Scientific Computing I

Basic Operations, Formats and Matrix-Norms

Martin Köhler

Computational Methods in Systems and Control Theory (CSC) Max Planck Institute for Dynamics of Complex Technical Systems

Winter Term 2024/2025

This Lecture:

Basic Operations, Formats and Matrix-Norms

Vector Norms and Inner Products

Vector Norms and Inner Products

Definition 6.1

Let X be a linear space over the field $\mathbb{F}$. A mapping

$$\|.\| : X \to \mathbb{R},$$

with

1. $\|x\| \geqslant 0 \quad \forall x \in X,$ (positivity)
2. $\|x\| = 0 \iff x = 0,$ (definiteness)
3. $\|\alpha x\| = |\alpha| \, \|x\| \quad \forall \alpha \in \mathbb{F}, \forall x \in X,$ (homogeneity)
4. $\|x + y\| \leqslant \|x\| + \|y\| \quad \forall x, y \in X,$ (triangle inequality)

is called **norm** on X. A linear space together with a norm $(X, \|.\|_X)$ is called **normed linear space**.

Vector Norms and Inner Products

Example 6.2

Let $X = \mathbb{R}^n$, $p \in \mathbb{N}$. The functions

$$\|x\|_p := \sqrt[p]{\sum_{i=1}^{n} |x_i|^p} \qquad p \in \mathbb{N}$$

$$\|x\|_\infty := \max_i |x_i|$$

define norms on X.

Vector Norms and Inner Products

Definition 6.3

Let X be a linear space over the field $\mathbb{F} \in \{\mathbb{R}, \mathbb{C}\}$. An **inner product** on X is defined by a sesquilinear form

$$(.,.) : X \times X \to \mathbb{F}$$

with properties

1. $(x,x) \in \mathbb{R}_{\geqslant 0} \quad \forall x \in X,$ (positivity)
2. $(x,x) = 0 \iff x = 0,$ (definiteness)
3. $(x,y) = \overline{(y,x)} \quad \forall x,\ y \in X,$ (symmetry)
4. $(\alpha x + \beta y, z) = \alpha(x,z) + \beta(y,z) \quad \forall x,\ y,\ z \in X,\ \forall \alpha,\ \beta \in \mathbb{F}$ (linearity)

A linear space with an inner product $(X, (.,.))$ is called a **pre-Hilbert space**.

Vector Norms and Inner Products

Theorem 6.4

Let $(X, (.,.))$ *be a pre-Hilbert space. Then*

$$\|x\| := \sqrt{(x,x)} \quad \forall x \in X$$

defines a norm in X.

Vector Norms and Inner Products

Definition 6.5

Two norms $\|x\|_a$, $\|x\|_b$ on a linear space X are called **equivalent**, if and only if any sequence converging with respect to $\|x\|_a$ also converges with respect to $\|x\|_b$ and vice versa.

Theorem 6.6

$\|.\|_a$, $\|.\|_b$ on the linear space X are equivalent

$$\Leftrightarrow \exists \alpha, \beta > 0 : \alpha \|x\|_a \leqslant \|x\|_b \leqslant \beta \|x\|_a \quad \forall x \in X \tag{1}$$

Vector Norms and Inner Products

Idea of the proof.

"$\Leftarrow$": direct consequence of (1) applied to $x = y_n - y_\infty$ for a sequence $(y_n)_{n\in\mathbb{N}} \to y_\infty$ in either $\|.\|_a$, or $\|.\|_b$.

"$\Rightarrow$": Assume we can not find a γ such that $\|x\|_a < \gamma$ for all $x \in X$ with $\|x\|_b = 1$. Then there exists a sequence $(x_n)_{n\in\mathbb{N}}$ with $\|x_n\|_a \to \infty$ for $n \to \infty$ and $\|x_n\|_b = 1$ for all n. Now we define $y_n := \frac{x_n}{\|x_n\|_a}$, which in $\|.\|_b$ obviously converges to 0, but $\|x_n\|_a = 1$ and thus it does not converge in $\|.\|_a$, which contradicts our assumption.

Thus, we can find such $\gamma \in \mathbb{R}_{>0}$ and $\forall y \in X \backslash \{0\}$ we have

$$\|y\|_a = \left\| \|y\|_b \frac{y}{\|y\|_b} \right\|_a = \|y\|_b \left\| \frac{y}{\|y\|_b} \right\|_a \leqslant \|y\|_b \, \gamma$$

This proves the left inequality with $\alpha = \frac{1}{\gamma}$. The other half can be shown analogously.

□

Vector Norms and Inner Products

Theorem 6.7

Let X be a finite dimensional linear space over $\mathbb{R}$, or $\mathbb{C}$. All norms on X are equivalent.

Linear Operators, Operator and Matrix Norms

Vector Norms and Inner Products

Definition 6.8

Let $(X, \|.\|_X), (Y, \|.\|_Y)$ normed linear spaces. An operator $A : X \to Y$ is called

1. **continuous in** $x \in X$, if for all sequences $(x_n)_{n\in\mathbb{N}}$ in X with $x_n \to x$ for $n \to \infty$ we have
$$Ax_n \to Ax \text{ for } n \to \infty$$
2. **continuous**, if A is continuous in all $x \in X$.
3. **linear** if it fulfills
$$A(\alpha x + \beta y) = \alpha Ax + \beta Ay$$
4. **bounded** if A is linear and $\exists C \geqslant 0$, such that
$$\|Ax\|_Y \leqslant C \|x\|_X \quad \forall x \in X$$
Any C with this property are called **upper bound of** A.

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

The norms $\|.\|_X$, and $\|.\|_Y$ allow to measure distances in X and Y. We need similar norms to measure distances of matrices or linear operators mapping between them. The most important among those norms are the induced operator or matrix norms introduced in the following definition.

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

The norms $\|.\|_X$, and $\|.\|_Y$ allow to measure distances in X and Y. We need similar norms to measure distances of matrices or linear operators mapping between them. The most important among those norms are the induced operator or matrix norms introduced in the following definition.

Definition 6.9

Let $A : X \to Y$ be a linear operator $(X, \|.\|_X), (Y, \|.\|_Y)$ normed linear spaces. The **operator norm** of A is defined as

$$\|A\| := \sup_{\|x\|_X = 1} \|Ax\|_Y = \sup_{x \in X \setminus \{0\}} \frac{\|Ax\|_Y}{\|x\|_X}$$

$\|A\|$ is also called **induced operator norm.** In case A is a matrix, one also speaks of an **induced matrix norm**.

Definition 6.10

Let $(X, \|.\|_X), (Y, \|.\|_Y)$ normed linear spaces and denote the space of linear operators from X to Y by $\mathcal{L}(X, Y)$. A norm $\|.\|$ on $\mathcal{L}(X, Y)$ is called **consistent** with $\|.\|_X$ and $\|.\|_Y$, if for any $x \in X$ and $A \in \mathcal{L}(X, Y)$ we have $\|Ax\|_Y \leqslant \|A\| \, \|x\|_X$.
In case $Y = X$, i.e. $\|Ax\|_X \leqslant \|A\| \, \|x\|_X$, the norm $\|.\|$ is called **compatible** with $\|.\|_X$.

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

Definition 6.10

Let $(X, \|.\|_X), (Y, \|.\|_Y)$ normed linear spaces and denote the space of linear operators from X to Y by $\mathcal{L}(X, Y)$. A norm $\|.\|$ on $\mathcal{L}(X, Y)$ is called **consistent** with $\|.\|_X$ and $\|.\|_Y$, if for any $x \in X$ and $A \in \mathcal{L}(X, Y)$ we have $\|Ax\|_Y \leqslant \|A\| \, \|x\|_X$.
In case $Y = X$, i.e. $\|Ax\|_X \leqslant \|A\| \, \|x\|_X$, the norm $\|.\|$ is called **compatible** with $\|.\|_X$.

Remark

- The induced norms fulfill the consistency, and compatibility condition by definition.
- They are not the only norm that do so.

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

Theorem 6.11

$\|A\|$ *is the smallest upper bound of* A *and* A *is bounded if and only if* $\|A\| < \infty$.

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

Theorem 6.11

$\|A\|$ is the smallest upper bound of A and A is bounded if and only if $\|A\| < \infty$.

Proof.

"$\Rightarrow$": Let A be bounded $\rightarrow \exists \infty > C \geqslant 0$ with

$$\|Ax\|_Y \leqslant C \quad \forall x \in X, \|x\|_X = 1 \text{ and } \|A\| = \sup_{\|x\|_X = 1} \|Ax\|_Y \leqslant C < \infty.$$

Especially $\|A\| \leqslant C$ for all upper bounds C.

"$\Leftarrow$": Let A be linear with $\|A\| < \infty$. Then, for arbitrary $x \in X \backslash \{0\}$, we have

$$\|Ax\|_Y = \left\| \|x\|_X A\left(\frac{x}{\|x\|_X}\right) \right\|_Y = \|x\|_X \left\| A\left(\frac{x}{\|x\|_X}\right) \right\|_Y$$
$$\leqslant \|x\|_X \sup_{\|z\|_X = 1} \|Az\|_Y = \|x\|_X \|A\| .$$

That means, A is bounded with upper bound $\|A\|$.

□

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

$\rightarrow$ Matrices are a special type of linear operators.

Theorem 6.12

Let $(X, \|.\|_X)$ and $(Y, \|.\|_Y)$ be normed linear spaces, and $A : X \rightarrow Y$ a linear operator. The following are equivalent:

1. *A is continuous in $x = 0$*
2. *A is continuous*
3. *A is bounded*

Proof.

1⇒2: Let $x \in X$, $(x_n)_{n\in\mathbb{N}} \subseteq X$ with $x_n \to x$, $n \to \infty$

$$\Rightarrow Ax_n \overset{A\text{ linear}}{=} A\underbrace{(x_n - x)}_{\overset{\|\cdot\|_X}{\longrightarrow} 0,\ n\to\infty} + Ax \overset{\|\cdot\|_Y}{\longrightarrow} Ax \quad \text{for } n \to \infty$$

$\square$

Proof.

