NUMERICAL METHODS AND PROGRAMMING **2024/2025**

Storage and handling of matrices **(PROBLEMS 3)**

1.— We want to calculate the matrix product $\boldsymbol{K} = \boldsymbol{L}\boldsymbol{U}$ where $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ an upper triangular matrix, both of size n:

a) What is the shape of matrix $\boldsymbol{K}$?

b) Design the minimum storage schemes for the three matrices.

c) Write a multiplication algorithm adapted to the above storage schemes.

d) Describe how does the computational cost grow (measured both in terms of the amount of memory and in terms of the computational time required) as a function of the size of the matrices. Compare it with that which would result from storing the complete matrices and using a multiplication algorithm for full matrices.

Sol. 1.

$$\boldsymbol{K} = \boldsymbol{L}\,\boldsymbol{U}, \qquad \boldsymbol{L} = \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix}, \qquad \boldsymbol{U} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{bmatrix}$$

a) The product of matrices can be expressed as:

$$\boldsymbol{K} = [k_{ij}], \qquad k_{ij} = \sum_{m=1}^{n} l_{im} u_{mj}$$

but it has to be considered that:

$$\begin{aligned} l_{im} &= 0 \quad \text{if } \ m > i \\ u_{mj} &= 0 \quad \text{if } \ m > j \end{aligned}$$

Thus, the product can be obtained as:

$$k_{ij} = \sum_{\substack{m=1,n \\ m \le i,\ m \le j}} l_{im}\, u_{mj} \; = \sum_{m=1}^{\min\{i,j\}} l_{im}\, u_{mj}$$

It can be generally observed that $k_{ij} \neq 0$, since there is always a product that adds to element k_{ij}

Then $\boldsymbol{K}$ is a full matrix.

b) We store:

$$\begin{array}{l} \boldsymbol{L} \longrightarrow \text{Row major lower triangular} \\ \boldsymbol{U} \longrightarrow \text{Column major upper triangular} \\ \boldsymbol{K} \longrightarrow \text{Column major full} \end{array}$$

So that:

$$l_{ij} \rightsquigarrow vl(lpl) \quad \text{with} \quad lpl = \frac{i(i-1)}{2} + j; \qquad j \leq i$$

$$u_{ij} \rightsquigarrow vu(lpu) \quad \text{with} \quad lpu = \frac{j(j-1)}{2} + i; \qquad i \leq j$$

$$k_{ij} \rightsquigarrow vk(lpk) \quad \text{with} \quad lpk = (j-1)n + i$$

c)

```
do j=1,n
  lpk0=(j-1)*n
  lpu0=(j*(j-1))/2
  do i=1,n
    lpl0=(i*(i-1))/2
    lpk=lpk0+i
    vk(lpk)=0.
    do m=1,min(i,j)
      lpl=lpl0+m
      lpu=lpu0+m
      vk(lpk)=vk(lpk)+vl(lpl)*vu(lpu)
    enddo
  enddo
enddo
```

d) The previous algorithm needs:

1) Storage $= \dfrac{n(n+1)}{2}$ terms for $\boldsymbol{L}$

$\dfrac{n(n+1)}{2}$ terms for $\boldsymbol{U}$

n^2 terms for $\boldsymbol{K}$

Total $= 2n^2 + n \quad \Rightarrow \quad \mathcal{A}(2n^2 + n) \approx \mathcal{A}(2n^2)$

2) Computing time =

$$(2(n-1)+1) \text{ times [1 product and 1 addition]} \quad \rightsquigarrow \quad \begin{bmatrix} \bullet & \bullet & \cdots & \bullet & \bullet \\ \bullet & & & & \\ \vdots & & & & \\ \bullet & & & & \\ \bullet & & & & \end{bmatrix}$$

$$(2(n-2)+1) \text{ times [2 products and 2 additions]} \quad \rightsquigarrow \quad \begin{bmatrix} * & * & \cdots & * & * \\ * & \bullet & \cdots & \bullet & \bullet \\ \vdots & \vdots & & & \\ * & \bullet & & & \\ * & \bullet & & & \end{bmatrix}$$

