List 5 report

Albert Kołodziejski

January 12, 2024

Without Partial Selection

Data Structure

We can safely assume that only cells that are colored can hold meaningful numbers when performing the algorithms.

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$c_{1,4}$	0	0	0	0	0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	0	$c_{2,5}$	0	0	0	0
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	0	0	$c_{3,6}$	0	0	0
0	0	$b_{4,3}$	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	$c_{4,7}$	0	0
0	0	$b_{5,3}$	$a_{5,4}$	$a_{5,5}$	$a_{5,6}$	0	$c_{5,8}$	0
0	0	$b_{6,3}$	$a_{6,4}$	$a_{6,5}$	$a_{6,6}$	0	0	$c_{6,9}$
0	0	0	0	0	$b_{7,6}$	$a_{7,7}$	$a_{7,8}$	$a_{7,9}$
0	0	0	0	0	$b_{8,6}$	$a_{8,7}$	$a_{8,8}$	$a_{8,9}$
0	0	0	0	0	$b_{9,6}$	$a_{9,7}$	$a_{9,8}$	$a_{9,9}$

It is also noteworthy that the majority of work done by the algorithms is done for one row, then another and so on. That's why I propose storing every color as a different Vector so that there will be fewer cache misses. The consideration here is to correctly interpret coordinates, as they are not the same as in the matrix.

$\mathbf{Ax = b}$

When developing the algorithm I have considered the following optimizations:

- There will be no operations for the last row
- When performing operations on rows, Let's say multiplying elements from the first row, and subtracting them from elements of the third row, we can stop when we reach the last meaning full element in the first row, so we will not do any operations on zeros
- When looking for elements that we want to turn into zeros, we can only look up to the last row in our block, or block below if we are in the last row

Speed: O(n)
Memory: O(n)

$\mathbf{A = LU}$

When developing the algorithm I have considered the following optimizations:

- There will be no operations for the last row
- When performing operations on rows, Let's say multiplying elements from the first row, and subtracting them from elements of the third row, we can stop when we reach the last meaning full element in the first row, so we will not do any operations on zeros
- When looking for elements that we want to turn into zeros, we can only look up to the last row in our block, or block below if we are in the last row
- I can store L and U in the same matrix

Speed: O(n)
Memory: O(n)

$\mathbf{LUx = b}$

When developing the algorithm I have considered the following optimizations:

- Both L and U are triangular matrices, so computing y and x is very easy

Speed: O(n)
Memory: O(n)

With Partial Selection

Data Structure

Now we need to be careful because we can change rows, so the first row could end up in the last row. That's why I propose to store every row placed for 2 blocks plus one additional element. There should be ever more meaningful elements, and when we want to change a row on $2L + 2$ place, we can interpret the first element in the row as that, going around indexes if needed. Sadly that means we need a second structure to store L as it will be no longer safe to store it in the same matrix as U.

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$c_{1,4}$	0	0	0	0	0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	0	$c_{2,5}$	0	0	0	0
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	0	0	$c_{3,6}$	0	0	0
0	0	$b_{4,3}$	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	$c_{4,7}$	0	0
0	0	$b_{5,3}$	$a_{5,4}$	$a_{5,5}$	$a_{5,6}$	0	$c_{5,8}$	0
0	0	$b_{6,3}$	$a_{6,4}$	$a_{6,5}$	$a_{6,6}$	0	0	$c_{6,9}$
0	0	0	0	0	$b_{7,6}$	$a_{7,7}$	$a_{7,8}$	$a_{7,9}$
0	0	0	0	0	$b_{8,6}$	$a_{8,7}$	$a_{8,8}$	$a_{8,9}$
0	0	0	0	0	$b_{9,6}$	$a_{9,7}$	$a_{9,8}$	$a_{9,9}$

Swapping rows of A could take a long time so to avoid it we can use the vector of swaps, which simply stores indexes of rows, after swaps. So swaps are done on that vector, and when to get or set row y, we first change it to a local row after swaps.

$\mathbf{Ax = b}$

When developing the algorithm I have considered the following optimizations:

- There will be no operations for the last row
- When performing operations on rows, Let's say multiplying elements from the first row, and subtracting them from elements of the third row, we can stop when we reach the last meaning full element in the first row, so we will not do any operations on zeros
- When looking for elements that we want to turn into zeros, we can only look up to the last row in our block, or block below if we are in the last row
- When looking for a bigger element than the pivot, we can only look up to the last row in our block, or block below if we are in the last row

Speed: O(n)
Memory: O(n)

$\mathbf{A = LU}$

When developing the algorithm I have considered the following optimizations:

- There will be no operations for the last row
- When performing operations on rows, Let's say multiplying elements from the first row, and subtracting them from elements of the third row, we can stop when we reach the last meaning full element in the first row, so we will not do any operations on zeros
- When looking for elements that we want to turn into zeros, we can only look up to the last row in our block, or block below if we are in the last row
- When looking for a bigger element than the pivot, we can only look up to the last row in our block, or block below if we are in the last row
- When looking for a bigger element than the pivot, we can only look up to the last row in our block, or block below if we are in the last row