2⇒3: We prove this part using a contradiction argument. Assume A continuous, but unbounded. Then there exists $(x_n)_{n\in\mathbb{N}} \subseteq X$ with $\|x_n\|_X = 1$ and $\|Ax_n\| \geqslant n$. Define:

$$y_n := \frac{x_n}{\|Ax_n\|_Y}.$$

Then we immediately get

$$\|y_n\|_X = \left\|\frac{x_n}{\|Ax_n\|_Y}\right\|_X = \frac{\|x_n\|_X}{\|Ax_n\|_Y} = \frac{1}{\|Ax_n\|_Y} \leqslant \frac{1}{n}$$

and thus

$$y_n \xrightarrow{\|.\|_X} 0 \quad n \to \infty.$$

On the other hand,

$$\|Ay_n\|_Y = \left\|A\frac{x_n}{\|Ax_n\|_Y}\right\|_Y = \frac{\|Ax_n\|_Y}{\|Ax_n\|_Y} = 1$$

for all $n \in \mathbb{N}$, which contradicts continuity of A in $x = 0$.

□

Proof.

3⇒1: Let A be bounded and $(x_n)_{n\in\mathbb{N}} \subseteq X$ with $x_n \overset{\|.\|_X}{\longrightarrow} 0$ for $n \to \infty$. Then

$$\|Ax_n\|_Y \leqslant \|A\| \, \|x_n\|_X \to 0 \quad \text{as} \quad n \to \infty$$

and thus A continuous in $x = 0$.

$\square$

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

Lemma 6.13 (Submultiplicativity)

Let $(X, \|.\|_X)$, $(Y, \|.\|_Y)$, $(Z, \|.\|_Z)$ *be normed linear spaces.*

$$\begin{aligned} A :& X \to Y \\ B :& Y \to Z \end{aligned}$$

bounded linear operators, then the operator concatenation

$$BA : X \to Z$$

is bounded with

$$\|BA\| \leqslant \|B\| \, \|A\| \,. \tag{2}$$

Vector Norms and Inner Products

Proof.

First we note that for any $x \in X$ due to boundedness of A and B we have

$$\|BAx\| \leqslant \|B\| \, \|Ax\|_Y \leqslant \|B\| \, \|A\| \, \|x\|_X$$

The lemma, thus, is a direct consequence of

$$\begin{aligned}\|BA\| = \sup_{\|x\|_X = 1} \|BAx\| &\leqslant \sup_{\|x\|_X = 1} \|B\| \, \|Ax\|_Y \\ &\leqslant \sup_{\|x\|_X = 1} \|B\| \, \|A\| \, \|x\|_X = \|B\| \, \|A\|\end{aligned}$$

□

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

$\rightarrow$ A bounded linear op. on finite dimensional linear spaces can always be expressed as a matrix.

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

→ A bounded linear op. on finite dimensional linear spaces can always be expressed as a matrix.

Definition 6.14

Given

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \in \mathbb{R}^{n \times m},$$

1. the **transposed** matrix A^T is defined as

$$A^\mathsf{T} = \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \cdots & a_{nm} \end{bmatrix} \in \mathbb{R}^{m \times n},$$

2. If $A^\mathsf{T} = A$, then A is called **symmetric** $(n = m)$
3. If $A^\mathsf{T} A = I$, then A is called **orthogonal** $(n \geqslant m)$
4. If $A^\mathsf{T} A = AA^\mathsf{T}$, then A is called **normal** $(n = m)$

Definition 6.15

1. Given
$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \in \mathbb{C}^{n\times m},$$
the **conjugate transposed** matrix A^{H} is defined as
$$A^{\mathrm{H}} = \begin{bmatrix} \overline{a_{11}} & \cdots & \overline{a_{n1}} \\ \vdots & \ddots & \vdots \\ \overline{a_{1m}} & \cdots & \overline{a_{nm}} \end{bmatrix} \in \mathbb{C}^{m\times n},$$
2. If $A^{\mathrm{H}} = A$, then A is called **hermitian** $(n = m)$
3. If $A^{\mathrm{H}}A = I$, then A is called **unitary** $(n \geqslant m)$
4. If $A^{\mathrm{H}}A = AA^{\mathrm{H}}$, then A is called **normal** $(n = m)$

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

Definition 6.16

Let $X = \mathbb{R}^n$, or $X = \mathbb{C}^n$. A matrix $A : X \to X$ is called

1. **upper triangular**, if $a_{ij} = 0 \ \forall i > j$,
2. **lower triangular**, if $a_{ij} = 0 \ \forall i < j$,
3. **diagonal**, if $a_{ij} = 0 \ \forall i \neq j$,
4. **positive semidefinite** if $(Ax, x)_2 \geqslant 0 \ \forall x \in X$,
5. **positive definite** if $(Ax, x)_2 > 0 \ \forall x \in X \backslash \{0\}$,
6. **negative (semi)definite** if $-A$ is positive (semi)definite.

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

Lemma 6.17

Let $P \in \mathbb{C}^{n\times n}$ be invertible and $A \in \mathbb{C}^{n\times n}$, then the linear systems of equations $Ax = y$ and $PAx = Py$ for $x, y \in \mathbb{C}^n$ are equivalent.

Proof.

$$\begin{aligned} P \text{ is invertible} &\Rightarrow \text{“}Px = 0 \iff x = 0\text{”} \\ &\Rightarrow \text{“}P(Ax - y) = 0 \iff Ax - y = 0\text{”} \end{aligned}$$

□

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

Lemma 6.18

The linear system $Ax = b$ permits a solution if and only if $\text{rank}(A) = \text{rank}([A\ b])$

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

Lemma 6.18

The linear system $Ax = b$ *permits a solution if and only if* $\operatorname{rank}(A) = \operatorname{rank}([A\ b])$

Lemma 6.19

Products of lower (upper) triangular matrices are lower (upper) triangular.

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

Lemma 6.18

The linear system $Ax = b$ permits a solution if and only if $\operatorname{rank}(A) = \operatorname{rank}([A\ b])$

Lemma 6.19

Products of lower (upper) triangular matrices are lower (upper) triangular.

Lemma 6.20

Products of orthogonal matrices are orthogonal matrices.

Vector Norms and Inner Products

Linear Operators, Operator and Matrix Norms

Some matrix norm examples:

1. $\|A\| := \max\limits_{i,j} |a_{ij}|$ (induced by the pair $(\|.\|_1, \|.\|_\infty)$ of norms, not sub-multiplicative)
2. $\|A\|_F := \sqrt{\sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} \left|a_{ij}^2\right|}$ (not induced, compatible with the vector $\|.\|_2$-norm, **Frobenius Norm**)
3. $\|A\|_1 := \max\limits_{j=1,\ldots,n} \sum\limits_{i=1}^{n} |a_{ij}|$ (induced, **column sum norm**)
4. $\|A\|_\infty := \max\limits_{i=1,\ldots,n} \sum\limits_{j=1}^{n} |a_{ij}|$ (induced, **row sum norm**)
5. $\|A\|_2 := \sup\limits_{\|x\|_2=1} \|Ax\|_2$ (induced, **spectral norm)**

Vector Norms and Inner Products

Some matrix norm examples:

1. $\|A\| := \max\limits_{i,j} |a_{ij}|$ (induced by the pair $(\|.\|_1, \|.\|_\infty)$ of norms, not sub-multiplicative)
2. $\|A\|_F := \sqrt{\sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} \left|a_{ij}^2\right|}$ (not induced, compatible with the vector $\|.\|_2$-norm, **Frobenius Norm**)
3. $\|A\|_1 := \max\limits_{j=1,\dots,n} \sum\limits_{i=1}^{n} |a_{ij}|$ (induced, **column sum norm**)
4. $\|A\|_\infty := \max\limits_{i=1,\dots,n} \sum\limits_{j=1}^{n} |a_{ij}|$ (induced, **row sum norm**)
5. $\|A\|_2 := \sup\limits_{\|x\|_2=1} \|Ax\|_2$ (induced, **spectral norm)**

Theorem 6.21

Any matrix $A \in \mathbb{C}^{n \times n}$ is bounded in every matrix norm.

Spectral Norm and Spectral Radius

Vector Norms and Inner Products

Spectral Norm and Spectral Radius

A complex number $\lambda \in \mathbb{C}$ is called **eigenvalue** of a matrix A if $\exists x \neq 0$

$$Ax = \lambda x$$

Then x is called **(right) eigenvector** of A. The set of all eigenvalues is $\Lambda(A) := \{\lambda \in \mathbb{C} : Ax = \lambda x\}$, it is called **spectrum** of A. The value $\rho(A) = \max\{|\lambda| : \lambda \in \Lambda(A)\}$ is called the **spectral radius** of A.

Vector Norms and Inner Products

Spectral Norm and Spectral Radius

A complex number $\lambda \in \mathbb{C}$ is called **eigenvalue** of a matrix A if $\exists x \neq 0$

$$Ax = \lambda x$$

Then x is called **(right) eigenvector** of A. The set of all eigenvalues is $\Lambda(A) := \{\lambda \in \mathbb{C} : Ax = \lambda x\}$, it is called **spectrum** of A. The value $\rho(A) = \max\{|\lambda| : \lambda \in \Lambda(A)\}$ is called the **spectral radius** of A.

Remark

In the following A^* denotes either A^{H} when A is complex or A^{T} when it is real.

Vector Norms and Inner Products

Spectral Norm and Spectral Radius

Theorem 6.22 (Schur decomposition)

Let $A \in \mathbb{C}^{n \times n}$ $(\mathbb{R}^{n \times n})$. There exists a unitary (orthogonal) matrix $U \in \mathbb{C}^{n \times n}$ $(\mathbb{R}^{n \times n})$ such that

$$T = U^* A U$$

is a (quasi) upper triangular matrix.

Vector Norms and Inner Products

Theorem 6.22 (Schur decomposition)

Let $A \in \mathbb{C}^{n\times n}$ $(\mathbb{R}^{n\times n})$. There exists a unitary (orthogonal) matrix $U \in \mathbb{C}^{n\times n}$ $(\mathbb{R}^{n\times n})$ such that

$$T = U^*AU$$

is a (quasi) upper triangular matrix.