$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$

$$(2(n-n)+1) \text{ times [(n) products and (n) additions]} \quad \rightsquigarrow \quad \begin{bmatrix} * & * & \cdots & * & * \\ * & * & \cdots & * & * \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ * & * & \cdots & * & * \\ * & * & \cdots & * & \bullet \end{bmatrix}$$

$$\begin{aligned} \text{Total} &= \sum_{d=1}^{n} (2(n-d)+1)\,[\ (d)\ \text{“products”}\ + (d)\ \text{“additions”}\ ] \\ &= \sum_{d=1}^{n} (2(n-d)+1)\ \ (2d) \qquad \text{FPO} \quad \text{(Floating Point Operations)} \\ &= \left(\sum_{d=1}^{n} 2d(2n+1) - \sum_{d=1}^{n} (2d)^2\right) \text{ FPO} = \left((2n+1)(n+1)n - \frac{2}{3}(2n+1)(n+1)n\right) \text{ FPO} \\ &= \left(\frac{2n^3}{3} + \frac{3n^2}{3} + \frac{n}{3}\right) \text{ FPO } \Rightarrow T\left(\frac{2n^3}{3} + \frac{3n^2}{3} + \frac{n}{3}\right) \approx T\left(\frac{2n^3}{3}\right) \end{aligned}$$

If we would have used full matrices then:
$3n^2$ terms for $\boldsymbol{L}$, $\boldsymbol{U}$, $\boldsymbol{K}$ $\Rightarrow$ $\mathcal{A}(3n^2)$
$2n^3$ FPO to obtain $\boldsymbol{K}$ $\Rightarrow$ $T(2n^3)$

Thus, it is saved approximately $\left\{ \begin{array}{l} 1/3 \text{ in memory} \\ 2/3 \text{ in computing time} \end{array} \right.$

2.— Repeat the previous problem when the matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ have half-bandwidths l and u respectively, with $l << n$, and $u << n$.

Sol. 2.

$$
\boldsymbol{L} = \begin{bmatrix}
l_{11} & & & & & & & \\
l_{21} & l_{22} & & & & & & \\
\vdots & \vdots & \ddots & & & & & \\
l_{l+1,1} & l_{l+1,2} & \cdots & l_{l+1,l+1} & & & & \\
 & \ddots & \ddots & & \ddots & & & \\
 & & l_{i,i-l} & l_{i,i-l+1} & & l_{i,i} & & \\
 & & & \ddots & \ddots & & \ddots & \\
 & & & & l_{n,n-l} & l_{n,n-l+1} & \cdots & l_{n,n}
\end{bmatrix}
$$

$$
\boldsymbol{U} = \begin{bmatrix}
u_{11} & u_{12} & \cdots & u_{1,u+1} & & & & \\
 & u_{22} & \cdots & u_{2,u+1} & \ddots & & & \\
 & & \ddots & \vdots & \ddots & u_{j-u,j} & & \\
 & & & u_{u+1,u+1} & & u_{j-u+1,j} & \ddots & \\
 & & & & \ddots & \vdots & \ddots & u_{n-u,n} \\
 & & & & & u_{j,j} & & u_{n-u+1,n} \\
 & & & & & & \ddots & \vdots \\
 & & & & & & & u_{n,n}
\end{bmatrix}
$$

a) $\boldsymbol{K} = [k_{ij}]; \quad k_{ij} = \sum\limits_{m=1}^{n} l_{im} u_{mj}$

however

$$
\begin{aligned}
l_{im} &= 0 \quad \text{if} \quad m > i \quad \text{or} \quad m < i - l \\
u_{mj} &= 0 \quad \text{if} \quad m > j \quad \text{or} \quad m < j - u
\end{aligned}
$$

So that:

$$
K_{ij} = \sum_{\substack{m=1,n \\ i-l \le m \le i \\ j-u \le m \le j}} l_{im} u_{mj} \qquad = \sum_{m=\max\{i-l,j-u,1\}}^{\min\{i,\ j\}} l_{im} u_{mj}
$$

Thus whenever $\min\{i,\ j\} < \max\{i-l, j-u, 1\}$ there will be no products adding terms to k_{ij}. Therefore, only coefficient k_{ij} such that $\min\{i,\ j\} \ge \max\{i-l,\ j-u,\ 1\}$ would be non-zero, a priori, so those that $i \ge i-l,\ i \ge j-u,\ j \ge i-l,\ j \ge j-u$.

