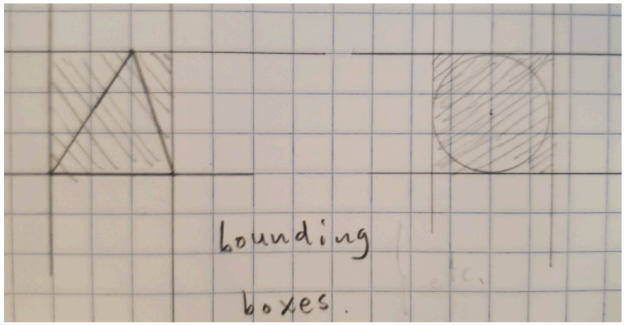
*lecture 11*

- *another inheritance example*
- *Kreyszig some concepts from functional analysis*
- *cancellation -numerical*
- *loop data dependencies*
- *machine independent code transformations*

*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*

```cpp
class Shape
{
public:
    virtual ~Shape() {} // virtual destructor

    virtual double area() const = 0;
    virtual double perimeter() const = 0;
    virtual void printInfo() const = 0;
    virtual std::pair<std::pair<double, double>, std::pair<double, double>> boundingBox() const = 0;
};
```




- a function declared as **virtual** in a base class
  - indicates to compiler the function can be overridden by derived classes
  - allows derived classes to provide their own implementation of the function while still being able to access it through a base class pointer or reference
  - polymorphism

*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*

```cpp
class Rectangle : public Shape
{
private:
    double length;
    double width;

public:
    Rectangle(double l, double w) : length(l), width(w) {}

    double area() const override
    {
        return length * width;
    }

    double perimeter() const override
    {
        return 2 * (length + width);
    }

    void printInfo() const override
    {
        std::cout << "Rectangle: Length = " << length << ", Width = " << width << std::endl;
    }

    std::pair<std::pair<double, double>, std::pair<double, double>> boundingBox() const override
    {
        return std::make_pair(std::make_pair(0.0, 0.0), std::make_pair(length, width));
    }
};
```

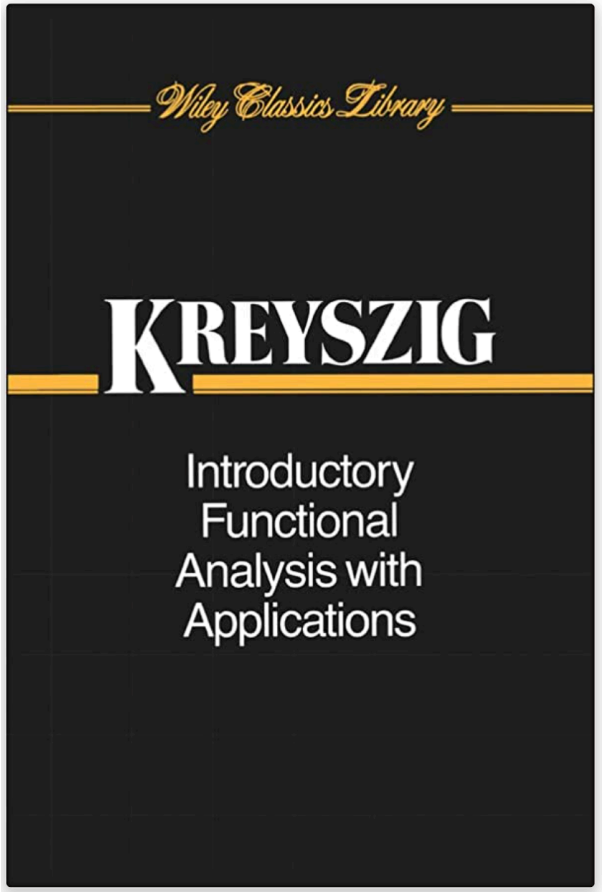
```cpp
class Triangle : public Shape
{
private:
    double side1, side2, side3;

public:
    Triangle(double s1, double s2, double s3) : side1(s1), side2(s2), side3(s3) {}

    double area() const override
    {
        // Using Heron's formula to calculate the area
        double s = (side1 + side2 + side3) / 2;
        return sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }

    double perimeter() const override
    {
        return side1 + side2 + side3;
    }

    void printInfo() const override
    {
        std::cout << "Triangle: side1 = " << side1 << ", side2 = " << side2 << ", side3 = " << side3 << std::endl;
    }

    std::pair<std::pair<double, double>, std::pair<double, double>> boundingBox() const override
    {
        return std::make_pair(std::make_pair(0.0, 0.0), std::make_pair(side1, side2));
    }
};
```

```cpp
class Circle : public Shape
{
private:
    double radius;

public:
    Circle(double r) : radius(r) {}

    double area() const override
    {
        return M_PI * radius * radius;
    }

    double perimeter() const override
    {
        return 2 * M_PI * radius;
    }

    void printInfo() const override
    {
        std::cout << "Circle: Radius = " << radius << std::endl;
    }

    std::pair<std::pair<double, double>, std::pair<double, double>> boundingBox() const override
    {
        return std::make_pair(std::make_pair(-radius, -radius), std::make_pair(radius, radius));
    }
};
```

```cpp
int main()
{
    std::vector<Shape *> shapes;
    shapes.push_back(new Rectangle(5, 3));
    shapes.push_back(new Circle(4));
    shapes.push_back(new Triangle(3, 4, 5));

    for (auto shape : shapes)
    {
        shape->printInfo();
        std::cout << "Area: " << shape->area() << std::endl;
        std::cout << "Perimeter: " << shape->perimeter() << std::endl;
        auto bb = shape->boundingBox();
        std::cout << "Bounding Box: (" << bb.first.first << ", " << bb.first.second << "), \
        (" << bb.second.first << ", " << bb.second.second << ")" << std::endl;
        std::cout << std::endl;
    }

    for (auto shape : shapes)
    {
        delete shape;
    }

    return 0;
}
```

```
//g++ -std=c++14 shapes.cpp -o xshapes
//./xshapes
// Rectangle: Length = 5, Width = 3
// Area: 15
// Perimeter: 16
// Bounding Box: (0, 0),         (5, 3)

// Circle: Radius = 4
// Area: 50.2655
// Perimeter: 25.1327
// Bounding Box: (-4, -4),         (4, 4)

// Triangle: side1 = 3, side2 = 4, side3 = 5
// Area: 6
// Perimeter: 12
// Bounding Box: (0, 0),         (3, 4)
```