Remark

- $\Lambda(A) = \{t_{ii} : i = 1, \dots, n\}$ $(A \in \mathbb{C}^{n\times n})$, where t_{ii} are the diagonal entries in T from the above Theorem.
- The Schur decomposition can be computed in a QR-algorithm in $\mathcal{O}(n^3)$ floating point operations.

Vector Norms and Inner Products

Spectral Norm and Spectral Radius

Corollary 6.23

Let $A \in \mathbb{C}^{n\times n}$ $(\mathbb{R}^{n\times n})$ hermitian (symmetric). There exists a unitary (orthogonal) matrix $U \in \mathbb{C}^{n\times n}$ $(\mathbb{R}^{n\times n})$ such that

$$\begin{bmatrix} \diagdown \end{bmatrix} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n) = U^* A U$$

Here λ_i $(i = 1, \ldots, n)$ is the i-th eigenvalue of A with the i-th column of U the corresponding eigenvector.

Vector Norms and Inner Products

Spectral Norm and Spectral Radius

Theorem 6.24

The $\|.\|_2$ operator norm of A is called spectral norm since we have:

1. $\|A\|_2 = \sqrt{\rho(A^*A)}$
2. $\rho(A) \leqslant \|A\|$ *for an arbitrary induced norm* $\|.\|$
3. $A = A^* \Rightarrow \rho(A) = \|A\|_2$

Vector Norms and Inner Products

Spectral Norm and Spectral Radius

Proof.

i) $(A^*A) = (A^*A)^*$ thus Corollary 6.23 tells us that there exists an orthogonal matrix U with

$$U^*A^*AU = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$$

Further, for all $x \in \mathbb{C}^n$ we find coefficients α_i, $(i = 1, \ldots, n)$, such that

$$x = \sum_{i=1}^{n} \alpha_i u_i.$$

Vector Norms and Inner Products

Spectral Norm and Spectral Radius

Proof.

Thus,

$$A^*Ax = \sum_{i=1}^{n} \lambda_i \alpha_i u_i,$$

and therefore

$$\begin{aligned} \|Ax\|_2^2 &= (Ax, Ax)_2 = (x, A^*Ax)_2 \\ &= \left(\sum \alpha_i u_i, \sum \lambda_i \alpha_i u_i\right)_2 \\ &= \sum (\alpha_i u_i, \lambda_i \alpha_i u_i)_2 \\ &= \sum \lambda_i |\alpha_i|^2 (u_i, u_i)_2 \end{aligned}$$

Vector Norms and Inner Products

Spectral Norm and Spectral Radius

Proof.

$$
\begin{aligned}
&= \sum \lambda_i |\alpha_i|^2 \, \|u\|_2^2 \\
&= \sum \lambda_i |\alpha_i|^2 \\
&\leqslant \rho(A^* A) \sum |\alpha_i|^2 \\
&= \rho(A^* A) \, \|x\|_2^2 \, ,
\end{aligned}
$$

such that

$$\frac{\|Ax\|_2}{\|x\|_2} \leqslant \rho(A^* A)$$

and $\lambda_i \geqslant 0 \forall i$.

Vector Norms and Inner Products

Spectral Norm and Spectral Radius

Proof.

Now let $\lambda_{i_0} = \rho(A^*A)$, and u_{i_0} the corresponding eigenvector, then

$$\frac{\|Au_{i_0}\|_2^2}{\|u_{i_0}\|_2^2} = \frac{\lambda_{i_0}\,\|u_{i_0}\|_2^2}{\|u_{i_0}\|_2^2} = \lambda_{i_0} = \rho(A^*A).$$

So we have proved the first statement.

Vector Norms and Inner Products

Spectral Norm and Spectral Radius

Proof.

ii) By definition of the induced norm we have for each pair of eigenvalue λ and corresponding eigenvector u that

$$\|A\| = \sup_{\|x\|=1} \|Ax\| \geqslant \|Au\| = \|\lambda u\| = |\lambda| \, \|u\| = |\lambda|,$$

and therefore $\rho(A) \leqslant \|A\|$.

iii) $A^* = A$:

$$\|A\|_2 = \sqrt{\rho(A^* A)} = \sqrt{\rho(A^2)} = \sqrt{\rho(A)^2} = \rho(A)$$

$\square$

Condition Number and Singular Values

Vector Norms and Inner Products

Condition Number and Singular Values

Recall the condition number:

$$c_{\mathrm{rel}}(f, x) \leqslant \frac{\|x\|}{\|f(x)\|} \cdot \|f'(x)\| \, .$$

Vector Norms and Inner Products

Condition Number and Singular Values

Recall the condition number:

$$c_{\mathrm{rel}}(f,x) \leqslant \frac{\|x\|}{\|f(x)\|} \cdot \left\|f'(x)\right\|.$$

Now let $f \equiv A$ and A invertible $\Rightarrow$

$$y = Ax \Leftrightarrow x = A^{-1}y$$

$$\Rightarrow \frac{\|x\|}{\|f(x)\|} = \frac{\|x\|}{\|Ax\|} = \frac{\left\|A^{-1}y\right\|}{\|y\|} \leqslant \sup_{y \neq 0} \frac{\left\|A^{-1}y\right\|}{\|y\|} = \left\|A^{-1}\right\|.$$

Since the Jacobian of a linear operator is the linear operator, we have

$$f'(x) = A\Big|_{x}.$$

Such that we find

$$c_{\mathrm{rel}}(A,x) \leqslant \|A\| \left\|A^{-1}\right\|.$$

Vector Norms and Inner Products

Condition Number and Singular Values

Using $A = I = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}$ we further have

$$c_{\text{rel}} = \frac{\|x\|}{\|x\|} \|I\| = 1 = \|I\| \left\|I^{-1}\right\|,$$

which proves that the bound is indeed sharp.

Vector Norms and Inner Products

Condition Number and Singular Values

Using $A = I = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}$ we further have

$$c_{\mathrm{rel}} = \frac{\|x\|}{\|x\|} \|I\| = 1 = \|I\| \left\|I^{-1}\right\|,$$

which proves that the bound is indeed sharp.

Definition 6.25

Let $A \in \mathbb{C}^{n \times n}$ and $\|.\|_a$ an induced operator norm

$$\kappa_a(A) := \|A\|_a \left\|A^{-1}\right\|_a$$

denotes the a-**condition number** of A.

Vector Norms and Inner Products

Condition Number and Singular Values

Lemma 6.26

For any induced operator norm $\|.\|_a$ it holds

$$\kappa_a(A) \geqslant \kappa_a(I) = 1$$

Vector Norms and Inner Products

Condition Number and Singular Values

Lemma 6.26

For any induced operator norm $\|.\|_a$ it holds

$$\kappa_a(A) \geqslant \kappa_a(I) = 1$$

Proof.

$$\kappa_a(I) = \|I\|_a \left\|I^{-1}\right\|_a = 1 = \|I\|_a = \left\|AA^{-1}\right\|_a$$

$$\overset{\text{Submultiplicativity}}{\leqslant} \|A\|_a \left\|A^{-1}\right\|_a = \kappa_a(A)$$

$\square$

Vector Norms and Inner Products

Condition Number and Singular Values

Theorem 6.27

Let $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$. Let x be the exact solution of $Ax = b$ and $x + \Delta x$ the exact solution of the perturbed $A(x + \Delta x) = b + \Delta b$. Then

$$\frac{\|\Delta x\|}{\|x\|} \leqslant \kappa(A)\frac{\|\Delta b\|}{\|b\|}.$$

Vector Norms and Inner Products

Condition Number and Singular Values

Theorem 6.27

Let $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$. Let x be the exact solution of $Ax = b$ and $x + \Delta x$ the exact solution of the perturbed $A(x + \Delta x) = b + \Delta b$. Then

$$\frac{\|\Delta x\|}{\|x\|} \leqslant \kappa(A) \frac{\|\Delta b\|}{\|b\|}.$$

Theorem 6.28

Let $Ax = b$, as above. Moreover define the error $e_k := A^{-1}b - x_k$, and the residual $r_k := b - Ax_k$ in step k of an iterative solver for $Ax = b$. It holds:

$$\frac{1}{\kappa(A)} \frac{\|r_k\|}{\|r_0\|} \leqslant \frac{\|e_k\|}{\|e_0\|} \leqslant \kappa(A) \frac{\|r_k\|}{\|r_0\|} \leqslant \kappa(A)^2 \frac{\|e_k\|}{\|e_0\|}. \tag{3}$$

Vector Norms and Inner Products

Condition Number and Singular Values

Proof.

Note

$$\|r_k\| = \|b - Ax_k\| = \left\|A(A^{-1}b - x_k)\right\| = \|Ae_k\| \leqslant \|A\| \, \|e_k\|$$

and analogously

$$\|e_k\| = \left\|A^{-1}b - x_k\right\| \leqslant \left\|A^{-1}\right\| \, \|r_k\|$$

Thus

$$\frac{1}{\kappa(A)} \frac{\|r_k\|}{\|r_0\|} = \frac{1}{\|A\| \, \|A^{-1}\|} \frac{\|r_k\|}{\|r_0\|} \leqslant \frac{1}{\|A\|} \frac{\|r_k\|}{\|A^{-1}r_0\|} = \frac{1}{\|A\|} \frac{\|Ae_k\|}{\|e_0\|} \leqslant \frac{\|e_k\|}{\|e_0\|}.$$

This proves the leftmost inequality in (3). The others can be shown similarly. □

Some Remarks on $\kappa_2(A)$

Vector Norms and Inner Products

Some Remarks on $\kappa_2(A)$

Theorem 6.29

Let $A \in \mathbb{R}^{n\times n}$. There exist orthogonal matrices $U, V \in \mathbb{R}^{n\times n}$ such that

$$U^\mathsf{T} A V = \begin{pmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_n \end{pmatrix} \tag{4}$$

where $0 \leqslant \sigma_n \leqslant \cdots \leqslant \sigma_1$. For $i = 1, \ldots, n$ we further have

$$\det(A^\mathsf{T} A - \sigma_i^2 I) = 0 \tag{5}$$

i.e. $\sigma_i^2 = \lambda_i$ with $\lambda_i \in \Lambda(A^\mathsf{T} A)$.

Vector Norms and Inner Products

Some Remarks on $\kappa_2(A)$

Proof.

