Efficient Modification of the Upper Triangular Square Root Matrix on Variable Reordering

Khen Elimelech and Vadim Indelman

ICRA 2021

ANPL | Autonomous Navigation and Perception Lab

TECHNION Israel Institute of Technology

State Estimation → Linear Systems

- State estimation problems, such as Simultaneous Localization And Mapping (SLAM) and Bundle Adjustment (BA), are some of the main focus points of modern robotics research.
- In such optimization problems, we wish to estimate a state vector $X \in \mathbb{R}^n$, which typically consists of robot poses, and/or positions of landmarks, given a set of (m) stochastic constraints.

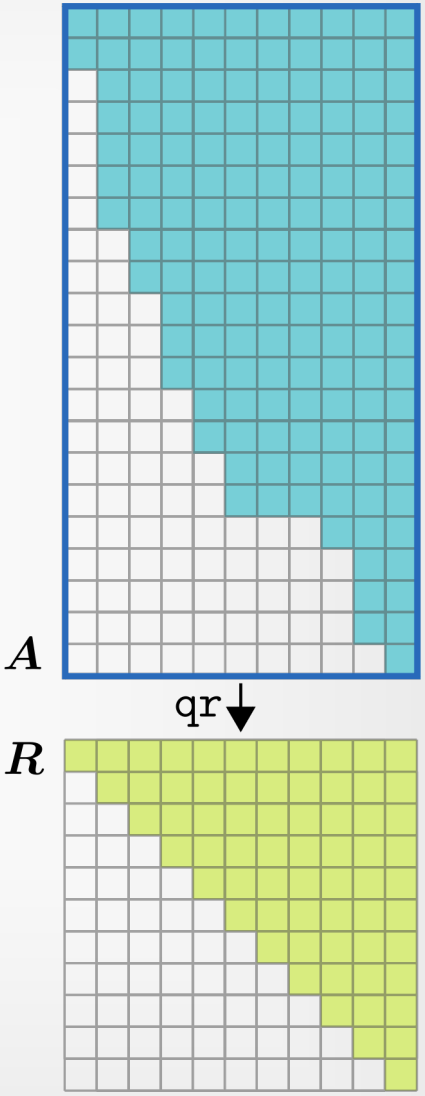
- When these constraints are linear (or linearized), these problems can be represented as systems of linear equations.
- Such systems can be written in matrix form as

$$A \cdot X = b,$$

where $A \in \mathbb{R}^{m \times n}$ is the coefficient matrix, and $b \in \mathbb{R}^m$ is the RHS.

Solving Linear Systems

- Such systems are often solved by finding the upper triangular "square root information matrix", marked $R \in \mathbb{R}^{n \times n}$, and then performing "back-substitution".
- This "square root matrix" can be found by calculating the Cholesky factorization of the information matrix $\Lambda \doteq A^T A$, such that $\Lambda \doteq R^T R$.
- Equivalently, by calculating the QR factorization of A, such that $A \doteq QR$, for an orthogonal matrix Q.



- In light blue – entries of the coefficient matrix;
- In green – entries of the original square root matrix;
- Colored cells represent possible non-zero entries in the matrices.