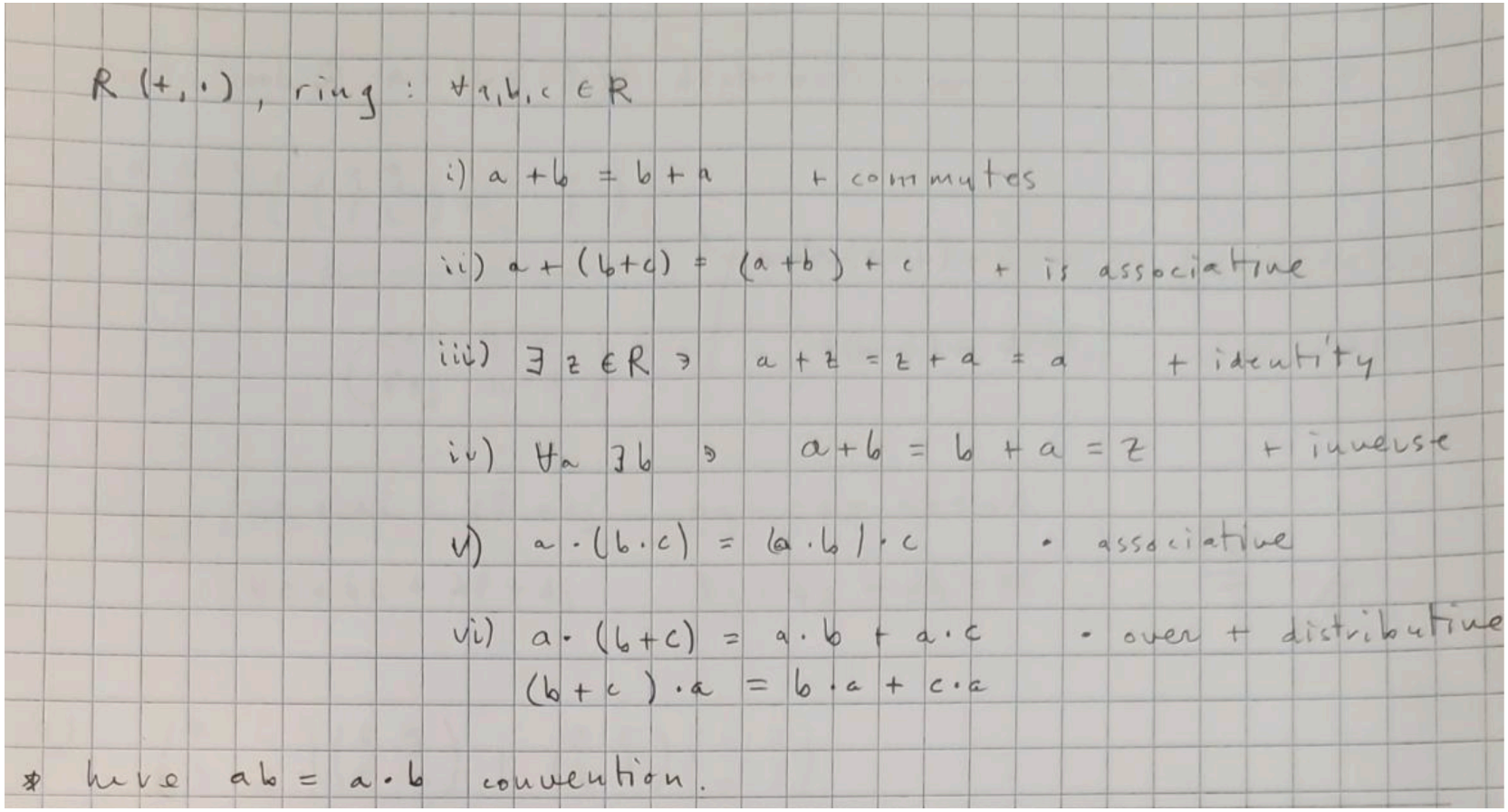
- *some math notes*
  - *vector, inner product, hilbert spaces*