$A^\mathsf{T}A$ is symmetric and positive semidefinite, so there exists $V \in \mathbb{R}^{n\times n}$, such that

$$V^\mathsf{T}A^\mathsf{T}AV = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$$

where $\lambda_1 \geqslant \cdots \geqslant \lambda_n \geqslant 0$. Thus $\sigma_i = \sqrt{\lambda_i}$ is well defined in Theorem 6.29 and (5) follows from Corollary 6.23. For (4) we define $U = AVD^{-1}$, where $D = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$. Since we have

$$U^\mathsf{T}U = D^{-T}V^\mathsf{T}A^\mathsf{T}AVD^{-1} = D^{-1}\operatorname{diag}(\lambda_1, \ldots, \lambda_n)\,D^{-1} = I$$

U is ortogonal and

$$U^\mathsf{T}AV = D^{-T}V^\mathsf{T}A^\mathsf{T}AV = D^{-1}\operatorname{diag}(\lambda_i) = D$$

$\square$

$\rightarrow$ If A is invertible we have $\sigma_n > 0$ and $\lambda_n > 0$.

Vector Norms and Inner Products

Some Remarks on $\kappa_2(A)$

Definition 6.30

The σ_i in the last Theorem are called **singular values** of A. The corresponding columns in U, V are called the i-th left/right **singular vectors**.

Vector Norms and Inner Products

Some Remarks on $\kappa_2(A)$

Definition 6.30

The σ_i in the last Theorem are called **singular values** of A. The corresponding columns in U, V are called the i-th left/right **singular vectors**.

Now from

$$\begin{aligned}\sup_{x\neq 0}\frac{\|Ax\|_2^2}{\|x\|_2^2} &= \sup_{x\neq 0}\frac{(Ax,Ax)_2}{(x,x)_2} = \sup_{x\neq 0}\frac{x^\mathsf{T}A^\mathsf{T}Ax}{x^\mathsf{T}x}\\ &\overset{V\text{ reg.}}{=} \sup_{Vx\neq 0}\frac{x^\mathsf{T}V^\mathsf{T}A^\mathsf{T}AVx}{x^\mathsf{T}V^\mathsf{T}Vx}\\ &\overset{U,V\text{ orth.}}{=} \sup_{x\neq 0}\frac{x^\mathsf{T}V^\mathsf{T}A^\mathsf{T}UU^\mathsf{T}AVx}{x^\mathsf{T}x} = \sup_{x\neq 0}\frac{x^\mathsf{T}D^\mathsf{T}Dx}{x^\mathsf{T}x} = \sigma_1^2,\end{aligned}$$

we analogously find for the infimum

$$\inf_{x\neq 0}\frac{\|Ax\|_2}{\|x\|_2} = \sigma_n.$$

Vector Norms and Inner Products

Some Remarks on $\kappa_2(A)$

Further we have

$$U^\mathsf{T} A V = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)\,,$$

and

$$V^\mathsf{T} A^{-1} U = \operatorname{diag}\left(\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_n}\right)$$

and thus $\|A\|_2 = \sigma_1$ and $\left\|A^{-1}\right\|_2 = \frac{1}{\sigma_n}$, which proves the following Corollary.

Vector Norms and Inner Products

Some Remarks on $\kappa_2(A)$

Further we have

$$U^\mathsf{T} A V = \operatorname{diag}(\sigma_1, \dots, \sigma_n)\,,$$

and

$$V^\mathsf{T} A^{-1} U = \operatorname{diag}\left(\frac{1}{\sigma_1}, \dots, \frac{1}{\sigma_n}\right)$$

and thus $\|A\|_2 = \sigma_1$ and $\left\|A^{-1}\right\|_2 = \frac{1}{\sigma_n}$, which proves the following Corollary.

Corollary 6.31

Let $A \in \mathbb{R}^{n\times n}$ invertible, σ_1, σ_n its largest and smallest singular values, then we have

$$\kappa_2(A) = \frac{\sigma_1}{\sigma_n}$$

If A is in addition normal and λ_1 and λ_n are its largest and smallest magnitude eigenvalues, then we also have

$$\kappa_2(A) = \frac{|\lambda_1|}{|\lambda_n|}$$

Vector Norms and Inner Products

Some Remarks on $\kappa_2(A)$

Definition 6.32

$\|.\|_a, \|.\|_b$ vector norms on $\mathbb{R}^n$. The condition numbers κ_a, κ_b are called **equivalent** if one can find $\alpha, \beta > 0$ such that

$$\alpha\kappa_a(A) \leqslant \kappa_b(A) \leqslant \beta\kappa_a(A) \qquad \forall A \in \mathbb{R}^{n\times n} \text{ invertible}$$

Vector Norms and Inner Products

Some Remarks on $\kappa_2(A)$

Definition 6.32

$\|.\|_a, \|.\|_b$ vector norms on $\mathbb{R}^n$. The condition numbers κ_a, κ_b are called **equivalent** if one can find $\alpha, \beta > 0$ such that

$$\alpha \kappa_a(A) \leqslant \kappa_b(A) \leqslant \beta \kappa_a(A) \qquad \forall A \in \mathbb{R}^{n \times n} \text{ invertible}$$

$\rightarrow$ Similar to the norm-equivalence, the constants α and β coincide with the constants α, β for equivalence for norms.

Matrix Storage Formats

Matrix Storage Formats

We need data structures to store matrices in a proper way on a computer. Thereby, we have to take care about “meta” information like

- the dimensions,
- structure (lower/upper triangular)
- a large number of zero entries,
- symmetry, ...

Matrix Storage Formats

We need data structures to store matrices in a proper way on a computer. Thereby, we have to take care about "meta" information like

- the dimensions,
- structure (lower/upper triangular)
- a large number of zero entries,
- symmetry, ...

For illustrating different approaches we use:

$$A = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 5 & 0 & 6 \\ 0 & 0 & 7 & 0 \end{bmatrix}.$$

Dense Matrices

Matrix Storage Formats

Dense Matrices

Definition 6.33

A matrix is called **dense**, or **densely populated** if essentially all its entries are non-zero.

Alternative: A matrix is called **dense** if there is no benefit from handling zero entries differently.

Matrix Storage Formats

Dense Matrices

Definition 6.33

A matrix is called **dense**, or **densely populated** if essentially all its entries are non-zero.

Alternative: A matrix is called **dense** if there is no benefit from handling zero entries differently.

We need to store a 2d object of $m \times n$ entries in memory, where each entry (k, l) can be accessed directly.

Matrix Storage Formats

Dense Matrices – 2d Arrays in C

Using the C array definitions we can realize a 2d array in two ways:

- **`double A[5][10]`** (static array),
- **`double **A + malloc()`** (dynamic array).

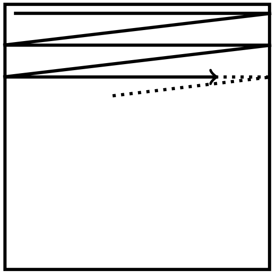
Matrix Storage Formats

Dense Matrices – 2d Arrays in C

Using the C array definitions we can realize a 2d array in two ways:

- `double A[5][10]` (static array),
- `double **A + malloc()` (dynamic array).

In both cases it is stored **"row major"**, i.e., the order of elements follows the model:



Matrix Storage Formats

Dense Matrices – 2d Arrays in C

Static Arrays in C:

A static array in C is essentially one big row vector:

`double A[5][10]`

$a_{00}, \dots, a_{09}$	$a_{10}, \dots, a_{19}$	$a_{20}, \dots, a_{29}$	$a_{30}, \dots, a_{39}$	$a_{40}, \dots$

- the size of static arrays is limited by the size of the stack, typically 8MiB. $\rightarrow$ maximum size 1024×1024 in double precision
- continuous in memory
- size must be known a-priori

Matrix Storage Formats

Dense Matrices – 2d Arrays in C

Static Arrays in C:

A static array in C is essentially one big row vector:

`double A[5][10]`

$$\boxed{a_{00},\dots,a_{09}}\boxed{a_{10},\dots,a_{19}}\boxed{a_{20},\dots,a_{29}}\boxed{a_{30},\dots,a_{39}}\boxed{a_{40},\dots}$$

- the size of static arrays is limited by the size of the stack, typically 8MiB.
  $\rightarrow$ maximum size 1024×1024 in double precision
- continuous in memory
- size must be known a-priori

Example 6.34

Our example A turns into:

$$A = \boxed{1\ 2\ 0\ 0}\boxed{0\ 3\ 4\ 0}\boxed{0\ 5\ 0\ 6}\boxed{0\ 0\ 7\ 0}$$

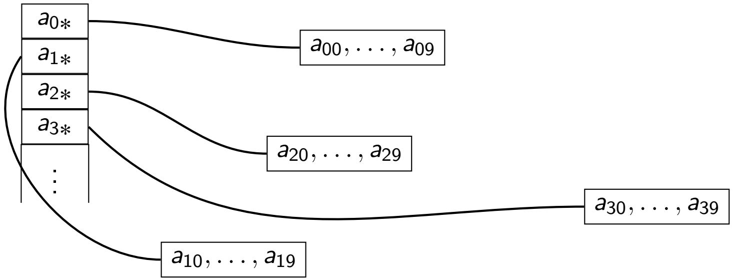
Matrix Storage Formats

Dense Matrices – 2d Arrays in C

Dynamic 2d Arrays in C:

In the design of C, a dynamically allocated 2d array results in an array of single arrays:

```
double **A;
```



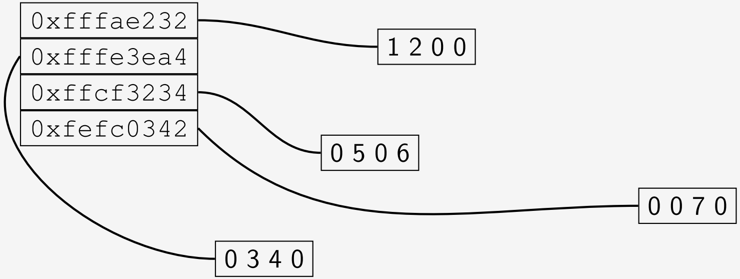
- size can be determined at runtime
- not continuous in memory
- swapping rows is easy/cheap

Matrix Storage Formats

Dense Matrices – 2d Arrays in C

Example 6.35

Again, our example A turns into:

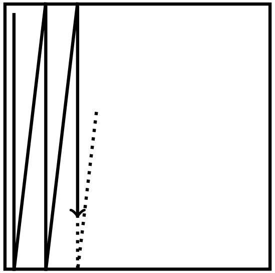


Matrix Storage Formats

Dense Matrices – 2d Arrays in Fortran

→ C was designed for system and hardware programming, Fortran for solving the solution of mathematical problems.

