$LDL^\top$ Rank-m Update

For ease of translation into code, we use zero indexed notation for all the mathematical constructs in this document.
For a matrix M (resp. vector v), we denote by m_i its i-th column (resp. v_i its i-th element).

Let H be a symmetric $n \times n$ matrix and (L, d) its $LDL^\top$ decomposition.
Let Z be a $n \times m$ matrix and λ a vector of size m.
Our objective is to compute a decomposition for the rank-m updated matrix.

$$H' = H + Z \operatorname{diag}(\lambda) Z^\top, \tag{1}$$

$$= H + \sum_{i=0}^{m-1} \lambda_i z_i z_i^\top. \tag{2}$$

This is therefore equivalent to m successive rank-one $LDL^\top$ updates.

1 Rank-one update:

Let z be a vector of size n and λ be a scalar.
We have

$$H = L \operatorname{diag}(d) L^\top, \tag{3}$$

$$= \sum_{i=0}^{n-1} d_i l_i l_i^\top, \tag{4}$$

$$H + \lambda z z^\top = \lambda z z^\top + \sum_{i=0}^{n-1} d_i l_i l_i^\top. \tag{5}$$

Since L is lower triangular with unit diagonal, l_i has exactly i leading zeros followed by 1.
Assume z has at least i leading zeros. We show that $d_i l_i l_i^\top + \lambda z z^\top$ can be rewritten as $a x x^\top + b y y^\top$, where x has i leading zeros followed by 1, and y has at least $i + 1$ leading zeros.

x and y are linear combinations of l_i and z. One possible solution is:

$$a = d_i + \lambda z_i^2, \tag{6}$$
$$\mu = \lambda z_i / a, \tag{7}$$
$$b = \lambda - a\mu^2, \tag{8}$$
$$y = z - z_i l_i, \tag{9}$$
$$x = l_i + \mu y. \tag{10}$$

This process allows us to iteratively increase the number of leading zero elements of z by reducing it with l_i successively, until we end up with z being zero. The values of a and x represent the updated components d_i' and l_i' of the new $LDL^\top$ decomposition.

2 Rank-m update

We can compute a rank-m update using m successive rank-one updates, giving us the sequence $(L^{(k)}, d^{(k)})_{k \in \{0,\dots,m\}}$.
This corresponds to computing $l_{i,j}^{(k)}$ in the lexicographical order over (k, j, i).
One benefit of doing this is that it allows for vectorization in the innermost loop (i).
Numerically, the updates of L, d, Z and λ for each k are done in place. The entire matrix L is repeatedly loaded and written to, which makes the algorithm memory-bound.
So it is preferable to choose a computation order that computes $l_{i,j}^{(0)}, \dots, l_{i,j}^{(m)}$ in quick succession to avoid unnecessary loads and stores to the same memory location.
Computing in the lexicographical order over (j, i, k) accomplishes that, but sacrifices vectorization. So for optimal performance, these must be interleaved, i.e., the nested loop structure would look like:

- iterate j over the columns of L,
- iterate i_0 over the row blocks of L, skipping N rows,
- iterate k over the columns of Z,
- iterate i_1 over the row block elements,

where N is a blocking parameter which must be a multiple of the machine register size. Details about handling the remainder elements are omitted in this analysis, but shouldn't add much complexity.
This loads and stores L and d only once, but requires more memory to store all of Z as all the columns must be available simultaneously.

The update of a column of L and each column of Z requires holding 2 constants per column of Z, (z_i and μ from 6), the current value of l, and the current value of z when k varies.

Assuming we process m_0 columns at a time, this requires $2m_0 + 2\frac{N}{N_{\text{reg}}}$ available registers.
This means that for 16 available registers, we can process at most 7 columns of Z at a time without spilling values on the stack.

The storage layout of Z must be so that the memory access is sequential for the hardware prefetcher to easily predict the data access patterns. This means that it is split into submatrices with number of columns at most m_0. And for each group, it is stored so that N elements from the first column come first, followed by N elements from the second column, ..., followed by N elements from the m_0-th column. Then the next N elements from the first column are placed after that, and the pattern is repeated until all the elements have been stored.