Erwin Kreyszig. *Introductory functional analysis with applications*, volume 17. John Wiley & Sons, 1991.

*ring*




*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*

# *Metric spaces, Cauchy sequence, Completeness (notes1)*

A sequence $\{x_n\} \subset M$ is Cauchy if:

$$\forall \varepsilon > 0,\ \exists N \in \mathbb{N} \text{ such that } m, n > N \Rightarrow d(x_n, x_m) < \varepsilon$$

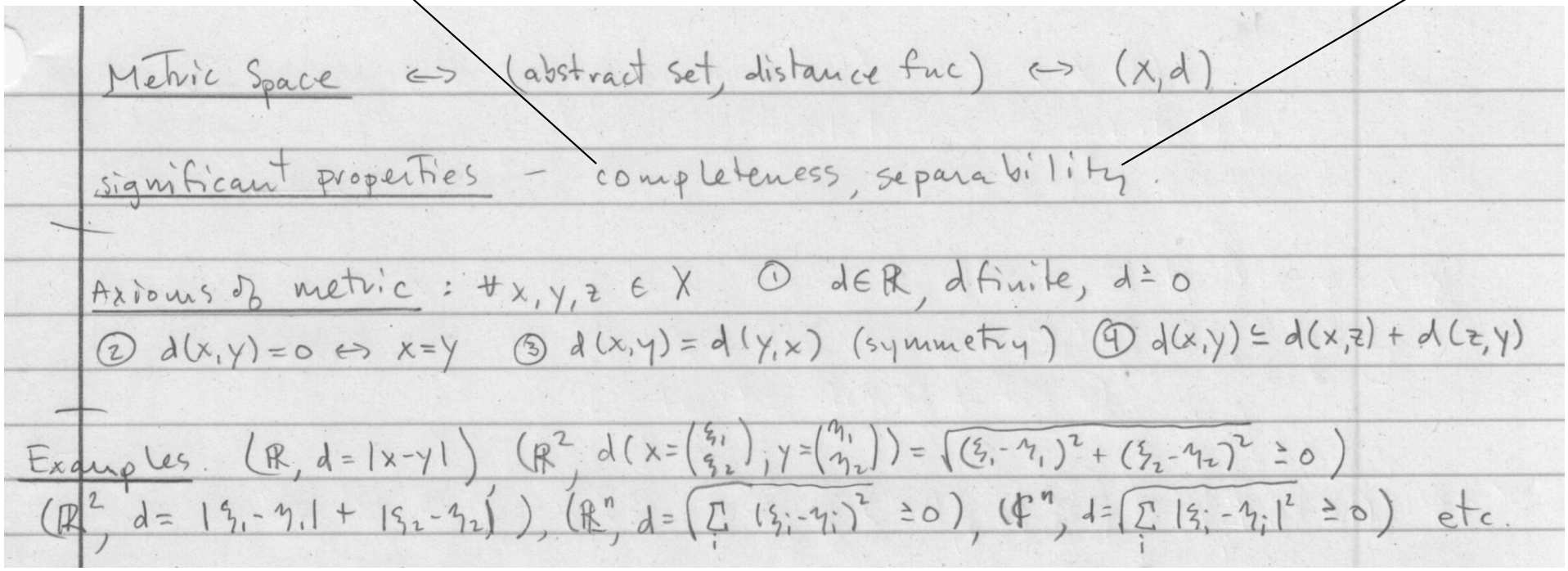
This means the points get arbitrarily close to each other as $n \to \infty$.