Variable Reordering: Motivation

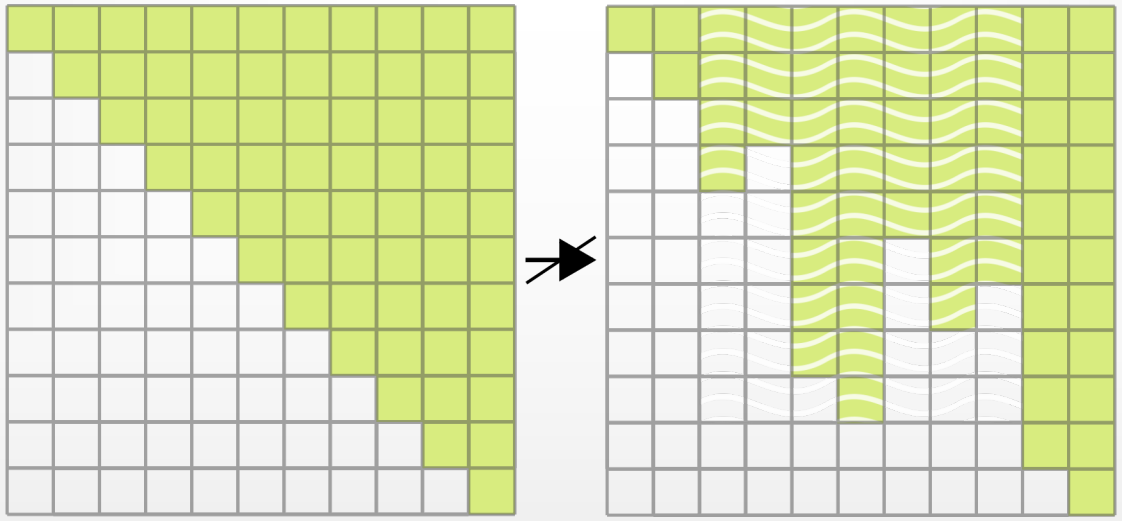
- In sequential estimation, as time progresses, and more and more constraints are added to the system, we are required to update its solution (i.e., state estimate).
- It is known that the order of state variables can drastically affect the cost of such updates, especially in sparse systems.
- It is hence common to periodically optimize the order of variables, e.g., in order to apply a fill-reducing variable order.
- Furthermore, in our recent work, titled “PIVOT: Predictive Incremental Variable Ordering Tactic”, we demonstrated the effectiveness of predictive optimization of the variable order, before planning sessions, in reducing their computational cost.

[1] Efficient Belief Space Planning in High-Dimensional State Spaces using PIVOT: Predictive Incremental Variable Ordering Tactic, K. Elimelech and V. Indelman, IJRR ’21 (invited)

Variable Reordering: Challenges

- Surely, the order of variables in X must match the order of columns in R (and A).
- Yet, a naïve attempt to permute the columns of R, to convey reordering of the variables, would break its triangular shape, and is, therefore, an inappropriate solution.

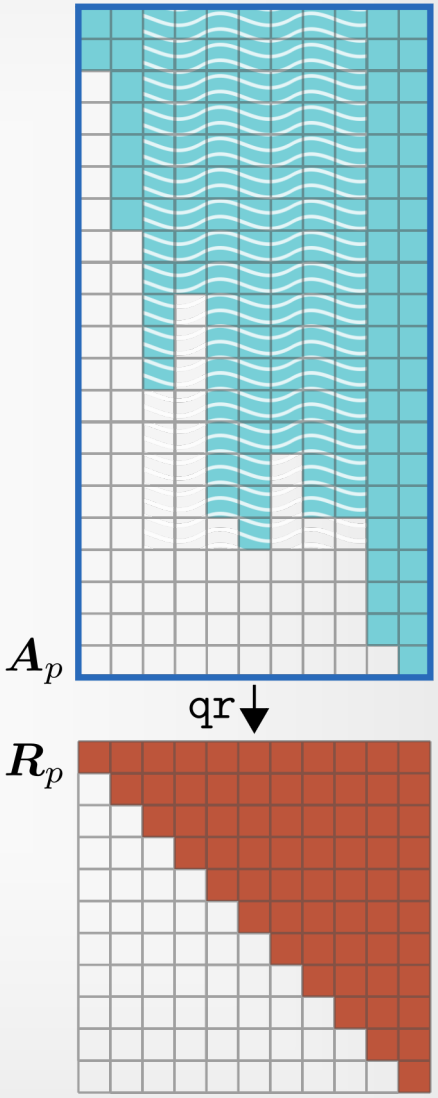
- White overlay – reordered columns.



Variable Reordering: Challenges

- Thus, variable reordering typically implies expensive re-factorization of the system, under the new order, in order to calculate the modified square root matrix.

- Unfortunately, re-factorization can be challenging or even infeasible when handling large-scale systems.