All arrays (static and dynamic) are stored in **“column major”**:



which leads to

$a_{00}, \dots, a_{n0}$	$a_{01}, \dots, a_{n1}$	$a_{02}, \dots, a_{n2}$	$a_{03}, \dots, a_{n3}$	$a_{04}, \dots$

Matrix Storage Formats

Dense Matrices – 2d Arrays in Fortran

- continuous in memory
- no difference between static and dynamic arrays
- used by the majority of linear algebra software
- can be emulated in C by using an one dimensional array and an index transformation function

Matrix Storage Formats

Dense Matrices – 2d Arrays in Fortran

- continuous in memory
- no difference between static and dynamic arrays
- used by the majority of linear algebra software
- can be emulated in C by using an one dimensional array and an index transformation function

Example 6.36

In Fortran storage, our A turns into:

$$\boxed{1\ 0\ 0\ 0}\boxed{2\ 3\ 5\ 0}\boxed{0\ 4\ 0\ 7}\boxed{0\ 0\ 6\ 0}$$

Matrix Storage Formats

Dense Matrices – 2d Arrays in Fortran

- continuous in memory
- no difference between static and dynamic arrays
- used by the majority of linear algebra software
- can be emulated in C by using an one dimensional array and an index transformation function

Example 6.36

In Fortran storage, our A turns into:

$$\boxed{1\ 0\ 0\ 0}\boxed{2\ 3\ 5\ 0}\boxed{0\ 4\ 0\ 7}\boxed{0\ 0\ 6\ 0}$$

Remark

We stick to Fortran “column-major” storage for our applications.

Matrix Storage Formats

Dense Matrices

Definition 6.37

The distance, counted in the number of elements, between the beginnings of 2 subsequent columns in a 2d array is called the **leading dimension** (LD) of the array.

$$\Rightarrow a_{kl} \,\hat{=}\, A[l \cdot LD + k]$$

Matrix Storage Formats

Dense Matrices

Definition 6.37

The distance, counted in the number of elements, between the beginnings of 2 subsequent columns in a 2d array is called the **leading dimension** (LD) of the array.

$$\Rightarrow a_{kl} \hat{=} A[l \cdot LD + k]$$

C-Caveats

C used 0-based indexing, Fortran (and mathematicians) 1-based indexing.

If C indexing is used, the location of (k, l) in A is determined via

```
#define IDX2C(k,l,lda) ( (k) + (l) * (lda) )
```

If Fortran indexing is used, the location of (k, l) in A is determined via

```
#define IDX2F(k,l,lda) ( ((k)-1) + ((l)-1) * (lda) )
#define IDX1F(k) ((k)-1)
```

Matrix Storage Formats

Dense Matrices

Fortran does the mapping between $A(k, l)$ automatically. The leading dimension is set by the expression **`double precision A(LD, *)`**.

- Data locality is enforced also for dynamic arrays since the single row/column pointers can no longer be scattered around the main memory.
- More importantly, the array is now stored in Fortran 77 compliant column major format and can thus be passed directly to (optimized) Fortran libraries.

Matrix Storage Formats

Dense Matrices

Fortran does the mapping between $A(k, l)$ automatically. The leading dimension is set by the expression **`double precision A(LD, *)`**.

- Data locality is enforced also for dynamic arrays since the single row/column pointers can no longer be scattered around the main memory.
- More importantly, the array is now stored in Fortran 77 compliant column major format and can thus be passed directly to (optimized) Fortran libraries.

Remark

For a matrix $A \in \mathbb{R}^{m \times n}$, the leading dimension have to fulfill

$$LD \geqslant \max(1, m) \, .$$

Basic Object Oriented Design

Matrix Storage Formats

Basic Object Oriented Design

A matrix and its meta information can be represented by a **struct** in C:

```
struct my_matrix_st{
        INT cols;
        INT rows;
        INT LD;
        double *values;
        char structure;
};
```

- INT can either be int or long depending on the application and, if Fortran libraries are used, the default integer size in Fortran.
- The default behavior would be 32-bit (4-byte) integers:

```
#define INT int
```

- If 64-bit integers are used, one defines:

```
#define INT long
```

Matrix Storage Formats

Basic Object Oriented Design

Remark

If for the matrix element $A(m, n)$ the index mapping fulfills

$$\mathrm{IDX2F}(m, n, ld) > 2^{31} - 1,$$

the 32-bit integers could no longer be used to call (external) Fortran subroutines. This causes in **integer overflow** during the computation of the mapping.
$\rightarrow$ 64-bit integers must be used.

Matrix Storage Formats

Remark

If for the matrix element $A(m, n)$ the index mapping fulfills

$$\mathrm{IDX2F}(m, n, ld) > 2^{31} - 1,$$

the 32-bit integers could no longer be used to call (external) Fortran subroutines. This causes in **integer overflow** during the computation of the mapping.
$\rightarrow$ 64-bit integers must be used.

Example 6.38

If we assume $A \in \mathbb{R}^{m \times m}$, we have to use 64-bit integers if $m \geqslant 43\,340$:

- **`double`**: matrix larger than 16 GiB
- **`float`**: matrix larger than 8 GiB
- **`double complex`**: matrix larger than 32 GiB
- **`float complex`**: matrix larger than 16 GiB

Matrix Storage Formats

Basic Object Oriented Design

Our example matrix A now leads to:

- **`A.cols`**$= 4$,
- **`A.rows`**$= 4$,
- **`A.LD`**$= 4$
- **`A.values`**$=$ $\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{2}\boxed{3}\boxed{5}\boxed{0}\boxed{0}\boxed{4}\boxed{0}\boxed{7}\boxed{0}\boxed{0}\boxed{6}\boxed{0}$

$$A = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 5 & 0 & 6 \\ 0 & 0 & 7 & 0 \end{bmatrix}$$

Matrix Storage Formats

Basic Object Oriented Design

Our example matrix A now leads to:

$$A = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 5 & 0 & 6 \\ 0 & 0 & 7 & 0 \end{bmatrix}$$

- **`A.cols`**$= 4$,
- **`A.rows`**$= 4$,
- **`A.LD`**$= 4$
- **`A.values`**$=$

1	0	0	0	2	3	5	0	0	4	0	7	0	0	6	0

Now, we want to access the 2×2 submatrix of A beginning at the $(2, 2)$ position of A:

$$B = \begin{bmatrix} 3 & 4 \\ 5 & 0 \end{bmatrix}$$

This gives

- **`B.cols`**$= 2$,
- **`B.rows`**$= 2$,
- **`B.LD`**$= 4$
- **`B.values = &A.values[5]`**

Tiled Matrix Storage

Matrix Storage Formats

Tiled Matrix Storage

- column-major storage is well understood and works well in most sequential algorithms,
- memory access issues, when dealing with submatrices
- bad for "tall-and-skinny" matrices (rows $\gg$ columns)

Design goals:

- reduce the leading dimension by organizing the matrix data in smaller, contiguous **tiles** rather than storing it in a linear column-major order.
- each **tile** is stored in column-major again
- title stored in a block column-major storage again.
- data structures have to keep track of
  - the size of the tiles,
  - the number of tiles in each dimension,
  - the leading dimension in each tile,
  - information to store the tiles

Matrix Storage Formats

Tiled Matrix Storage

By using 2×2 tiles our example A gets partitioned into:

$$\mathbf{\texttt{A.tiles}} = \begin{array}{|c|c|c|c|}\hline \begin{matrix}1 & 2\\ 0 & 3\end{matrix} & \begin{matrix}0 & 5\\ 0 & 0\end{matrix} & \begin{matrix}0 & 0\\ 4 & 0\end{matrix} & \begin{matrix}0 & 6\\ 7 & 0\end{matrix} \\\hline\end{array}$$

This leads to the following layout in memory:

$$\mathbf{\texttt{A.values}} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1 & 0 & 2 & 3 & 0 & 0 & 5 & 0 & 0 & 4 & 0 & 0 & 0 & 6 & 7 & 0 \\\hline\end{array}$$

Matrix Storage Formats

Tiled Matrix Storage

By using 2×2 tiles our example A gets partitioned into:

$$\texttt{A.tiles} = \boxed{\begin{matrix}1 & 2\\ 0 & 3\end{matrix}}\boxed{\begin{matrix}0 & 5\\ 0 & 0\end{matrix}}\boxed{\begin{matrix}0 & 0\\ 4 & 0\end{matrix}}\boxed{\begin{matrix}0 & 6\\ 7 & 0\end{matrix}}$$

This leads to the following layout in memory:

$$\texttt{A.values} = \boxed{1}\boxed{0}\boxed{2}\boxed{3}\boxed{0}\boxed{0}\boxed{5}\boxed{0}\boxed{0}\boxed{4}\boxed{0}\boxed{0}\boxed{0}\boxed{6}\boxed{7}\boxed{0}$$

Remark

There are two major applications for the tile storage:

- parallel multi-CPU aware algorithms ↗ see summer term,
- internal/intermediate representation for high performance computational kernels in order to optimize the memory access scheme.

Sparse Matrices

Matrix Storage Formats

Example 6.39

We want to solve

$$-\frac{\Delta^2}{\Delta x^2} u = f \text{ with } u, f : [0,1] \to \mathbb{R} \text{ and } u(0) = g(0),\ u(1) = g(1)$$

and discretize $[0,1]$ in steps of size $h = \frac{1-0}{n-1}$. Then we have $x_i = i \cdot h$, $i = 0, \dots, n$ and $u_i = u(x_i)$. Furthermore, the second derivative of u in x_i can be approximated by

$$\frac{\Delta^2}{\Delta x^2} u_i \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}$$

This yields:

$$\begin{aligned} u(x_0) = &\qquad u_0 = g(0) \\ & -\frac{\Delta^2}{\Delta x^2} u_i \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} = f(x_i) \\ u(x_n) = &\qquad u_n = g(1) \end{aligned}$$

Matrix Storage Formats

Sparse Matrices

Example 6.39

With $n = 8$ we obtain the linear systems this gives

$$\frac{1}{h^2}\begin{bmatrix} 2 & & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & h^2 \end{bmatrix}\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} g(0) \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ g(1) \end{bmatrix}$$

$\rightarrow$ at most 3 non-zero entries per row, independent from n!