A metric space $(M, d)$ is **separable** if it contains a **countable dense subset**.

A subset $D \subset M$ is **dense** if:

$$\forall x \in M,\ \forall \varepsilon > 0,\ \exists d \in D \text{ such that } d(x, d) < \varepsilon$$

This means: Every point in the space can be approximated arbitrarily well by elements of $D$.





*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*

**Cauchy / Completeness.** $(x_n)_{n\geq 1}$ in $(X,d)$ Cauchy $\Rightarrow \forall \varepsilon > 0 \ \exists N_0(\varepsilon) \ni \forall m,n > N_0(\varepsilon)$ $d(x_m, x_n) < \varepsilon$. Space $X$ is complete when every Cauchy seq in space $X$ converges (has limit) to an element $x \in X$.

nb $\mathbb{R}, \mathbb{C}$ are complete // $\mathbb{Q}$ is incomplete; $(a,b)$ w/ metric induced from $\mathbb{R}$ incomplete

**Some Complete Spaces.** $\mathbb{R}^n$, $\mathbb{C}^n$, $l^\infty$, $c \equiv$ the space of all convergent sequences of complex #'s, $l^p$, $C[a,b]$

and **Incomplete Spaces:** $(\mathbb{Q}, d = |x-y|)$, polynomials on finite closed interval $\{ (P, d = \max_{t \in [a,b]} |x(t) - y(t)|)$.

and the space of continuous fncs on $[0,1]$ w/ $d = \int_0^1 |x(t) - y(t)| \, dt$.

**Completeness Proofs.** given set $X$, $X$ is made into $(X,d)$ and its completeness is checked: select $(x_n)_{n\geq 1} \in X$ arbitrary

show $x_n \to x$ w/ $x \in X$ in the sense of metric $d$

The $\ell^\infty$ space is the set of all **bounded sequences**:

$$x = (x_1, x_2, x_3, \dots)$$

such that the **supremum norm** is finite:

$$\|x\|_\infty = \sup_{n \geq 1} |x_n| < \infty$$

This means there's some upper bound on the absolute values of the sequence entries.

An $\ell^p$ **space** is the set of all **infinite sequences of real or complex numbers**:

$$x = (x_1, x_2, x_3, \dots)$$

such that the $p$**-norm** is finite:

$$\|x\|_p = \left( \sum_{n=1}^{\infty} |x_n|^p \right)^{1/p} < \infty$$

*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*

# *Vector space, Normed space, Banach space (complete normed space) (notes2)*

**norm**, the generalization of the length of a vector

**normed space**, vector space with a metric defined by a norm, $\|\cdot\|$

**Vector Space** A **vector space** over $\mathbb{F}$ is a non-empty set of elements (vectors) with vector addition and multiplication of vectors by scalars (elements of $\mathbb{F}$)

**addition**, Let $X$ a vector space. Let $x, y \in X$ $\quad x+y \in X$

commutes - $x+y = y+x$

associates - $x+(y+z) = (x+y)+z$ $\quad z \in X$ as well.