- Dark blue borders – matrices due to re-factorization.
- In red – entries calculated via re-factorization of A.

Contribution

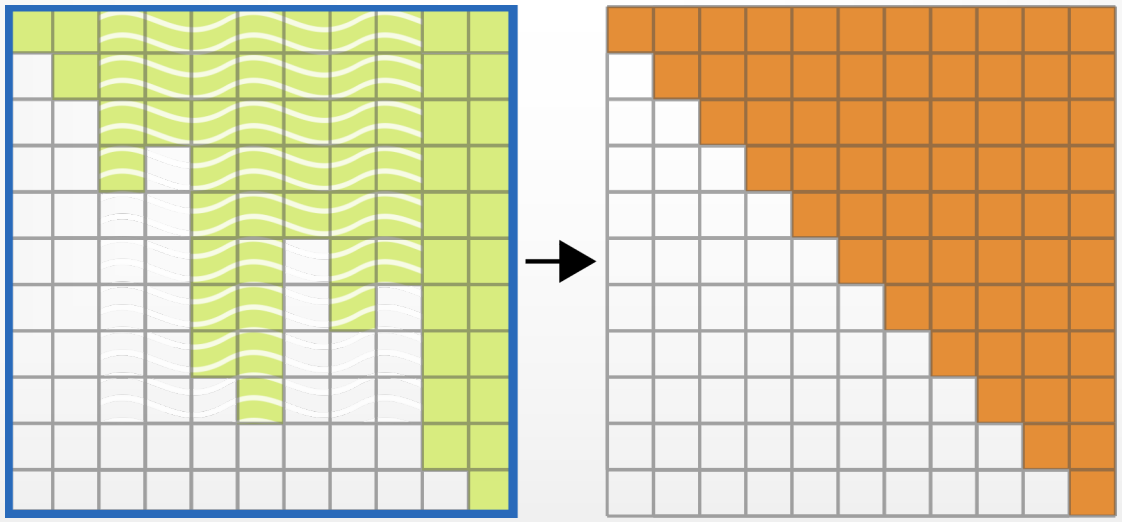
- To tackle this concern, in this work, we propose a novel algorithm for efficient modification of this square root matrix, on variable reordering.

- As we shall explain, such modification can be performed efficiently without re-accessing A at all, or with minimal re-factorization of it.

- Note that we assume the new variable order is given (as a permutation), and only discuss how to practically apply it.

Direct Modification

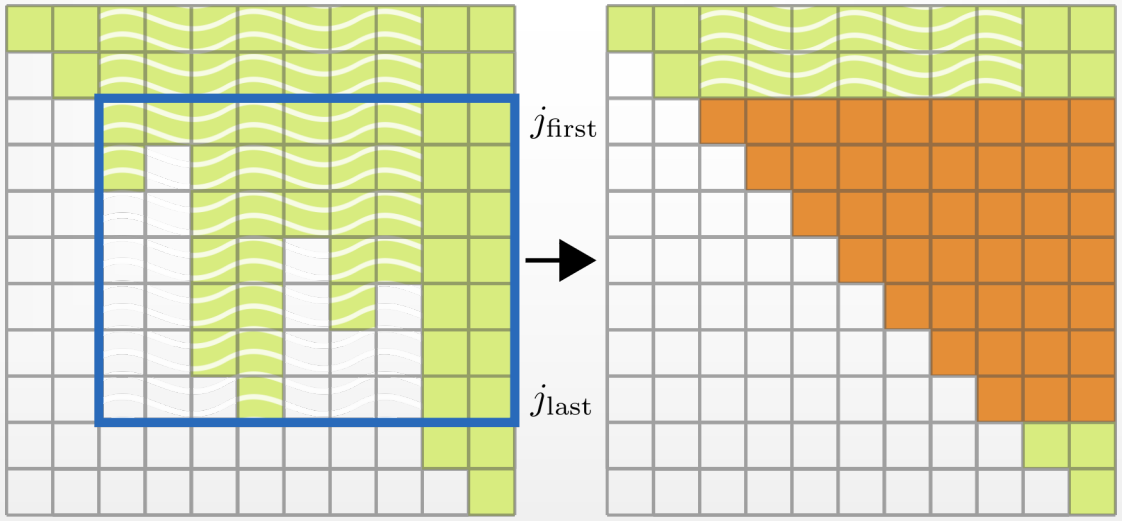
- First, we shall recognize that the modified square root matrix can, in fact, be inferred without re-accessing A.
- Instead, we can simply re-apply the QR factorization on the square root matrix directly, after permuting its columns, in order to “correct” its shape; we hence refer to such approach as “direct modification”.
- Nonetheless, this naïve re-application of QR is still not optimal for the task of variable reordering.