Matrix Storage Formats

Sparse Matrices

Definition 6.40

We call a matrix $A \in \mathbb{R}^{n\times n}$ or $A \in \mathbb{C}^{n\times n}$ **sparse** if only a few entries of A per row or column are non-zero, in average.

Precisely, we want A to be such that storing A uses $\mathcal{O}(n)$ storage and multiplication with A is performed in $\mathcal{O}(n)$ effort.

Alternatively: It is worth to keep track of the non-zero entries.

Matrix Storage Formats

Sparse Matrices

Definition 6.40

We call a matrix $A \in \mathbb{R}^{n \times n}$ or $A \in \mathbb{C}^{n \times n}$ **sparse** if only a few entries of A per row or column are non-zero, in average.

Precisely, we want A to be such that storing A uses $\mathcal{O}(n)$ storage and multiplication with A is performed in $\mathcal{O}(n)$ effort.

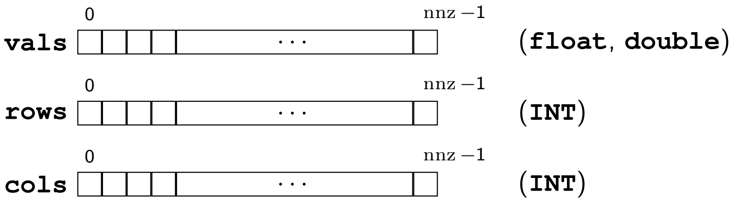
Alternatively: It is worth to keep track of the non-zero entries.

Basic idea: We store “only” the non-zero entries and neglect the zeros.

Matrix Storage Formats

Sparse Matrices – Coordinate Storage (COO)

Stores A in 3 vectors of length $\mathrm{nnz}(A)$ for entry values, row indices, and column indices:



Advantages:

- easy to implement
- easy addition of new entries
- easy elementwise access

Matrix Storage Formats

Sparse Matrices – Coordinate Storage (COO)

Drawbacks:

- non local memory access
- (atomic access to output vector in threaded implementation)

[1]`https://math.nist.gov/MatrixMarket/`

Matrix Storage Formats

Sparse Matrices – Coordinate Storage (COO)

Drawbacks:

- non local memory access
- (atomic access to output vector in threaded implementation)

Note that the format does not prescribe any ordering of the entries, i.e., the storage for the matrix A might look like (using C indexing to avoid shifts)

`vals`	1	7	2	3	4	5	6
`rows`	0	3	0	1	1	2	2
`cols`	0	2	1	1	2	1	3

[1]`https://math.nist.gov/MatrixMarket/`

Matrix Storage Formats

Sparse Matrices – Coordinate Storage (COO)

Drawbacks:

- non local memory access
- (atomic access to output vector in threaded implementation)

Note that the format does not prescribe any ordering of the entries, i.e., the storage for the matrix A might look like (using C indexing to avoid shifts)

`vals`	1	7	2	3	4	5	6
`rows`	0	3	0	1	1	2	2
`cols`	0	2	1	1	2	1	3

Remark

The coordinate storage format is, for example, the basis of the sparse matrix version of the Matrix Market[1] file exchange format.

[1] `https://math.nist.gov/MatrixMarket/`

Matrix Storage Formats

Sparse Matrices – Compressed Sparse Row Storage (CSR/CRS)

We use three vectors to store the data:

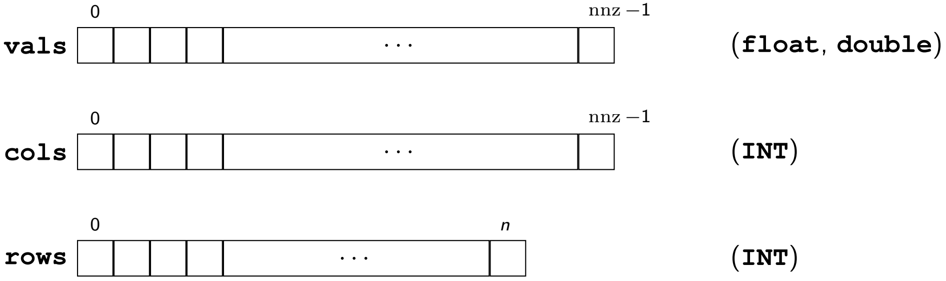
- two vectors **`vals`** and **`cols`** store the entry values and its column indices.
- a third vector holding the row indices (**`rows`**) stores, where the corresponding row starts in the vectors **`vals`** and **`cols`**,
- the last entry of **`rows`** stores the number of non-zero entries $\mathrm{nnz}(A)$,
- the first entry in **`rows`** is not needed, but stored to simplify implementations.

Matrix Storage Formats

Sparse Matrices – Compressed Sparse Row Storage (CSR/CRS)

We use three vectors to store the data:

- two vectors **`vals`** and **`cols`** store the entry values and its column indices.
- a third vector holding the row indices (**`rows`**) stores, where the corresponding row starts in the vectors **`vals`** and **`cols`**,
- the last entry of **`rows`** stores the number of non-zero entries $\mathrm{nnz}(A)$,
- the first entry in **`rows`** is not needed, but stored to simplify implementations.



Matrix Storage Formats

Sparse Matrices – Compressed Sparse Row Storage (CSR/CRS)

Advantages:

- optimal storage requirements
- can exploit BLAS (see later) in per row operations
- allows multithreading/parallelization

Matrix Storage Formats

Sparse Matrices – Compressed Sparse Row Storage (CSR/CRS)

Advantages:

- optimal storage requirements
- can exploit BLAS (see later) in per row operations
- allows multithreading/parallelization

Drawbacks:

- non local memory access due to indirect indexing
- (load balancing problem in threaded implementations due to different row lengths)

Matrix Storage Formats

Sparse Matrices – Compressed Sparse Row Storage (CSR/CRS)

Advantages:

- optimal storage requirements
- can exploit BLAS (see later) in per row operations
- allows multithreading/parallelization

Drawbacks:

- non local memory access due to indirect indexing
- (load balancing problem in threaded implementations due to different row lengths)

Remark

An equivalent format swapping the roles of row and column pointers in the above, is used, e.g., in MATLAB. It is called **compressed sparse column storage** (CSC/CCS).

Matrix Storage Formats

Sparse Matrices – Compressed Sparse Row Storage (CSR/CRS)

Remark

A row in the CSR storage can be accessed via

```
for ( j = rowptr [i] ; j < rowptr[i+1]; j++) {...}
```

By storing the redundant information in the first and the last entry of **`rowptr`**, the first and the last row does not need a special treatment.

Matrix Storage Formats

Sparse Matrices – Compressed Sparse Row Storage (CSR/CRS)

Remark

A row in the CSR storage can be accessed via

```
for ( j = rowptr [i] ; j < rowptr[i+1]; j++) {...}
```

By storing the redundant information in the first and the last entry of **`rowptr`**, the first and the last row does not need a special treatment.

Using the CSR storage, our example matrix A looks like:

`vals`	1	2	3	4	5	6	7
`cols`	0	1	1	2	1	3	2
`rows`	0	2	4	6	7		

Matrix Storage Formats

Sparse Matrices – Ellpack and Ellpack-R (ELLR)

COO and CSR are easy to understand and easy to implement but:

- not well suited for CPUs with vector registers
- no load balancing
- memory access does not fit well with cache-line and page access

Matrix Storage Formats

Sparse Matrices – Ellpack and Ellpack-R (ELLR)

COO and CSR are easy to understand and easy to implement but:

- not well suited for CPUs with vector registers
- no load balancing
- memory access does not fit well with cache-line and page access

We assume the n_r be the maximum number of non-zeros in all rows, then **Ellpack** uses two arrays of size $n \times n_r$:

- the **`vals`** array, where each row contains the non-zero values of the corresponding row in the represented matrix,
- and the **`cols`** array, which contains the corresponding column indices,
- eventually existing gaps will be filled with zeros.

Matrix Storage Formats

Sparse Matrices – Ellpack and Ellpack-R (ELLR)

COO and CSR are easy to understand and easy to implement but:

- not well suited for CPUs with vector registers
- no load balancing
- memory access does not fit well with cache-line and page access

We assume the n_r be the maximum number of non-zeros in all rows, then **Ellpack** uses two arrays of size $n \times n_r$:

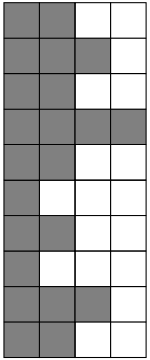
- the **`vals`** array, where each row contains the non-zero values of the corresponding row in the represented matrix,
- and the **`cols`** array, which contains the corresponding column indices,
- eventually existing gaps will be filled with zeros.

The Ellpack-R format adds a third vector, containing the actual length (non-zero elements) of each row to avoid the processing of zero elements.

Matrix Storage Formats

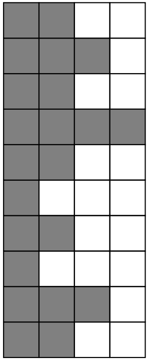
Sparse Matrices – Ellpack and Ellpack-R (ELLR)

vals $(n \times n_r)$



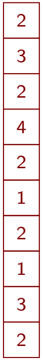
(**float**, **double**)

cols $(n \times n_r)$



(**INT**)

r (n)



(**INT**)

Matrix Storage Formats

Sparse Matrices – Ellpack and Ellpack-R (ELLR)

Advantages:

- constant per row length $\rightsquigarrow$ good load balancing properties
- padding could be integated, e.g. n_r must be a multiple of the cache-line length
- coalesced memory access (threads k, k+1 access consecutive memory cells)
- no synchronization required

Matrix Storage Formats

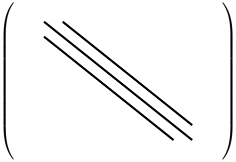
Sparse Matrices – Ellpack and Ellpack-R (ELLR)

Advantages:

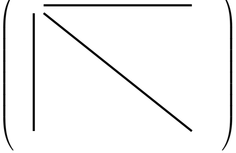
- constant per row length $\rightsquigarrow$ good load balancing properties
- padding could be integated, e.g. n_r must be a multiple of the cache-line length
- coalesced memory access (threads k, k+1 access consecutive memory cells)
- no synchronization required

Drawbacks:

- The storage requirement is dominated by the longest row. $\Rightarrow$ Possibly, many zeros are stored.



good



bad

- The zeros are actually processed without leading to new information.