Since $i \ge i-l$ and $j \ge j-u$ are always satisfied:

$$
\boxed{k_{ij} \text{ might be non-zero } \iff \begin{cases} i \geq j-u \;\Rightarrow\; \text{uper bandwidth (u)} \\ j \geq i-l \;\Rightarrow\; \text{lower bandwidth (l)} \end{cases}}
$$

Then $\boldsymbol{K}$ is a banded matrix with lower bandwidth that of $\boldsymbol{L}$ and upper bandwidth that of $\boldsymbol{U}$.

$$
\boldsymbol{K} = \begin{array}{c} \\ \\ \\ l \\ \\ \\ \\ \\ \\ \\ \\ \end{array}
\begin{bmatrix}
\ddots & * & \cdots & * & & & & & & & \\
* & \ddots & * & \cdots & * & & & & & & \\
\vdots & * & \ddots & * & \cdots & * & & & & & \\
\vdots & \vdots & * & \ddots & * & \cdots & * & & & & \\
* & \vdots & \vdots & * & \ddots & * & \cdots & * & & & \\
 & * & \vdots & \vdots & * & \ddots & * & \cdots & * & & \\
 & & * & \vdots & \vdots & * & \ddots & * & \cdots & * & \\
 & & & * & \vdots & \vdots & * & \ddots & * & \cdots & * \\
 & & & & * & \vdots & \vdots & * & \ddots & * & \vdots \\
 & & & & & * & \vdots & \vdots & * & \ddots & * \\
 & & & & & & * & \cdots & \cdots & * & \ddots
\end{bmatrix}
\begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ u \\ \\ \\ \end{array}
$$

b) We store matrices $\boldsymbol{L}$, $\boldsymbol{U}$, $\boldsymbol{K}$ banded (by diagonals)

$$
\begin{array}{lll}
l_{ij} \rightsquigarrow vl(lpl) & \text{with} \quad lpl = (j-i+l)n+i; & i-l \leq j \leq i \\
u_{ij} \rightsquigarrow vu(lpu) & \text{with} \quad lpu = (j-i)n+i; & j-u \leq i \leq j \\
k_{ij} \rightsquigarrow vk(lpk) & \text{with} \quad lpk = (j-i+l)n+i; & \begin{cases} i-l \leq j \leq i \\ j-u \leq i \leq j \end{cases}
\end{array}
$$

c)

```
do j=1,n
  do i=max(1,j-u), min(n,j+l)
    lpk=(j-i+l)*n+i
    vk(lpk)=0.d+00
    do k=max(i-l,j-u,1), min(i,j)
      lpl=(k-i+l)*n+i
      lpu=(j-k)*n+k
      vk(lpk)=vk(lpk)+vl(lpl)*vu(lpu)
    enddo
  enddo
enddo
```

d) The algorithm requires:

1) Storage =

$$
\begin{array}{ll}
n(l+1) & \text{elements for } \boldsymbol{L} \\
n(u+1) & \text{elements for } \boldsymbol{U} \\
\underline{\quad n(l+1+u) \quad} & \text{elements for } \boldsymbol{K}
\end{array}
$$

$$
\text{Total} = \quad 2n(l+1+u)+n \quad \Rightarrow \quad \mathcal{A}(2n(l+1+u)+n) \;=\; \mathcal{A}(2n(l+u+1{,}5))
$$

2) Computing time

The exact computation is difficult to obtain, however we know that $l << n$ and $u << n$

If we assume that, what happens on the central rows and columns is of application for the rest (which is reasonable since the only irregular ones would be the $(1+l)$ first rows and the $(1+u)$ last columns) then we can proceed as follows.

To obtain the $(l+1+u)$ elements of row i we need:

$$\begin{cases} (1) \text{ product } + (1) \text{ addition} \\ (2) \text{ products } + (2) \text{ additions} \\ \quad\vdots \\ (\text{mín}(l,\ u){+}1) \text{ products } + (\text{mín}(l,\ u){+}1) \text{ additions} \\ \quad\vdots \\ (\text{mín}(l,\ u){+}1) \text{ products } + (\text{mín}(l,\ u){+}1) \text{ additions} \\ \quad\vdots \\ (2) \text{ products } + (2) \text{ additions} \\ (1) \text{ product } + (1) \text{ addition} \end{cases}$$