- In orange – entries calculated via re-factorization of R.

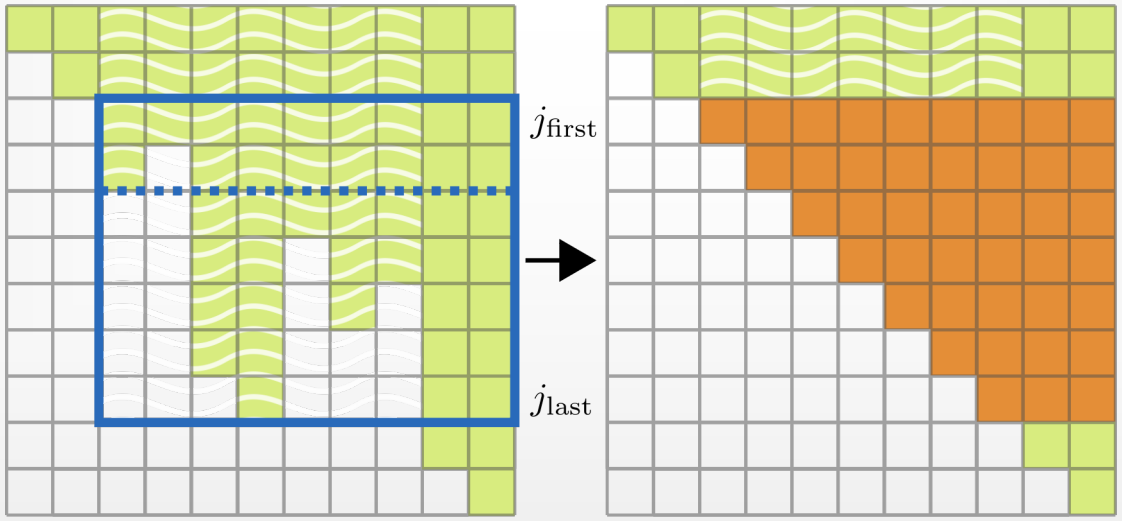
Local Effect of Variable Reordering

- Let us mark with j_{first} the index of the first permuted variable, and with j_{last} the index of the last permuted variable.
- As the first step in optimizing the modification process, we identified that variable reordering has only a "local effect" on R.
- In other words, to apply the new variable order, we should only re-calculate the block of affected rows indexed between j_{first} and j_{last}; all other rows remain unchanged.



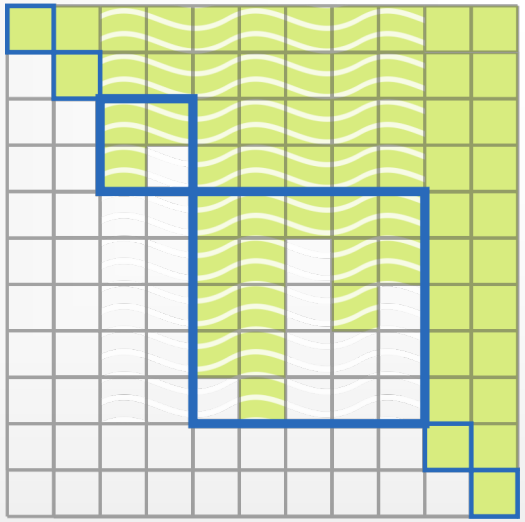
Identifying Independent Row Blocks

- Further, by decomposing the variable permutation into disjoint cycles, we can divide it into distinct “sub-permutations”, which affect non-overlapping subsets of variables.
- Hence, we can divide (as marked with the dashed blue line) the affected row block into distinct sub-blocks, which can be re-calculated independently, and even in parallel!