zero $\quad x+0 = x$

negative $\quad x+y = 0$, $\quad y = -x$

**multiplication**, $\beta, \alpha \in \mathbb{F}$, $x, y \in X$. Then $\alpha x \in X$.

$\alpha(\beta x) = (\alpha\beta)x$

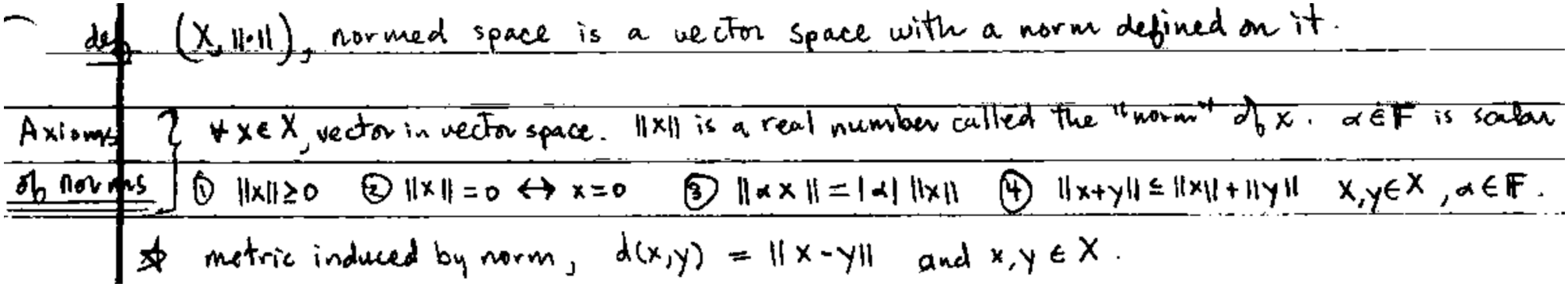
$1 \cdot x = x$

and $\alpha(x+y) = \alpha x + \alpha y$

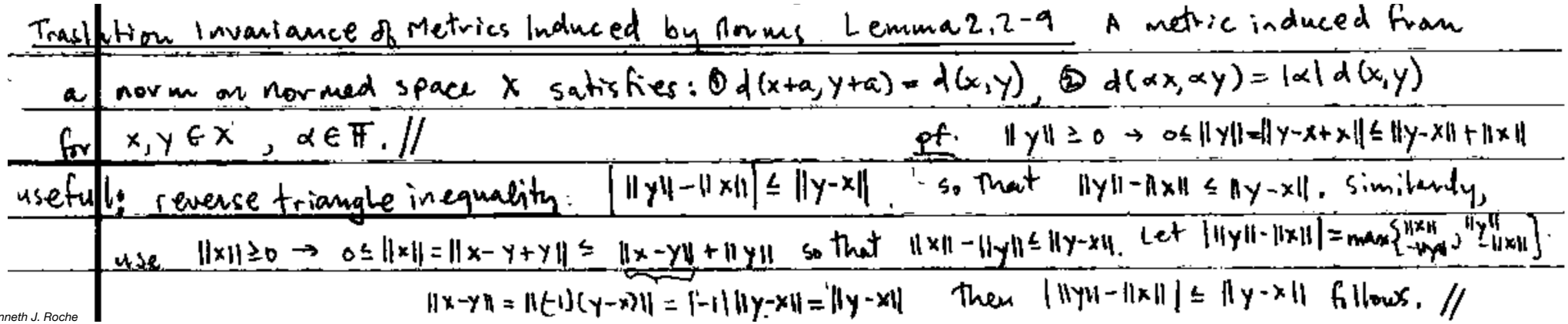
$(\alpha+\beta)x = \alpha x + \beta x$.

*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*

## *Normed space, Banach space (complete normed space) (notes2)*




## *metrics induced from norms (notes2)*




*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*

# *equivalent norms (notes2)*

**Equivalent norms** $\exists\, a, b \in \mathbb{R}$, $a, b > 0$ such that $\|\cdot\|$ and $\|\cdot\|_0$ are equivalent

$\leftrightarrow \quad a\|x\|_0 \le \|x\| \le b\|x\|_0$ for all $x \in X$.

**2.4-5 Equivalent Norms Thm.** In finite dimensional vector space $X$, any norm $\|\cdot\|$ is equivalent to any other norm $\|\cdot\|_0$. // **Pf** Choose $\{e_1, \dots, e_n\}$ basis for $X \to \forall x \in X$, $x = \alpha_1 e_1 + \cdots + \alpha_n e_n$.