Therefore the number of operations per row is:

$$(l+1+u-\min(l,u))\ 2(\min(l,u)+1) \quad \text{OCF}$$

$$2(\max(l,u)+1)\ (\min(l,u)+1) \quad \text{OCF}$$

$$2(l+1)(u+1) \quad \text{OCF}$$

So the total computing complexity will be approximately $T(2n(l+1)(u+1))$

Remember that if full matrices were considered then:

$$\begin{cases} 3n^2 \text{ elements} & \text{for } \boldsymbol{L},\ \boldsymbol{U},\ \boldsymbol{K} \implies \mathcal{A}(3n^2) \\ 2n^3 \text{ OCF} & \text{to obtain } \boldsymbol{K} \implies T(2n^3) \end{cases}$$

Thus, when $l << n$ and $u << n$ the reduction in storage and computing time is very significant (specially in time).

3.— Of the matrix $\boldsymbol{L}$ it is known that it is lower triangular and that all its elements are null except for those located on the main diagonal (which are always nonzero) and those in the row α (which in general will be nonzero, but not necessarily), this is:

$$\boldsymbol{L} = \begin{bmatrix} l_{11} & & & & & & & \\ & l_{22} & & & & & & \\ & & \ddots & & & & & \\ l_{\alpha 1} & l_{\alpha 2} & \dots & l_{\alpha\alpha} & & & & \\ & & & & \ddots & & & \\ & & & & & l_{\beta\beta} & & \\ & & & & & & \ddots & \\ & & & & & & & l_{nn} \end{bmatrix}.$$

In order to obtain the matrix product $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$.

a) Is $\boldsymbol{A}$ going to be symmetric? And positive definite? Why?

b) What is the general shape of matrix $\boldsymbol{A}$?

c) Develop the storage schemes that are considered the most suitable for the two matrices. As matrix $\boldsymbol{A}$ is being computed, is it possible to store its coefficients in the place occupied by the corresponding coefficients of the matrix $\boldsymbol{L}$ matrix to save memory space? Why?

d) Develop an algorithm suited to perform the matrix product with the storage schemes described in the previous section.

e) How does the computational time necessary to obtain $\boldsymbol{A}$ grow as the size of the matrix increases?

Sol. 3.

a) $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$ with $det(\boldsymbol{L}) = \prod_{i=1}^{n} l_{ii} \neq 0$ is

$$\left\{ \begin{array}{l} \text{1) Symmetric, because } \boldsymbol{A}^T = (\boldsymbol{L}\boldsymbol{L}^T)^T = (\boldsymbol{L}^T)^T\,\boldsymbol{L}^T = \boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{A} \\ \text{2) positive semidefinite, because } \boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v} = \boldsymbol{v}^T(\boldsymbol{L}\boldsymbol{L}^T)\boldsymbol{v} = \\ \qquad (\boldsymbol{v}^T\boldsymbol{L})(\boldsymbol{L}^T\boldsymbol{v}) = \underbrace{(\boldsymbol{L}^T\boldsymbol{v})^T}_{\boldsymbol{w}^T}\underbrace{(\boldsymbol{L}^T\boldsymbol{v})}_{\boldsymbol{w}} = \boldsymbol{w}^T\boldsymbol{w} \geq 0 \\ \text{3) positive definite, because } \boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v} = 0 \Longleftrightarrow \boldsymbol{w} = \boldsymbol{L}^T\boldsymbol{v} = \boldsymbol{0} \overset{det(\boldsymbol{L})\neq 0}{\Longrightarrow} \boldsymbol{v} = \boldsymbol{0} \end{array} \right.$$

b)

$$\boldsymbol{L} = \begin{bmatrix} l_{11} & & & & & \\ & l_{22} & & & & \\ & & \ddots & & & \\ l_{\alpha 1} & l_{\alpha 2} & \cdots & l_{\alpha\alpha} & & \\ & & & & \ddots & \\ & & & & & l_{nn} \end{bmatrix} \qquad \boldsymbol{L}^T = \begin{bmatrix} l_{11} & & & l_{\alpha 1} & & \\ & l_{22} & & l_{\alpha 2} & & \\ & & \ddots & \vdots & & \\ & & & l_{\alpha\alpha} & & \\ & & & & \ddots & \\ & & & & & l_{nn} \end{bmatrix}$$

