STAT 309: MATHEMATICAL COMPUTATIONS I
FALL 2015
LECTURE 1

1. LAUNDRY LIST

- web site: `http://www.stat.uchicago.edu/~lekheng/courses/309/`
- notes: `http://www.stat.uchicago.edu/~lekheng/courses/309/notes/`
- books: `http://www.stat.uchicago.edu/~lekheng/courses/309/books/`
- last year's notes: `http://www.stat.uchicago.edu/~lekheng/courses/309f14/notes/`
- no required textbook
- main references:
  - Trefethen and Bau
  - Watkins
  - Golub and Van Loan
  - Demmel
- facts about matrices:
  - Bernstein
- lectures:
  - no lecture on Tue, Oct 6
  - make-up lectures on Mon, Oct 12 and Mon, Oct 19 in Eckhart 133, 5:30–8:30pm
- homework:
  - homework due beginning of class
  - collaboration allowed but must be declared
  - six assignments, lowest score will be dropped, accounting for 50% of grade
  - no late homework will be accepted
- exams:
  - two quizzes on Thu, Oct 29 and Tue, Nov 24 in Eckhart 133, 3:00–4:20pm
  - in-class, closed-book, no cheat sheet, 2 hr 50 min
- grade: 50% homework, 50% exams
- office hours: Wed, 1:30pm–3:00pm, Eckhart 122

2. NUMERICAL ANALYSIS

- numerical analysis: study of algorithms for continuous mathematics
- examples:
  - linear partial differential equation: given c_α's, find f

$$\sum_{|\alpha|\le n} c_\alpha(\mathbf{t})\frac{\partial^\alpha}{\partial \mathbf{t}^\alpha} f(\mathbf{t}) = 0 \tag{2.1}$$

  - Fredholm integral equation of the first kind: given K and g, find f

$$\int_\Omega K(\mathbf{s},\mathbf{t}) f(\mathbf{t})\, d\mathbf{t} = g(\mathbf{s}) \tag{2.2}$$

Date: September 29, 2015, version 1.0. Comments, bug reports: `lekheng@galton.uchicago.edu`.

- linear eigenvalue problem: given c_α's, find f and λ

$$\sum_{|\alpha|\le n} c_\alpha(\mathbf{t}) \frac{\partial^\alpha}{\partial \mathbf{t}^\alpha} f(\mathbf{t}) = \lambda f(\mathbf{t}) \tag{2.3}$$

- Fredholm integral equation of the first kind: given K and g, find f and λ

$$g(\mathbf{s}) + \lambda \int_\Omega K(\mathbf{s},\mathbf{t}) f(\mathbf{t})\, d\mathbf{t} = f(\mathbf{s}) \tag{2.4}$$

- nonlinear optimization: given $f_0, \dots, f_m$, find $\mathbf{t}_{\min}$

$$\min f_0(\mathbf{t}) \quad \text{subject to} \quad f_1(\mathbf{t}) \le 0, \dots, f_m(\mathbf{t}) \le 0 \tag{2.5}$$

- many scientific and engineering can be formulated in one of these forms — the PDE or integral equations would be a mathematical formulation of physical principles like Newton's second law or Maxwell equations or Schrödinger equation
- we can rarely solve these analytically, i.e., give a useful closed-form formula for the solution
- have to rely on computers, which can only deal with discrete problems
- discretization of (2.1) or (2.2), or Newton method applied to (2.5) yields

$$A\mathbf{x} = \mathbf{b} \tag{2.6}$$

- discretization of (2.3) or (2.4) yields

$$A\mathbf{x} = \lambda\mathbf{x} \tag{2.7}$$

- when we discretize, we have

$$\mathbf{x} = \begin{bmatrix} f(\mathbf{t}_1) \\ f(\mathbf{t}_2) \\ \vdots \\ f(\mathbf{t}_n) \end{bmatrix}$$