Efficient Row Modification

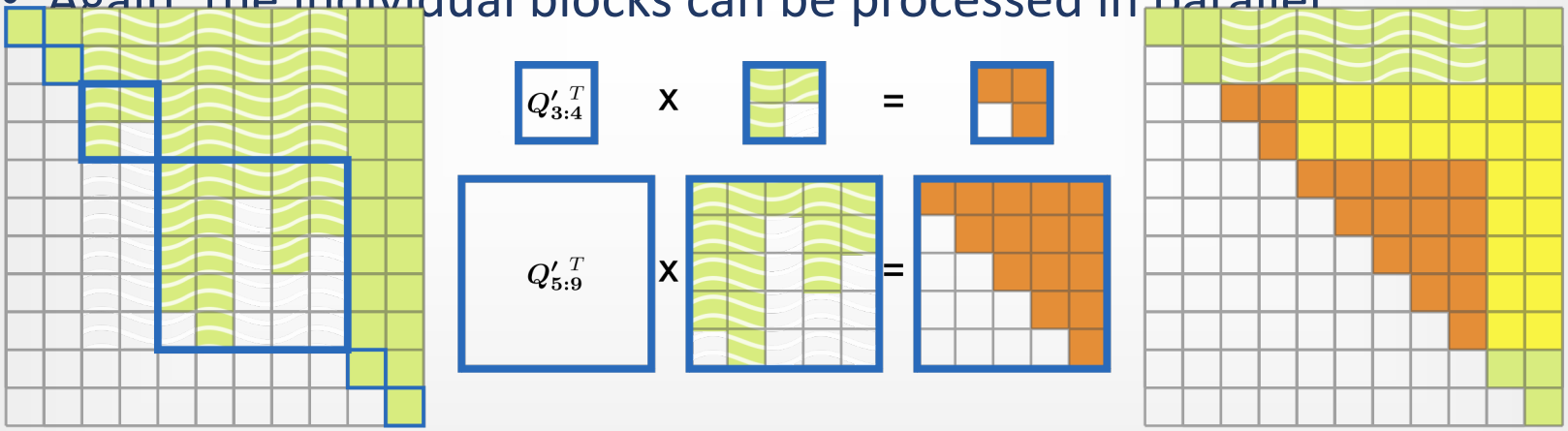
- We may note that each row block in the square root matrix is “wide”, i.e., it contains more columns than rows.
- Accordingly, we show that, for each such row block, it is sufficient to examine its square sub-block, around the matrix diagonal, in order to infer its “correcting” orthogonal transformation (via QR factorization).



Optimized Direct Modification

- After correcting the block around the diagonal, the correcting transformations should and can be easily applied to the remainder of the elements in each row block.

- The following figure represents our suggested optimized algorithm for direct modification of R, when considering variable reordering.
- Again, the individual blocks can be processed in parallel.



- In yellow – entries calculated by applying a pre-calculated transformation.

Numerical Concerns

- We note that direct modification is more prone to numerical errors, in comparison to naïve re-factorization, since it conveys an overall longer sequence of mathematical operations:

$$A \rightarrow R \rightarrow R_p \text{ (direct modification)}$$

vs.