So, $\|x\| \ge c\,(|\alpha_1| + \cdots + |\alpha_n|)$. Also, $\|x\|_0 \le \sum_{j=1}^{n} |\alpha_j| \|e_j\|_0 \le k \cdot \sum_{j=1}^{n} |\alpha_j|$ where $k = \max_j \|e_j\|_0$

If we want $a\|x\|_0 \le \|x\|$ then let $a = \frac{c}{k} > 0$. For $\|x\| \le b\|x\|_0$ use the same argument but reverse roles of $\|\cdot\|$, $\|\cdot\|_0$. ∎

* So, $a\|x\|_0 \le \|x\| \le b\|x\|_0 \quad \leftrightarrow \quad \frac{1}{b}\|x\| \le \|x\|_0 \le \frac{1}{a}\|x\| \quad \forall x \in X$, finite dimensional vector space.

recall Cauchy-Schwarz: $|\langle x, y\rangle| \le \|x\|_2 \cdot \|y\|_2$; $\|x\|_2 = \sqrt{\langle x, x\rangle}$ so $|\alpha_1| + |\alpha_2| = |\alpha_1| \cdot 1 + |\alpha_2| \cdot 1 \le \|\alpha_1 e_1 + \alpha_2 e_2\|_2 \cdot \sqrt{2}$ $\overset{n=2}{\Delta}$

*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

## *Inner product space, Hilbert space (notes3)*

**inner product space**, vector space $\mathbb{X}$ with inner product defined on $\mathbb{X}$ as a mapping from $\mathbb{X} \times \mathbb{X} \to \mathbb{K}$ (either $\mathbb{R}$ or $\mathbb{C}$) — we write $(x,y)$ w/ $x, y \in \mathbb{X}$

**Hilbert space**, inner product space complete in the metric defined by the inner product.

**Axioms for inner product.** vectors $x, y, z \in \mathbb{X}$, $\alpha, \beta \in \begin{pmatrix} \mathbb{R} \\ \mathbb{C} \end{pmatrix}$

① $(x+y, z) = (x,z) + (y,z)$ ② $(\alpha x, y) = \alpha (x,y)$ ③ $(x,y) = \overline{(y,x)}$ ④ $(x,x) \geq 0$ if $=0 \Rightarrow x = 0$.

**induced norm and metric:** $\|x\| = \sqrt{(x,x)}$, $d(x,y) = \|x-y\| = \sqrt{(x-y, x-y)}$.

**other basics:** $(x, \alpha y) = \bar{\alpha}(x,y)$ and $(\alpha x + \beta y, z) = \alpha(x,z) + \beta(y,z)$ and $(x, \alpha y + \beta z) = \bar{\alpha}(x,y) + \bar{\beta}(x,z)$.

## *parallelogram rule / polarization identities (notes3)*

norms induced by inner products satisfy the **parallelogram equality**: $\|x+y\|^2 + \|x-y\|^2 = 2\|x\|^2 + 2\|y\|^2$. $y \nearrow \to x$

**polarization identities** ($\mathbb{R}$) $(x,y) = \frac{1}{4}\left(\|x+y\|^2 - \|x-y\|^2\right)$ — used to recover the inner product from the given norm.

($\mathbb{C}$) $\mathrm{Re}\,(x,y) = \frac{1}{4}\left(\|x+y\|^2 - \|x-y\|^2\right)$, $\mathrm{Im}\,(x,y) = \frac{1}{4}\left(\|x+iy\|^2 - \|x-iy\|^2\right)$

$\mathbb{R}^n$ is Hilbert space: $(x,y) = \sum_{i=1}^n \xi_i \eta_i$ $x = (\xi_i)$, $y = (\eta_i)$. $\|x\| = \sqrt{(x,x)} = \sqrt{\sum_{i=1}^n \xi_i^2}$, $d(x,y) = \sqrt{\sum_{i=1}^n (\xi_i - \eta_i)^2}$

$\mathbb{C}^n$ is Hilbert space: $(x,y) = \sum_{i=1}^n \xi_i \bar{\eta}_i$, $\|x\| = \sqrt{\sum_{i=1}^n \xi_i \bar{\xi}_i} = \sqrt{\sum_{i=1}^n |\xi_i|^2}$, $d(x,y) = \|x-y\| = \sqrt{(x-y, x-y)} = \left[\sum_{i=1}^n |\xi_i - \eta_i|^2\right]^{1/2}$

# *polarization, Schwarz and triangle inequalities (notes3)*

* can use polarization identities (norms) to check the properties of the inner product, ie $(u,v) = \frac{1}{4}\left(\|u+v\|^2 - \|u-v\|^2\right)$.