- solving for $\mathbf{x}$ gives us a sample of point values of f, which is often enough for many purposes
- the larger n is, the more information we get about f
- the matrix A comes from discretization of the linear operator — differential operators in the case of (2.1) or (2.3) and integral operators in the case of (2.2) or (2.4)
- example: discretizing a 1-dimensional differential operator

$$\frac{d^2}{dt^2} \xrightarrow{\text{discretize}} \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{n\times n}$$

- example: discretizing a 2-dimensional differential operator

$$\frac{\partial^2}{\partial t_1^2} + \frac{\partial^2}{\partial t_2^2} \xrightarrow{\text{discretize}} \begin{bmatrix} D & -I & & \\ -I & D & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & D \end{bmatrix} \in \mathbb{R}^{mn\times mn}, \text{ where } D = \begin{bmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{bmatrix} \in \mathbb{R}^{m\times m}$$

- bottom line: many problems in science and engineering require that we solve (2.6) or (2.7)

3. OPTIMIZATION

- suppose you want to solve an optimization problem

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & h_i(\mathbf{x}) \le 0, & i = 1, \dots, p, \\ & A\mathbf{x} = \mathbf{b} & \end{array}$$

- one of the most widely used algorithm is interior point method (essentially Newton's method adpated to a constrained optimization problem) which requires us to solve a linear system of the form

$$\begin{bmatrix} t\nabla^2 f(\mathbf{x}_k) + \nabla^2\varphi(\mathbf{x}_k) & A^{\mathsf{T}} \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_k \\ \boldsymbol{\nu}_k \end{bmatrix} = -\begin{bmatrix} t\nabla f(\mathbf{x}_k) + \nabla\varphi(\mathbf{x}_k) \\ 0 \end{bmatrix}$$

  where φ is the so-called log barrier function that 'traps' the iterates $\mathbf{x}_k$ within the region defined by the constraints
- at each iterate $\mathbf{x}_k$, we will have to solve such a linear system for $\Delta\mathbf{x}_k$ to obtain the next iterate $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k$
- so the computational cost of interior point methods is largely dominated by the cost of solving linear systems

4. MACHINE LEARNING

- many modern problems are information theoretic in nature
  - no differential or integral equations describing your solution f
  - but a large *test set* of given data $\{(x_i, f(x_i)) : i = 1, \dots, n\}$ that allows you to guess your f
- example: classification problems
  - spam identification

$$f : \texttt{emails} \to \{\texttt{spam}, \texttt{nonspam}\}$$

  - image recognition

$$f : \texttt{facial images} \to \{\texttt{male}, \texttt{female}\}$$

    or more generally

$$f : \texttt{handwritten digits} \to \{\texttt{0,1,2,3,4,5,6,7,8,9}\}$$

  - there is no 'Newton's law' type of rule to describe f
- example: supervised learning for binary classification

$$f : X \to \{-1, +1\}$$

  - given training set $\Omega = \{x_1, \dots, x_n\} \subseteq X$, i.e., we already know the value $f(x_i) = y_i$ for any $x_i \in \Omega$
  - want to find f, i.e., given some $x \notin \Omega$, we want to predict the value $f(x)$
  - let us use spam identification as an example, then for any e-mail $x \in X$,

$$f(x) = \begin{cases} -1 & \text{if } x \text{ is spam} \\ +1 & \text{if } x \text{ is not spam} \end{cases}$$

  - we can encode an e-mail as a vector in $\mathbb{R}^N$, for example, by counting word frequencies
  - so if you like you may assume that $X \subseteq \mathbb{R}^N$ where N is very large
- one way to do this:

- assume that

$$f(x) = \sum_{i=1}^{n} c_i K(x, x_i) \tag{4.1}$$

  where $K : X \times X \to \mathbb{R}$ is some suitable *Mercer kernel*
- if $X \subseteq \mathbb{R}^N$ a common example is the Gaussian kernel

$$K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2}$$

- since we already know the value of $f(x)$ for $x \in \{x_1, \dots, x_n\}$, we could in principle determine $c_1, \dots, c_n$ by plugging $x_1, \dots, x_n$ into (4.1) to get

$$\begin{aligned} f(x_1) &= c_1 K(x_1, x_1) + \cdots + c_n K(x_1, x_n) \\ f(x_2) &= c_1 K(x_2, x_1) + \cdots + c_n K(x_2, x_n) \\ &\cdots \\ f(x_n) &= c_1 K(x_n, x_1) + \cdots + c_n K(x_n, x_n) \end{aligned}$$

  or equivalently

$$\begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) & \cdots & K(x_1,x_n) \\ K(x_2,x_1) & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ K(x_n,x_1) & \cdots & \cdots & K(x_n,x_n) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix} \tag{4.2}$$

  or

$$K\mathbf{c} = \mathbf{y}$$

- note that we know all values $K(x_i, x_j)$ of the matrix and also the right-hand side $f(x_i)$ as long as we have the training set $\{(x_i, f(x_i) : i = 1, \dots, n\}$
- so we end up with a linear system like (2.6) again

• in principle this is very nice but in practice it rarely works since (4.2) is unlikely to have a solution
• so what we often need to do is to solve linear systems (2.6) approximately, i.e., $A\mathbf{x} \approx \mathbf{b}$ where '$\approx$' is interpreted in some appropriate ways — we will look at some of these variants of (2.6) later
• the most common interpretation of $A\mathbf{x} \approx \mathbf{b}$ is the *least squares problem*

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2^2 = \min_{x_1,\dots,x_n\in\mathbb{R}} \sum_{i=1}^{m} \sum_{j=1}^{n} (a_{ij}x_j - b_i)^2 \tag{4.3}$$

• in the context of supervised learning, this is called *empical risk minimization*, i.e., find $c_1, \dots, c_n$ so that

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{4.4}$$

is minimized
• but (4.4) is often *ill-posed* (no unqiue solution) and so a common strategy is to do Tikhonov regularization and minimize instead

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \|f\|_K^2$$

where $\|\cdot\|_K$ is a special norm induced by the kernel $K(x, y)$ (if you must know, it is called the reproducing kernel Hilbert space or RKHS norm)

- as we will see later in this course, this leads to a problem of the form
$$(K+\lambda I)\mathbf{c}=\mathbf{y}$$
which is again a linear system except that we will need to find λ separately (we will see how to do this)
- now once we have $c_1,\dots,c_n$, given any x, we can find the value $f(x)$
- of course $f(x)$ would in general not be ± 1 but we can design a rule of the form
$$f(x)\begin{cases}<0 & \Rightarrow x \text{ is spam}\\ >0 & \Rightarrow x \text{ is not spam}\end{cases}$$
- so we have bulit a spam filter

5. solving linear systems

- most of the course will focus on solving *linear systems* (2.6) and its variants like least squares, regularized least squares, total least squares, etc
- the fundamental problem is
$$A\mathbf{x}=\mathbf{b}$$
where we are given $A\in\mathbb{C}^{m\times n}$, $\mathbf{b}\in\mathbb{C}^m$ and we seek a solution $\mathbf{x}\in\mathbb{C}^n$
- often we will work over $\mathbb{R}$ instead of $\mathbb{C}$ but these would be only fields of interest
- some of the stuff we say in this course will be false over arbitrary fields (e.g. $\mathbb{F}_2=\{0,1\}$ with mod 2 arithmetic)
- three important numbers associated to a matrix A or a linear system $A\mathbf{x}=\mathbf{b}$:
  - m = number of rows = number of equations
  - n = number of columns = number of variables
  - $r=\operatorname{rank}(A)=\dim(\operatorname{im}(A))=\dim(\operatorname{colsp}(A))=\dim(\operatorname{rowsp}(A))$
- m,n,r tell us about existence and uniqueness of solution to $A\mathbf{x}=\mathbf{b}$
- terminologies
  - $m=n$: A is square matrix, $A\mathbf{x}=\mathbf{b}$ is a square system, i.e. number of variables equals number of equations
$$A=\begin{bmatrix}\times&\times&\times&\times\\\times&\times&\times&\times\\\times&\times&\times&\times\\\times&\times&\times&\times\end{bmatrix}$$
  - $m>n$: A is tall-and-thin matrix, $A\mathbf{x}=\mathbf{b}$ is an overdetermined system, i.e., more equations than variables
$$A=\begin{bmatrix}\times&\times\\\times&\times\\\times&\times\\\times&\times\end{bmatrix}$$
  - $m<n$: A is short-and-fat matrix, $A\mathbf{x}=\mathbf{b}$ is an underdetermined system, i.e., more variables than equations
$$A=\begin{bmatrix}\times&\times&\times&\times\\\times&\times&\times&\times\end{bmatrix}$$
  - if $r=\min\{m,n\}$, we say A is of *full rank*, otherwise we say A is *rank deficient*
  - if $r=\min\{m,n\}=m$, we say A is of *full row rank*
  - if $r=\min\{m,n\}=n$, we say A is of *full column rank*
- question: what is the big deal about solving linear systems $A\mathbf{x}=\mathbf{b}$? don't we know all about this already?