Matrix Storage Formats

Sparse Matrices – Ellpack and Ellpack-R (ELLR)

Advantage of the ELLR:

- The unnecessary processing of zeros is avoided.

Drawback of the ELLR:

- Additional n integers for storing of the row lengths are required.
- Load balancing features of the Ellpack format are no longer valid.

Matrix Storage Formats

Sparse Matrices – Ellpack and Ellpack-R (ELLR)

Advantage of the ELLR:

- The unnecessary processing of zeros is avoided.

Drawback of the ELLR:

- Additional n integers for storing of the row lengths are required.
- Load balancing features of the Ellpack format are no longer valid.

Our example matrix A turns into:

`vals` (4×2)

1	2
3	4
5	6
7	0

`cols` (4×2)

0	1
1	2
1	3
2	0

`r` (4)

2
2
2
1

Matrix Storage Formats

Sparse Matrices – Ellpack and Ellpack-R (ELLR)

Remark

In the NVIDIA® CUDA® toolkit for acceleration of codes using NVIDIA® graphics adapters, or more precisely in the corresponding cusparse library used for working with sparse matrices, a hybrid matrix storage format is used. This format uses Ellpack for rows up to a length $n_r^* = \min\{tol, n_r\}$ and the remaining entries are stored in CSR or another sparse matrix storage scheme. This reduces the problem of exceptionally long rows in Ellpack and Ellpack-R.

Matrix Storage Formats

Sparse Matrices – Ellpack and Ellpack-R (ELLR)

Remark

In the NVIDIA® CUDA® toolkit for acceleration of codes using NVIDIA® graphics adapters, or more precisely in the corresponding cusparse library used for working with sparse matrices, a hybrid matrix storage format is used. This format uses Ellpack for rows up to a length $n_r^* = \min\{tol, n_r\}$ and the remaining entries are stored in CSR or another sparse matrix storage scheme. This reduces the problem of exceptionally long rows in Ellpack and Ellpack-R.

Remark

There are other storage schemes available:

- DAS/JDS/ITPACK (Diagonally Addresses Storage), column indicies are given w.r.t. to the diagonal
- BCSR (Block Compressed Sparse Rowsppkill soffice), essentially CSR but with blocks
- Skyline, similar to Ellpack, but focus on band matrices
- RSB (Recursive Sparse Blocks)
- ...

Complex Matrices

Matrix Storage Formats

Complex Matrices

The previous storage schemes cover only real matrices. In case of a complex valued data, two different approaches are used:

- a second array **`ivals`**, storing the imaginary parts, is added
  - used by pre 2018b versions of MATLAB
  - operations only need to implemented once and then used on **`vals`** and **`ivals`**
  - everything works real valued, but algorithms need to handle the real and imaginary parts properly

Matrix Storage Formats

Complex Matrices

The previous storage schemes cover only real matrices. In case of a complex valued data, two different approaches are used:

- a second array **`ivals`**, storing the imaginary parts, is added
  - used by pre 2018b versions of MATLAB
  - operations only need to implemented once and then used on **`vals`** and **`ivals`**
  - everything works real valued, but algorithms need to handle the real and imaginary parts properly
- each entry in **`vals`** consists of a structure containing the real and the imaginary part of the entry
  - all operations need to be implemented for complex values as well
  - better memory locality

Linear Algebra Software

BLAS and LAPACK

Linear Algebra Software

BLAS and LAPACK

The BLAS (Basic Linear Algebra Subroutines) and LAPACK (Linear Algebra Package) are the most commonly used software packages in Scientific Computing.

- development started in mid of the 1970s
- Fortran 77/90 code, now slowly transforming to newer standards
- LAPACK generalized the LINPACK and EISPACK package
- define a standard API and a reference implementation.
- one of inventors got the Turing Award (Jack Dongarra, 2021)
- Fortran and C interfaces defined

Linear Algebra Software

BLAS and LAPACK

The BLAS (Basic Linear Algebra Subroutines) and LAPACK (Linear Algebra Package) are the most commonly used software packages in Scientific Computing.

- development started in mid of the 1970s
- Fortran 77/90 code, now slowly transforming to newer standards
- LAPACK generalized the LINPACK and EISPACK package
- define a standard API and a reference implementation.
- one of inventors got the Turing Award (Jack Dongarra, 2021)
- Fortran and C interfaces defined

Beside the reference implementation, there tuned versions available:

- Intel® oneAPI Math Kernel Libary (MKL, oneMKL)
- IBM Engineering and Scientific Subroutines Library (ESSL)
- OpenBLAS
- BLIS, AMD BLIS
- ...

BLAS

Linear Algebra Software

BLAS

The basic linear algebra subroutines BLAS are sub-divided into three classes, called levels, that are mainly standing for the involved memory and computation complexities, but also for their historic development.

- Level 1 (1979): $\mathcal{O}(n)$ operations on $\mathcal{O}(n)$ data
- Level 2 (1988): $\mathcal{O}(n^2)$ operations on $\mathcal{O}(n^2)$ data
- Level 3 (1990): $\mathcal{O}(n^3)$ operations on $\mathcal{O}(n^2)$ data

Linear Algebra Software

BLAS

The basic linear algebra subroutines BLAS are sub-divided into three classes, called levels, that are mainly standing for the involved memory and computation complexities, but also for their historic development.

- Level 1 (1979): $\mathcal{O}(n)$ operations on $\mathcal{O}(n)$ data
- Level 2 (1988): $\mathcal{O}(n^2)$ operations on $\mathcal{O}(n^2)$ data
- Level 3 (1990): $\mathcal{O}(n^3)$ operations on $\mathcal{O}(n^2)$ data

BLAS has a Fortran induced naming scheme (Level 1):

$$\underbrace{\texttt{cblas\_}}_{\text{prefix}} \quad \underset{\text{datatype}}{\text{X}} \quad \underset{\text{operation}}{\text{XXXX}}$$

→ The prefix is required to avoid doubling function names in C and in Fortran:

- Calling from Fortran/via the Fortran interface: **no prefix**
- Calling via the C interface: **cblas_**

Linear Algebra Software

BLAS

Data types (allowed specifiers)

- **`s`** — single precision real
- **`c`** — single precision complex
- **`d`** — double precision real
- **`z`** — double precision complex

Linear Algebra Software

BLAS

Data types (allowed specifiers)

- **s** — single precision real
- **c** — single precision complex
- **d** — double precision real
- **z** — double precision complex

Operations (examples)

axpy	$y \leftarrow \alpha x + y$
dot	$r \leftarrow x^{\mathsf{T}} y$
nrm2	$r \leftarrow \|x\|_2 = \sqrt{x^{\mathsf{T}} x}$
asum	$r \leftarrow \|x\|_1 = \sum_i \lvert x_i \rvert$

Linear Algebra Software

BLAS

The levels 2 and 3 additionally respect/exploit matrix structures and indicate them in the corresponding function names:

$$\underbrace{\texttt{cblas\_}}_{\text{prefix}} \quad \underbrace{\text{X}}_{\text{datatype}} \quad \underbrace{\text{XX}}_{\text{structure}} \quad \underbrace{\text{XXX}}_{\text{operations}}$$

Linear Algebra Software

BLAS

The levels 2 and 3 additionally respect/exploit matrix structures and indicate them in the corresponding function names:

$$\underbrace{\texttt{cblas\_}}_{\text{prefix}} \quad \underset{\text{datatype}}{\text{X}} \quad \underset{\text{structure}}{\text{XX}} \quad \underset{\text{operations}}{\text{XXX}}$$

Structure Placeholders

GE	general	**GB**	general banded		
SY	symmetric	**SB**	symmetric banded	**SP**	symmetric packed
HE	hermitian	**HB**	hermitian banded	**HP**	hermitian packed
TR	triangular	**TB**	triangular banded	**TP**	triangular packed

The structure placeholder mostly specifies properties of the input data.

Linear Algebra Software

BLAS

Typical arguments:

UPLO For triangular matrix operations the type of triangular structure is controlled by the argument **`UPLO`**. It is taking character values '**`L`**', '**`U`**' for lower or upper triangular, respectively.

SIDE The operand order (e.g., decision about left or right multiplication) is steered by the **`SIDE`** arguments '**`L`**' or '**`R`**'.

DIAG For triangular matrices the **`DIAG`** argument specifies whether they have a unit diagonal '**`U`**' or not '**`N`**'.

TRANS Transposition is decided via **`TRANS`** argument, it takes one of the following values:

- '**`N`**' – non transposed – X
- '**`T`**' – transposed – X^{T}
- '**`C`**' – conjugate transposed – X^{H}

Linear Algebra Software

BLAS

Example 6.41

We take a look in the double precision and double precision complex matrix-matrix-product routines that perform the operation

$$C \leftarrow \alpha \cdot \mathrm{op}(A) \cdot \mathrm{op}(B) + \beta C,$$

where $\mathrm{op}(.)$ refers to the transposition types above.