Speed: O(n)
Memory: O(n)

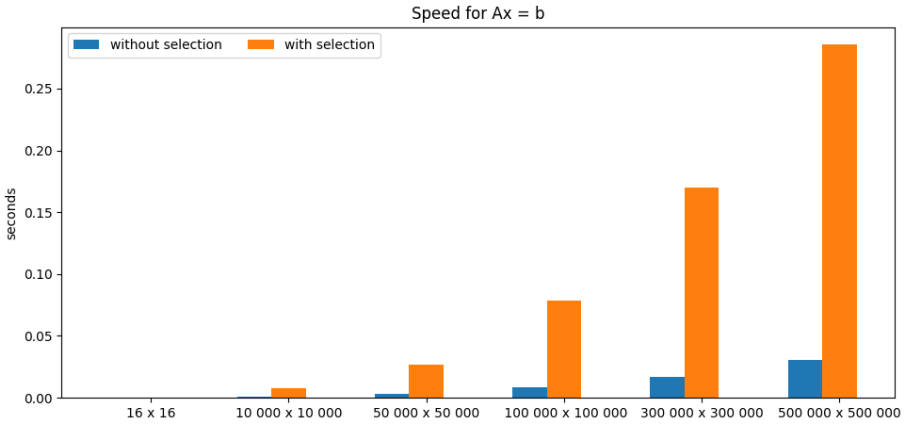
$\mathbf{LUx = b}$

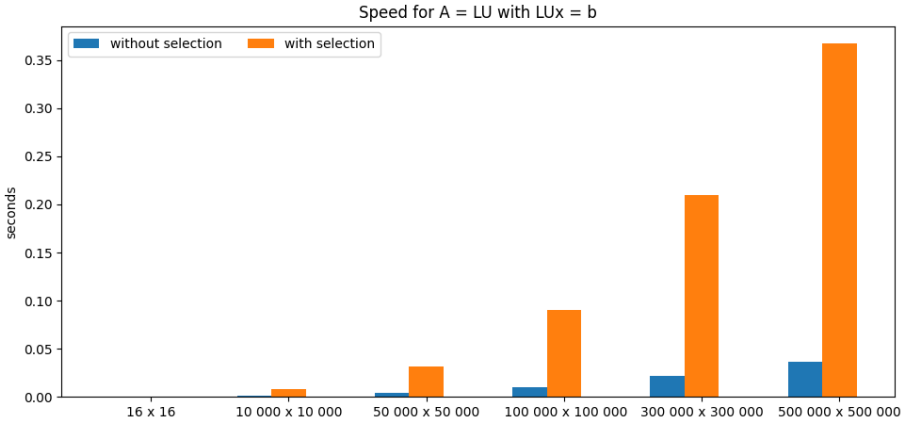
When developing the algorithm I have considered the following optimizations:

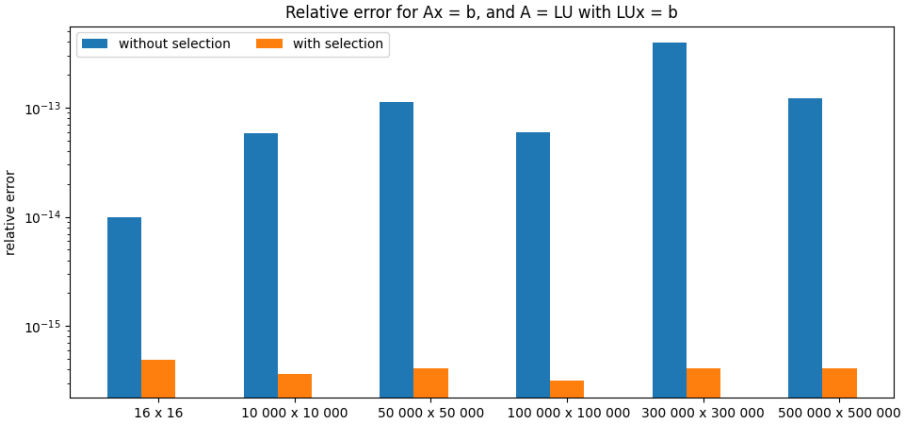
- Both L and U are triangular matrices, so computing y and x is very easy

Speed: O(n)
Memory: O(n)

Results:


Speed for Ax = b
without selection
with selection
seconds
0.00
0.05
0.10
0.15
0.20
0.25
16 x 16
10 000 x 10 000
50 000 x 50 000
100 000 x 100 000
300 000 x 300 000
500 000 x 500 000


Speed for A = LU with LUx = b
without selection
with selection
seconds
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
16 x 16
10 000 x 10 000
50 000 x 50 000
100 000 x 100 000
300 000 x 300 000
500 000 x 500 000


Relative error for Ax = b, and A = LU with LUx = b
without selection
with selection
relative error
10^{-13}
10^{-14}
10^{-15}
16 x 16
10 000 x 10 000
50 000 x 50 000
100 000 x 100 000
300 000 x 300 000
500 000 x 500 000