- answer: we only know how to solve idealized versions of the problem, but not in realistic situations
  - what if there are rounding errors in the coefficient matrix A or the right hand side $\mathbf{b}$ or both
  - what if we want to solve it quicker than $O(n^3)$
  - what if m and n are large
  - what if we want to do things in parallel on multicore processors
  - what if we need to deal with a variant with constraints on the solution $\mathbf{x}$, or where $A\mathbf{x} = \mathbf{b}$ has no solution or no unique solution (as we saw in the machine learning example)
- linear systems are arguably the most widely solved problem in science and engineering
  - 70% of supercomputing time is spent on this
  - that's why solution of linear system is used to benchmark supercomputers (cf. `http://www.top500.org`)

6. top 10 Algorithms of the 20th Century

- a broader motivation for this course and its sequel next quarter is that matrix computations are behind some of the most important algorithms
- the three bold faced ones are algorithms in matrix computations
- the four italics ones are algorithms are variants or extensions of algorithms in matrix computations
- see `http://www.stat.uchicago.edu/~lekheng/courses/309/top10/`

(1) Metropolis Algorithm for Monte Carlo
(2) *Simplex Method for Linear Programming*
(3) **Krylov Subspace Iteration Methods**
(4) **Decompositional Approach to Matrix Computations**
(5) Fortran Optimizing Compiler
(6) **QR Algorithm for Computing Eigenvalues**
(7) Quicksort Algorithm for Sorting
(8) *Fast Fourier Transform*
(9) *Integer Relation Detection*
(10) *Fast Multipole Method*

7. variants of $A\mathbf{x} = \mathbf{b}$

- notations
  - $\mathbf{x} = [x_1, \dots, x_n]^{\mathsf{T}} \in \mathbb{R}^n$

$$\|\mathbf{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}$$
$$\|\mathbf{x}\|_1 := |x_1| + \cdots + |x_n|$$
$$\|\mathbf{x}\|_\infty := \max\{|x_1|, \dots, |x_n|\}$$
$$\|\mathbf{x}\|_0 := \operatorname{nnz}(\mathbf{x}) = \#\{i : x_i \neq 0\}$$

  - $A = [a_{ij}]_{i,j=1}^{m,n} \in \mathbb{R}^{m\times n}$

$$\|A\|_F = \sqrt{\sum\nolimits_{i,j=1}^{m,n} |a_{ij}|^2}$$

  - we will discuss vector and matrix norms below
  - note that $\|\cdot\|_0$ is not a norm

(1) linear regression or least squares problem: know A exactly but $\mathbf{b}$ is corrupted by error $\mathbf{r}$, i.e., $A\mathbf{x} = \mathbf{b} + \mathbf{r}$, and we want an $\mathbf{x}$ that minimizes $\mathbf{r}$,

$$\min\{\|\mathbf{r}\|_2^2 : A\mathbf{x} = \mathbf{b} + \mathbf{r}\} = \min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2^2 \tag{7.1}$$

