

SOLUTION of (2-2.b):

Let us decompose $\mathbf{x} := [\mathbf{x}_1; \mathbf{x}_2], \mathbf{x}_i \in \mathbb{R}^n$. Performing a block matrix vector multiplication, we obtain:

$$\mathbf{Ax} = \begin{bmatrix} \mathbf{d}_1 \odot \mathbf{x}_1 + \mathbf{c} \odot \mathbf{x}_2 \\ \mathbf{c} \odot \mathbf{x}_1 + \mathbf{d}_2 \odot \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix} \odot \mathbf{x} + \begin{bmatrix} \mathbf{c} \\ \mathbf{c} \end{bmatrix} \odot \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{x}_1 \end{bmatrix}.$$

Here, “ $\odot$ ” denotes the component wise product of two vectors of equal length, and $\mathbf{d}_1, \mathbf{d}_2, \mathbf{c}$ are vectors containing the diagonal entries of the matrices $\mathbf{D}_1, \mathbf{D}_2$ and $\mathbf{C}$.

This can efficiently be implemented in C++.

C++ code 2.2.2: Implementation of `multA()`

```
2 Eigen::VectorXd multA(const Eigen::VectorXd& d1, const Eigen::VectorXd& d2,
3                      const Eigen::VectorXd& c, const Eigen::VectorXd& x) {
4     const unsigned int n = d1.size();
5     assert(n == d2.size() && n == c.size() && 2 * n == x.size() &&
6            "Size mismatch!");
7     Eigen::ArrayXd y = Eigen::VectorXd::Zero(2 * n);
8
9     // TODO: (2-2.b) Efficiently calculate A*x. Eigen's array arithmetic
10    // is helpful here.
11    // START
12    auto& x1 = x.head(n).array();
13    auto& x2 = x.tail(n).array();
14
15    y << d1.array() * x1 + c.array() * x2, d2.array() * x2 + c.array() * x1;
16    // END
17    return y;
18 }
```

Get it on  GitLab (blocklsepiv.hpp).