$$A_p \rightarrow R_p \quad \text{(re-factorization)}$$

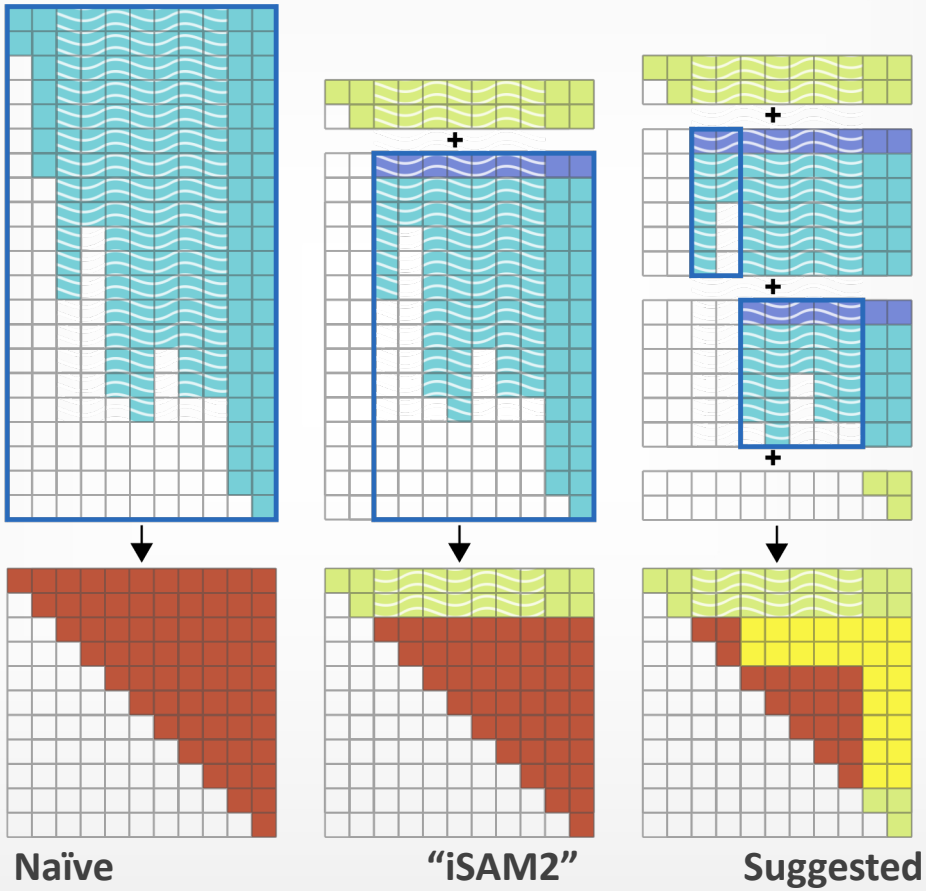
- Such numerical errors may lead to “false fill-in” in the matrix, when zero entries incorrectly become non-zeros.

Modification via Re-factorization

- Therefore, if maintaining sparsity is crucial, we should modify the affected row block via re-factorization of A, instead.
- This, of course, is more expensive than direct modification.
- Yet, by utilizing the same conclusions we identified before (locality of variable reordering, division into distinct row blocks, and efficient row modification), we can similarly minimize the sub-matrix of A, which is due for re-factorization.

Modification via Re-factorization

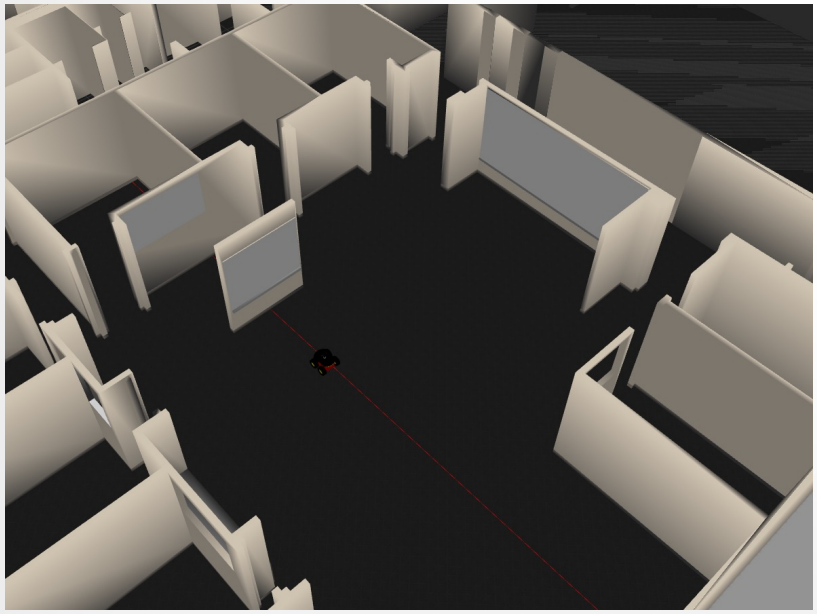
- The figure on the right represents our suggested optimized algorithm for modification of R via re-factorization.