Gauss–Markov theorem says that such an $\mathbf{x}$ is the maximum likelihood estimator if the error $\mathbf{r}$ is from a distribution that has zero mean and finite variance

(2) error-in-variables regression or total least squares problem: A and $\mathbf{b}$ are both corrupted by error E and $\mathbf{r}$, i.e., $(A + E)\mathbf{x} = \mathbf{b} + \mathbf{r}$, and we want an $\mathbf{x}$ that minimizes both E and $\mathbf{r}$,

$$\min\{\|E\|_F^2 + \|\mathbf{r}\|_2^2 : (A + E)\mathbf{x} = \mathbf{b} + \mathbf{r}\}$$

(3) data least squares problem: A is corrupted by error E, i.e., $(A+E)\mathbf{x} = \mathbf{b}$, and we want an $\mathbf{x}$ that minimizes E,

$$\min\{\|E\|_F^2 : (A + E)\mathbf{x} = \mathbf{b}\}$$

(4) minimum norm least squares: want the minimum length solution to (7.1),

$$\min\{\|\mathbf{x}\|_2^2 : \mathbf{x} \in \operatorname{argmin}\|A\mathbf{x} - \mathbf{b}\|_2^2\} = \min\{\|\mathbf{x}\|_2^2 : A^{\mathsf{T}}A\mathbf{x} = A^{\mathsf{T}}\mathbf{b}\} \tag{7.2}$$

the solution $\mathbf{x}_*$ to (7.2) is unique and can in fact be used to define the Moore–Penrose pseudoinverse of A: $\mathbf{x}_* = A^{\dagger}\mathbf{b}$

(5) robust regression: replace 2-norm by 1-norm (more generally, the Huber loss function) in (7.1),

$$\min\{\|\mathbf{r}\|_1 : A\mathbf{x} = \mathbf{b} + \mathbf{r}\} = \min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_1$$

great for reducing sensitivity to outliers

(6) ridge regression or regularized least squares

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2^2 + \|\Gamma\mathbf{x}\|_2^2$$

where $\Gamma \in \mathbb{R}^{p\times n}$ is some other matrix — most commonly $\Gamma = \lambda I$ or the finite-difference matrix

(7) sparse or structured linear systems: sparse means A has a lot of zeroes (sufficiently many that it pays to take advantage of the fact), structured means that A can be defined with fewer than the usual number of mn parameters. An example of a data sparse matrix is a Toeplitz matrix

$$T = \begin{bmatrix} a_0 & a_1 & a_2 & & a_{n-1} \\ a_{-1} & a_0 & a_2 & \ddots & \\ a_{-2} & a_{-1} & \ddots & \ddots & a_2 \\ & \ddots & \ddots & \ddots & a_1 \\ a_{-n+1} & & a_{-2} & a_{-1} & a_0 \end{bmatrix} \in \mathbb{R}^{n\times n}$$

i.e., a_{ij} depends only on $|i - j|$, and T can be specified with just $2n - 1$ parameters $a_{-n+1}, \dots, a_{n+1} \in \mathbb{R}$; a Toeplitz $T\mathbf{x} = \mathbf{b}$ can be solved in $O(n \log^2 n)$ time as opposed to the usual $O(n^3)$ time for general linear systems

(8) linear programming:

$$\min\{\mathbf{c}^{\mathsf{T}}\mathbf{x} : A\mathbf{x} \le \mathbf{b}\}$$

note that $\mathbf{c}^{\mathsf{T}}\mathbf{x} = c_1x_1 + \cdots + c_nx_n$ is a linear function; this is very important in economics

(9) quadratic programming: given $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times n}$, $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{d} \in \mathbb{R}^m$, want

$$\min\left\{\tfrac{1}{2}\mathbf{x}^{\mathsf{T}}A\mathbf{x} - \mathbf{c}^{\mathsf{T}}\mathbf{x} : B\mathbf{x} = \mathbf{d}\right\}$$

this reduces to a linear system

$$\begin{bmatrix} A & B^{\mathsf{T}} \\ B & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix}$$