**Schwarz Inequality:** $|(x,y)| \le \|x\| \cdot \|y\|$ pf: let $u = x - \alpha y$ w/ $\alpha = \frac{(x,y)}{(y,y)}$. $(u,u) = \|u\|^2 \ge 0$.

so, $(x-\alpha y, x-\alpha y) = (x,x) - (x,\alpha y) - (\alpha y, x) + |\alpha|^2 (y,y) = \|x\|^2 - \bar{\alpha}(x,y) - \alpha(y,x) + |\alpha|^2 \|y\|^2 \ge 0$

but $\bar{\alpha} = \frac{(y,x)}{(x,y)} = \frac{(y,x)}{\|y\|^2}$ and $\bar{\alpha}(x,y) = \frac{(y,x)\cdot(x,y)}{\|y\|^2} = \frac{|(x,y)|^2}{\|y\|^2}$. and $\alpha(y,x) = \frac{(x,y)}{\|y\|^2}(y,x) = \frac{|(x,y)|^2}{\|y\|^2}$

thus, $\|x\|^2 - \frac{|(x,y)|^2}{\|y\|^2} - \frac{\|(x,y)\|^2}{\|y\|^2} + \frac{|(x,y)|^2}{\|y\|^4}\cdot\|y\|^2 \ge 0 \rightarrow \|x\|^2 \ge \frac{|(x,y)|^2}{\|y\|^2} \rightarrow \|x\|^2\|y\|^2 \ge |(x,y)|^2$

or $|(x,y)| \le \|x\| \cdot \|y\|$ ∎

**Triangle Inequality:** $\|x+y\| \le \|x\| + \|y\|$. pf $(x+y, x+y) = \|x+y\|^2$

$= (x,x) + (x,y) + (y,x) + (y,y) = \|x\|^2 + \|y\|^2 + (x,y) + (y,x) \le \|x\|^2 + 2|(x,y)| + \|y\|^2 \le \|x\|^2 + 2\|x\|\|y\| + \|y\|^2$ (Schwarz)

so that $\|x+y\|^2 \le (\|x\| + \|y\|)^2$ or $\|x+y\| \le \|x\| + \|y\|$. ∎ note $(x,y)+(y,x) = (x,y) + \overline{(x,y)} = 2\mathrm{Re}(x,y)$. //

*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*

# cancellation (numerical issue)

```cpp
// g++ -std=c++14 cancellation.cpp -o xcancellation
// ./xcancellation
// a + b = 100000000
// a - b = 100000000
// Error: 2.98023223876953e-08
//

#include <iostream>
#include <cmath>
#include <iomanip>

int main()
{
    double a = 1.0e8;  // Large number
    double b = 1.0e-8; // Small number
    double c = a + b;
    double d = a - b;

    std::cout << "a + b = " << std::setprecision(15) << c << std::endl; // Expected: 100000000.00000001
    std::cout << "a - b = " << std::setprecision(15) << d << std::endl; // Expected: 99999999.99999999

    double error = std::abs(c - d);
    std::cout << "Error: " << std::setprecision(15) << error << std::endl;

    return 0;
}
```

$$\text{Error} = |(c-d)_{\text{FP64}} - 2b| = |2.9802322387695312 \times 10^{-8} - 2 \times 10^{-8}|$$

$$= 0.9802322387695312 \times 10^{-8} = 9.802322387695312 \times 10^{-9}$$

```
// Difference between the results
c = a + b = 100000000.00000001490116119385
d = a - b =  99999999.99999998509883880615
|c - d|   =           0.00000002980232238770
```

*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*

# loops and data dependencies - intro

## data dependencies

- *j* depends on *i* —> *i* produces result that *j* uses
- (chain) *j* depends on *k*, *k* depends on *i* —> *i* produces result that *j* uses

- 2 instructions that are *data dependent* cannot execute simultaneously

  - the compiler cannot schedule dependent instructions to completely overlap in the execution pipeline

    - if it tries, causes hazards and stalls in the pipeline

## data dependencies

- data *flows* between instructions either
  - *either* through registers
  - *or* memory locations

- detecting dependencies through registers is easier
  - i.e. register names are fixed and provide some insight

- detecting dependencies through memory locations is more difficult
  - i.e. 2 address instructions may refer to the same location in memory
    - *100(R4) and 20(R6)* … etc.