Then

$$
\boldsymbol{A}=\boldsymbol{L}\boldsymbol{L}^T=\begin{bmatrix}
l_{11}^2 & & & & l_{11}l_{\alpha 1} & & & \\
& l_{22}^2 & & & l_{22}l_{\alpha 2} & & & \\
& & \ddots & & \vdots & & & \\
& & & l_{\alpha-1,\alpha-1}^2 & l_{\alpha-1,\alpha-1}l_{\alpha,\alpha-1} & & & \\
& & & & \sum_{i=1}^{\alpha} l_{\alpha i}^2 & & & \\
& \text{SYM.} & & & & l_{\alpha+1,\alpha+1}^2 & & \\
& & & & & & \ddots & \\
& & & & & & & l_{nn}^2
\end{bmatrix}
$$

Thus, every element of $\boldsymbol{A}$ is zero except for those at the main diagonal , at row α (lower part) and at column α (upper part)

c) Since $\boldsymbol{A}$ is symmetric it is enough to store the lower part. In order to store $\boldsymbol{L}$ and the lower part of $\boldsymbol{A}$ we only need two arrays of $n+(\alpha-1)$ elements:

$$\boldsymbol{vl} \equiv (l_{11},\ l_{22},\ \cdots,\ l_{\alpha-1,\alpha-1},\ l_{\alpha 1},\ l_{\alpha 2},\ \cdots\ l_{\alpha\alpha},\ l_{\alpha+1,\alpha+1},\ \cdots,\ l_{nn})$$

$$\boldsymbol{va} \equiv (a_{11},\ a_{22},\ \cdots,\ a_{\alpha-1,\alpha-1},\ a_{\alpha 1},\ a_{\alpha 2},\ \cdots\ a_{\alpha\alpha},\ a_{\alpha+1,\alpha+1},\ \cdots,\ a_{nn})$$

Yes it is possible to store $\boldsymbol{A}$ over $\boldsymbol{L}$, although the elements of $\boldsymbol{L}$ (until the $l_{\alpha\alpha}$) are necessary to obtain some elements of $\boldsymbol{A}$ (for example, l_{11} is used to obtain a_{11} and $a_{\alpha 1}$) therefore the order of operations is important. We have to follow a specific order so the elements we need for further calculations are not overwritten before.

d)

do $i=1,\alpha-1$		`do i=1,ia-1`
$a_{ii}=l_{ii}^2$	$\rightsquigarrow$	`va(i) = vl(i) * vl(i)`
$a_{\alpha i}=l_{ii}\,l_{\alpha i}$	$\rightsquigarrow$	`va(ia-1+i) = vl(i) * vl(ia-1+i)`
enddo		`enddo`
		`va(2*ia-1) = 0.d+00`
		`do k=1,ia`
$a_{\alpha\alpha}=\sum_{k=1}^{\alpha} l_{\alpha k}^2$	$\rightsquigarrow$	`va(2*ia-1) = va(2*ia-1) + vl(ia-1+k) * vl(ia-1+k)`
		`enddo`
do $i=\alpha+1,n$		`do i=ia+1,n`
$a_{ii}=l_{ii}^2$	$\rightsquigarrow$	`va(ia-1+i) = vl(ia-1+i) * vl(ia-1+i)`
enddo		`enddo`

e) The number of floating point operations to obtain $\boldsymbol{A}$ is:

$$
\begin{array}{rcl}
& & (n-1)\text{ products} \\
& & (\alpha-1)\text{ products} \\
& & (\alpha)\text{ products} + (\alpha)\text{ additions} \\ \hline
\text{Total} & = & (n+2\alpha-2)\text{ products} + (\alpha)\text{ additions} \\
& = & (n+3\alpha-2)\text{ FPO} \quad\Longrightarrow\quad \text{T}(n+(3\alpha-2))
\end{array}
$$

