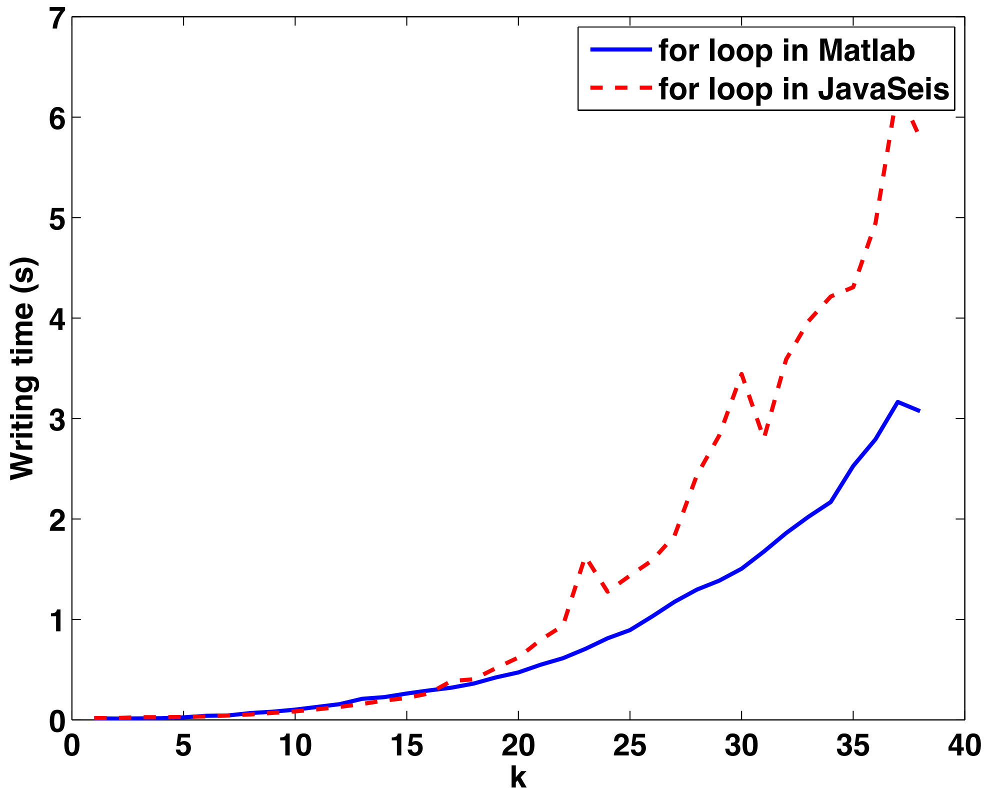


Figure 5: writing time of the current function, using a for loop in Matlab and of the suggested function, using a for loop in JavaSeis and no permutation. The abscissae are the values of k (cf. EXAMPLE 2 statement).