(10) basis pursuit: if we want the sparsest solution to an underdetermined linear system, we ought to solve

$$\min\{\|\mathbf{x}\|_0 : A\mathbf{x} = \mathbf{b}\}$$

but this is NP-hard and so we look at a convex relaxation

$$\min\{\|\mathbf{x}\|_1 : A\mathbf{x} = \mathbf{b}\}$$

which can in fact be reduced to a linear programming problem

8. norms

- a *norm* is a real-valued function on a vector space (over $\mathbb{R}$ or $\mathbb{C}$), denoted $\|\cdot\| : V \to \mathbb{R}$ satisfying
  (1) $\|\mathbf{x}\| \geq 0$ for all $\mathbf{x} \in V$
  (2) $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$
  (3) $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$ for all $\alpha \in \mathbb{C}$ and $\mathbf{x} \in V$
  (4) $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ for any $\mathbf{x}, \mathbf{y} \in V$
- we will be interested in two specific choices of V
  - $V = \mathbb{R}^n$ or $\mathbb{C}^n$
  - $V = \mathbb{R}^{m\times n}$ or $\mathbb{C}^{m\times n}$

9. vector norms

- if $V = \mathbb{C}^n$ or $V = \mathbb{R}^n$, we call a norm on V a *vector norm*
- example: consider $\|\cdot\|_1 : \mathbb{C}^n \to \mathbb{R}$ defined by

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$$

  for $\mathbf{x} = [x_1, \dots, x_n]^{\mathsf{T}} \in \mathbb{C}^n$ and where $|x|$ denotes the modulus/absolute value of $x \in \mathbb{C}$
  - check that this is a norm:
    (1) clearly $\|\mathbf{x}\|_1 \geq 0$
    (2) the only way a sum nonnegative entries $\|\mathbf{x}\|_1 = 0$ is if all entries $|x_i| = 0$ and so $\mathbf{x} = [0, \dots, 0]^{\mathsf{T}} = \mathbf{0}$
    (3) we have

$$\|\alpha\mathbf{x}\|_1 = \sum_{i=1}^{n} |\alpha x_i| = |\alpha| \sum_{i=1}^{n} |x_i| = |\alpha|\|\mathbf{x}\|_1$$

    since complex modulus satisfies $|\alpha x| = |\alpha||x|$
    (4) using the triangle inequality for complex numbers, we obtain

$$\|\mathbf{x} + \mathbf{y}\|_1 = \sum_{i=1}^{n} |x_i + y_i| \leq \sum_{i=1}^{n} |x_i| + |y_i| \leq \|\mathbf{x}\|_1 + \|\mathbf{y}\|_1$$

  - therefore the function defines a norm, called the 1-*norm* or *Manhattan norm*
- example: more generally, for $p \geq 1$ (can be any real number, not necessarily an integer), we define the *p-norm* $\|\mathbf{x}\|_p$ by

$$\|\mathbf{x}\|_p = (|x_1|^p + \cdots + |x_n|^p)^{1/p}$$

– most commonly used p-norms is the 2-*norm* or *Euclidean norm*:

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2\right)^{1/2}$$

– easy to see that for any p, we have

$$\left(\max_{i=1,\dots,n} |x_i|^p\right)^{1/p} \le \|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \le \left(n \max_{i=1,\dots,n} |x_i|^p\right)^{1/p}$$

– from which it follows that

$$\max_{i=1,\dots,n} |x_i| \le \|\mathbf{x}\|_p \le n^{1/p} \max_{i=1,\dots,n} |x_i|$$

– as $p \to \infty$, we obtain the *infinity norm*

$$\|\mathbf{x}\|_\infty = \lim_{p\to\infty} \|\mathbf{x}\|_p = \max_{i=1,\dots,n} |x_i|$$

which is also known as the *Chebyshev norm*

– easy to verify that p-norms for any $p \in [1, \infty]$ are indeed norms