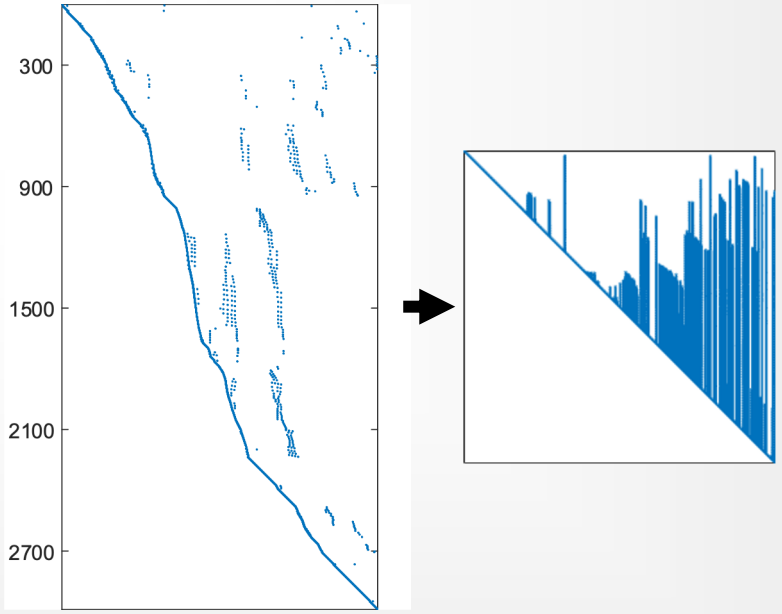
- In purple – marginal factors obtained from the original factorization process.

Experimental Demonstration

- To test our modification algorithm(s), we applied them to a realistic linear system, derived when solving a robotic SLAM problem.
- Constraints between the poses in this state represent the robot motion and the inferred loop closures (via point cloud matching).



A visualization of the robot navigating in the unknown indoor environment.



The coefficient matrix A, and its square root R, at the end of navigation.

Experimental Demonstration

- We considered a randomized variable permutation, and appropriately modified R, using different modification algorithms.
- We measured the runtimes of these calculations, in order to compare the efficiency of the algorithms.
- We also measured the number of non-zero entries in the modified matrices, in order to compare the accuracy of the algorithms.
- The results are summarized in the table below:

Table I: Comparison of modification algorithms.

Category	*Algorithm*	*Runtime (ms)*	*NNZ*
Direct	Naive	87	390k
	Optimized	13	392k
	Optimized (parallel)	11	392k
Re-factor	Naive	336	304k
	Incremental (iSAM2)	229	304k
	Optimized	83	307k
	Optimized (parallel)	70	307k

Summary

- In this work, we provided a new algorithm for modification of the square root matrix of a linear system, on variable reordering.
- We identified three main conclusions in that regard:
  - (1) variable reordering has a “local effect” on the matrix;
  - (2) the modification can be parallelized, by identifying independent row blocks;
  - (3) each row block can be efficiently modified, by only examining a relevant square sub-block.
- We applied these conclusions in two algorithmic variations: via direct modification of R, and via partial re-factorization of A.
- Generally, direct modification enjoys a relatively lower computation time, while re-factorization is less prone to numerical errors.
- In either case, we saw that our optimized algorithm was able to significantly reduce the modification time compared to naïve solutions.

Thank you!

Full implementation of the algorithms is available at:

www.khen.io