## data control dependencies

- determines ordering of instruction wrt a brach instruction so that non-branch instruction is only executed when it should be

```
// conditional branch
if (p1) { S1; }
if (p2) { S2; }
// S1 is control dependent on p1
// S2 is control dependent on p2 BUT not on p1
```

- the presence of conditional branches prevent scheduling overlapping instructions

## loop level //ism

- usually analyzed at the source code level

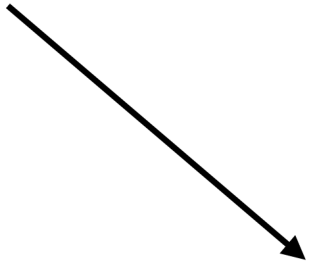
```
for (i=0;i<10;i++)
{
    x[i] = x[i] + s;
}
// dependence in loop body between 2 uses of x[i]
// BUT dependence is within a single iteration
// SO loop is parallel
```

# loop level //ism - loop carried dependence

```
for (i=0;i<10;i++)
{
    A[i+1] = A[i] + C[i]; // S1
    B[i+1] = B[i] + A[i+1]; // S2
}
// 2 data dependencies in loop body:
//   C1) i) S1 uses a value computed by S1 in an earlier iteration
//       ii) S2 uses a value computed by S2 in an earlier iteration
//
//   C2) S2 uses value computed by S1 in the same iteration

// C1) is example of a loop-carried dependence
// ... it occurs between different loop iterations

// C2) is actually resolvable to yield parallel loop
```

```
for (i=1;i<=100;i++)
{
    A[i] = A[i] + B[i]; // S1
    B[i+1] = C[i] + D[i]; // S2
}
// analysis
// S1 uses value assigned in previous iteration by S2
//       i.e. i=2, B[i+1=3] and i=3, B[i=3]
// this is a loop carried dependence
//
// S1 depends on S2, BUT S2 does not depend on S1
// Also, on iteration 1,
//  S1 uses value B[1] which must be computed prior to loop
```

# loop level //ism - fixable example



```
// removed loop carried dependence
A[1] = A[1] + B[1];
for (i=1;i<=99;i++)
{
    B[i+1] = C[i] + D[i];
    A[i+1] = A[i+1] + B[i+1];
}
B[101] = C[100] + D[100];
```

## more on code improving transformations

- Goal: machine independent transformations
  - machine dependent is deeper (register allocation, special instruction use, etc.)
    - keep most frequently used variables in registers
    - FMA (*fused multiply add* instruction) y = ax + b

- Rules on transformations
  - must preserve code meaning
  - should speed-up execution or require less resources
  - should be worth the effort!

- local transformations
  - performed only looking at statements in a basic block of code

## more on code improving transformations

- Examples of function preserving transformations
  - subexpression elimination
  - copy propagation
  - dead code removal
  - constant folding

- common subexpression E:
  - previously computed values in E have not changed since previous computation —> don't recompute

- constant folding
  - deducing an expression has constant value —> use the value instead of the expression

more on loops:

runtime often improved by decreasing the number of instructions in inner loop

- code motion: move code outside a loop
- induction variable elimination
- reduction in strength

```
// before
while (i <= limit - 2)
{
    // do stuff
}
// limit - 2 is invariant

//after
t = limit - 2;
while (i <= t)
{
    // do stuff
}
```

*Kenneth J. Roche*
*University of Washington*

# more on code improving transformations

```
// dead code elimination
debug = false;
// ...
if (debug) std::cout << "stuff";
// ...
if (debug) ...
// debug is dead, can safely remove all this code
```

```
//strength reduction
//  replace more expensive operations with cheaper ones
pow(x,2) --> x * x
2. * x   --> x + x
x / 2.   --> .5 * x
// etc ...
```

*Kenneth J. Roche*
*University of Washington*

# Constant folding examples

```cpp
#include <iostream>

int main() {
    int x = 3 * 4 + 5;  // This could be folded to 17 at compile time
    std::cout << "x = " << x << std::endl;
    return 0;
}
```

```cpp
int x = 17;
```

```cpp
constexpr int multiply_add(int a, int b, int c) {
    return a * b + c;
}

int main() {
    constexpr int result = multiply_add(3, 4, 5); // folded at compile time
    std::cout << "result = " << result << std::endl;
    return 0;
}
```

```cpp
constexpr int result = 17;
```

*Kenneth J. Roche*
*University of Washington*

*End Lecture 11*

*Kenneth J. Roche*
*University of Washington*
*AMATH 483 / 583 High Performance Scientific Computing*
*Spring Quarter 2025*