The Fortran interfaces and data types for the real case are

```
SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
  !.. Scalar Arguments ..
  REAL*8 ALPHA,BETA
  INTEGER K,LDA,LDB,LDC,M,N
  CHARACTER TRANSA,TRANSB

  !.. Array Arguments ..
  REAL*8 A(LDA,*),B(LDB,*),C(LDC,*)
```

Linear Algebra Software

BLAS

Example 6.41

Linear Algebra Software

BLAS

Considering the complex case, we have

```
SUBROUTINE ZGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
  !.. Scalar Arguments ..
  COMPLEX*16 ALPHA,BETA
  INTEGER K,LDA,LDB,LDC,M,N
  CHARACTER TRANSA,TRANSB

  !.. Array Arguments ..
  COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
```

Thus, the corresponding C prototypes look like

```
void dgemm_(char *transa, char *transb, int *m, int *n, int *k,
            double *alpha, double *A, int *lda,
            double *B, int *ldb,
            double *beta, double *C, int *ldc,
            size_t len_transa, size_t len_transb);
```

for the real and

Linear Algebra Software

BLAS

Example 6.41

```
void zgemm_(char *transa, char *transb, int *m, int *n, int *k,
            double complex *alpha, double complex *A, int *lda,
            double complex *B, int *ldb,
            double complex *beta, double complex *C, int *ldc,
            size_t len_transa, size_t len_transb);
```

for the complex case.

Linear Algebra Software

BLAS

Vector Operations (BLAS Level 1)

- scaling and addition: αx, $\alpha x + y$,
- inner products: $x^* y$,
- norm expressions: $\|x\|_2$, $\|x\|_1$,$\|x\|_\infty$.

Linear Algebra Software

BLAS

Vector Operations (BLAS Level 1)

- scaling and addition: αx, $\alpha x + y$,
- inner products: $x^* y$,
- norm expressions: $\|x\|_2$, $\|x\|_1$,$\|x\|_\infty$.

Matrix-Vector Operations (BLAS Level 2)

Let $\mathbb{F} \in \{\mathbb{C}, \mathbb{R}\}$, $\alpha, \beta \in \mathbb{F}$, $A \in \mathbb{F}^{m \times n}$, $x, y \in \mathbb{F}^n$:

- scaling and addition: $\alpha Ax + \beta y$, $\alpha A^* x + \beta y$,
- rank-1/2 updates: $A + \alpha xy^*, A + \alpha xx^*$, $A + \alpha xy^* + \beta yx^*$,
- triangular solves: $\alpha T^{-1} x$, $\alpha T^{-*} x$, T triangular.

Linear Algebra Software

BLAS

Matrix-Matrix Operations (BLAS Level 3)

- $\alpha AB + \beta C$, $\alpha AB^* + \beta C$, $\alpha A^* B^* + \beta C$,
- rank k updates: $\alpha AA^* + \beta C$, $\alpha A^* A + \beta C$
- rank $2k$ updates: $\alpha A^* B + \alpha B^* A + \beta C$
- triangular multi-solves: $\alpha T^{-1} C$, $\alpha T^{-*} C$, T triangular.

The Idea Behind the Level-3 Performance Gain

Linear Algebra Software

The Idea Behind the Level-3 Performance Gain

Level-3 BLAS operations are necessary for high performance.

$\rightarrow$ With $\mathcal{O}(n^3)$ operations but only $\mathcal{O}(n^2)$ data, each value is used $\mathcal{O}(n)$ times.

Linear Algebra Software

The Idea Behind the Level-3 Performance Gain

Level-3 BLAS operations are necessary for high performance.

$\rightarrow$ With $\mathcal{O}(n^3)$ operations but only $\mathcal{O}(n^2)$ data, each value is used $\mathcal{O}(n)$ times.

Block sub-structuring increases the reuse of already fetched information.
In case of the **`GEMM`** operation, $C \leftarrow C + AB^\mathsf{T}$, one can evaluate the 2×2 block structure:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} + \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} \begin{bmatrix} B_{11}^\mathsf{T} & B_{21}^\mathsf{T} \end{bmatrix},$$

which allows to compute the single blocks in the result as:

$$\begin{aligned} C_{11} &\leftarrow C_{11} + A_{11} B_{11}^\mathsf{T}, & C_{12} &\leftarrow C_{12} + A_{11} B_{21}^\mathsf{T}, \\ C_{21} &\leftarrow C_{21} + A_{21} B_{11}^\mathsf{T}, & C_{22} &\leftarrow C_{22} + A_{21} B_{21}^\mathsf{T}. \end{aligned}$$

The actual number of blocks and their sizes depend strongly on the used CPU and its memory hierarchy. $\nearrow$ BLAS libraries like OpenBLAS, MKL, BLIS respect this.

LAPACK

Linear Algebra Software

LAPACK

LAPACK (Linear Algebra PACKage) is a Fortran 90 based library that provides routines for

- solution of linear systems of equations,
- least squares solutions of linear systems of equations,
- solutions of eigenvalue problems,
- and singular value problems.

Linear Algebra Software

LAPACK

LAPACK (Linear Algebra PACKage) is a Fortran 90 based library that provides routines for

- solution of linear systems of equations,
- least squares solutions of linear systems of equations,
- solutions of eigenvalue problems,
- and singular value problems.

Some facts:

- developed since 1992 as successor of LINPACK and EISPACK,
- current version 3.12.0 (Nov 24, 2023)
- Mostly packaged with BLAS
- Optimized routines in vendor libraries like OpenBLAS, BLIS, MKL, . . .

Linear Algebra Software

LAPACK routines are divided in 3 Categories

1. auxiliary routines
2. computational routines
3. driver routines

$\rightarrow$ The general naming scheme follows the BLAS Level-2/3 approach.

- auxiliary routines: these routines in LAPACK provide common helper functionality: scaling, reordering, machine specifications. Examples are:
    - **`disnan`**, **`sisnan`** — check the argument for NaN
    - **`dlamch`**, **`slamch`** — retrieve machine parameters, i.e., get $\mathbb{M}$, **`eps`**, base, length of mantissa, $e_{\min}$, $e_{\max}$
    - **`xerbla`** — error handling in case of invalid inputs

Linear Algebra Software

LAPACK

- computational routines: perform simple specific tasks
  - factorizations: LU, LL^*, LDL^*, QR, $LQ, \ldots$
  - eigenvalue and singular value computations
  - recovery of eigenvectors, Schur vector
- driver routines: these routines call a set of computational routines to solve linear algebra problems
  - linear equations: $Ax = b$
  - linear least squares: $\min_x \|b - Ax\|_2$
  - generalized linear least squares
  - eigenvalue decompositions
  - generalized eigenvalue/singular value decompositions

Linear Algebra Software

LAPACK

- computational routines: perform simple specific tasks
  - factorizations: LU, LL^*, LDL^*, QR, $LQ, \ldots$
  - eigenvalue and singular value computations
  - recovery of eigenvectors, Schur vector
- driver routines: these routines call a set of computational routines to solve linear algebra problems
  - linear equations: $Ax = b$
  - linear least squares: $\min_x \|b - Ax\|_2$
  - generalized linear least squares
  - eigenvalue decompositions
  - generalized eigenvalue/singular value decompositions

Related software:

- LAPACKE (C/C++ wrapper to LAPACK)
- ScaLAPACK (distributed parallel version)
- MAGMA (Matrix Algebra on GPU and Multicore Architectures)

SuiteSparse

Linear Algebra Software

SuiteSparse

SuiteSparse is a suite of sparse matrix algorithms and software libraries:

- **Sparse Matrix Factorization**: Provides algorithms for LU, Cholesky, and QR factorizations,
- **Solvers**: Includes direct solvers for linear systems,
- **Graph Algorithms**: Offers tools for graph partitioning and ordering, which are useful in optimizing sparse matrix operations,
- **Sparse Matrix Manipulation**: Supports operations such as matrix addition, multiplication, and transposition for sparse matrices,
- **High Performance**: Designed to take advantage of modern hardware architectures for efficient computation,
- **Parallel Computing**: Some components are optimized for parallel execution to improve performance on multi-core processors,
- **Numerical Stability**: Implements algorithms that maintain numerical stability and accuracy in computations involving sparse matrices.

See: `https://people.engr.tamu.edu/davis/suitesparse.html`

ITPACK

Linear Algebra Software

ITPACK

This package is intended for solving large sparse linear systems by iterative methods. It is hosted at `https://www.netlib.org/itpack`.

The main library consists of three sub-packages for

- single precision,
- double precision,
- vector machines.

It uses CG, PCG, Chebyschev acceleration and generalized CG for systems with non-symmetric matrices.

The development of this Fortran based package takes place at Center for Numerical Analysis at University of Texas at Austin.

Trilinos

Linear Algebra Software

Trilinos

"Trilinos is a collection of open source software libraries intended to be used as building blocks for the development of scientific applications".[2]
Trilinos is developed at the Sandia National Labs. The current version is 16.0.0 from Jul. 2024.
The package is licensed under the terms of the LGPL[3] and covers:

- construction and usage of sparse and dense matrices, graphs and vectors.
- iterative and direct solution of linear systems
- parallel multilevel and algebraic preconditioning
- and many more ...

The basic library is written in C++ with Fortran kernels. Moreover Python bindings are provided via SWIG. Trilinos can be found online at:
`https://trilinos.org`

[2] `https://en.wikipedia.org/wiki/Trilinos`
[3] see, e.g., `https://opensource.org/licenses/lgpl-license`

Native Packages for other Programming Environments and Languages

Linear Algebra Software

Native Packages for other Programming Environments and Languages

- C++
  - C++26 — Matrix manipulation library built-in (on top of BLAS)
  - boost — supports threading as well
    `https://www.boost.org/`
  - MTL — The Matrix Template Library
    `https://github.com/simunova/mtl4`
    - The library uses boost and BLAS in kernels.
    - A single computer version available as OpenSource.
    - MTL4 has distributed computing capabilities, but those are connected to a payed license release.
  - Eigen — "Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms." `https://eigen.tuxfamily.org/`
- Python
  - NumPy — provides proper n-d array for Python
    `https://www.numpy.org/`
  - SciPy — amongst many others provides LAPACK functionality (calling F90 LAPACK)
    `https://www.scipy.org/`

Linear Algebra Software

Native Packages for other Programming Environments and Languages

- Java
  - JaMa — Java Matrix Package provides basic linear algebra in Java `https://math.nist.gov/javanumerics/jama/`
  - JaMPack — same as JaMa
  - maintenance questionable: latest release Nov 2012, previous version July 2005.
- Julia — Matrix support is part of the language core, built on top of BLAS and LAPACK
- Rust
  - nalgebra – basic dense and sparse math library `https://nalgebra.org/` content