Therefore, the time complexity grows linearly with the size of the matrix

4.— Of the matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ it is known that they are lower and upper triangular respectively, and that all their elements are null except for those located on the main diagonal, in the row α of $\boldsymbol{L}$ and in the β column of $\boldsymbol{U}$. This is:

$$
\boldsymbol{L} = \begin{bmatrix}
l_{11} & & & & & & & \\
& l_{22} & & & & & & \\
& & \ddots & & & & & \\
l_{\alpha 1} & l_{\alpha 2} & \dots & l_{\alpha\alpha} & & & & \\
& & & & \ddots & & & \\
& & & & & l_{\beta\beta} & & \\
& & & & & & \ddots & \\
& & & & & & & l_{nn}
\end{bmatrix}, \quad
\boldsymbol{U} = \begin{bmatrix}
u_{11} & & & & & u_{1\beta} & & \\
& u_{22} & & & & u_{2\beta} & & \\
& & \ddots & & & \cdot & & \\
& & & u_{\alpha\alpha} & & \cdot & & \\
& & & & \ddots & \cdot & & \\
& & & & & u_{\beta\beta} & & \\
& & & & & & \ddots & \\
& & & & & & & u_{nn}
\end{bmatrix}
$$

a) what is the general shape of the product matrix $\boldsymbol{A} = \boldsymbol{LU}$?
b) Design storage schemes suited for the three matrices
c) Write a specific algorithm including the storage schemes described for the matrix product.

Sol. 4.

a) It is a special case of product of two skyline matrices.

We know that $\boldsymbol{A} = \boldsymbol{LU}$ shares the same row profile as $\boldsymbol{L}$ and the same column profile as $\boldsymbol{U}$, so the only non-zero elements of $\boldsymbol{A}$ will be:

$$
\begin{aligned}
&a_{ii}; \quad i = 1, \dots, n \\
&a_{\alpha j}; \quad j = 1, \dots, \alpha - 1 \\
&a_{i\beta}; \quad i = 1, \dots, \beta - 1
\end{aligned}
$$

The the resulting matrix of the product will have the following shape:

$$
\begin{cases}
\text{Case } \alpha < \beta \quad \rightsquigarrow \quad \boldsymbol{A} = \begin{bmatrix}
\ddots & & \vdots & \\
\dots & \ddots & \vdots & \\
& & \ddots & \\
& & & \ddots
\end{bmatrix} \\[2em]
\text{Case } \alpha = \beta \quad \rightsquigarrow \quad \boldsymbol{A} = \begin{bmatrix}
\ddots & & \vdots & \\
& \ddots & \vdots & \\
\dots & \dots & \ddots & \\
& & & \ddots
\end{bmatrix} \\[2em]
\text{Case } \alpha > \beta \quad \rightsquigarrow \quad \boldsymbol{A} = \begin{bmatrix}
\ddots & \vdots & & \\
& \ddots & & \\
\dots & \dots & \ddots & \\
& & & \ddots
\end{bmatrix}
\end{cases}
$$

b) In order to store matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ we need to arrays of $(n+(\alpha-1))$ and $(n+(\beta-1))$ elements respectively:

$$\begin{aligned}\boldsymbol{vl} &= (l_{11},\, l_{22},\, \ldots,\, l_{\alpha-1,\alpha-1},\, l_{\alpha 1},\, l_{\alpha 2},\, \ldots,\, l_{\alpha,\alpha},\, l_{\alpha+1,\alpha+1},\, \ldots,\, l_{nn})\\ \boldsymbol{vu} &= (u_{11},\, u_{22},\, \ldots,\, u_{\beta-1,\beta-1},\, u_{1\beta},\, u_{2\beta},\, \ldots,\, u_{\beta\beta},\, u_{\beta+1,\beta+1},\, \ldots,\, u_{nn})\end{aligned}$$

To store matrix $\boldsymbol{A}$ we need a vector of $(n+(\alpha-1)+(\beta-1))$ elements.

We may, for example, arrange the data as:

$$\boldsymbol{va} = (a_{11},\, a_{22},\, \ldots,\, a_{nn},\, a_{\alpha 1},\, \ldots,\, a_{\alpha,\alpha-1},\, a_{1\beta},\, \ldots,\, a_{\beta-1,\beta})$$

c) This problem can be solve similarly to the previous one.

We need to multiply $\boldsymbol{LU}$ and identify the values of the elements of $\boldsymbol{A}$.

Considering the three possible cases ($\alpha < \beta,\ \alpha = \beta,\ \alpha > \beta$).

Finally, the elements l_{ik}, u_{kj} y a_{ij} are substituted by their indexes according to the storage scheme of the previous problem.