Linear Algebra – Đại Số Tuyến Tính

Nguyễn Quản Bá Hồng*

Ngày 9 tháng 2 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: https://nqbh.github.io/advanced_STEM/.
Latest version:

- *Linear Algebra – Đại Số Tuyến Tính.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/linear_algebra/NQBH_linear_algebra.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/linear_algebra/NQBH_linear_algebra.tex.

Mục lục

1 Basic Linear Algebra

Tôi được giải Nhì Đại số Olympic Toán Sinh viên 2014 (VMC2014) khi còn học năm nhất Đại học & được giải Nhất Đại số Olympic Toán Sinh viên 2015 (VMC2015) khi học năm 2 Đại học. Nhưng điều đó không có nghĩa là tôi giỏi Đại số. Bằng chứng là 10 năm sau khi nhận các giải đó, tôi đang tự học lại Đại số tuyến tính với hy vọng có 1 hay nhiều cách nhìn mới mẻ hơn & mang tính ứng dụng hơn cho các đề tài cá nhân của tôi.

Resources – Tài nguyên.

- [Hưn22]. Nguyễn Hữu Việt Hưng. *Đại Số Tuyến Tính.*
- [Tiệ25]. Vũ Hữu Tiệp. *Machine Learning Cơ Bản.*
  Mã nguồn cuốn ebook "Machine Learning Cơ Bản": https://github.com/tiepvupsu/ebookMLCB.

*A Scientist & Creative Artist Wannabe. E-mail: nguyenquanbahong@gmail.com. Bến Tre City, Việt Nam.

Phép nhân từng phần/tích Hadamard (Hadamard product) thường xuyên được sử dụng trong ML. Tích Hadamard của 2 ma trận cùng kích thước $A, B \in \mathbb{R}^{m\times n}$, được ký hiệu là $A \odot B = (a_{ij}b_{ij})_{i,j=1}^{m,n} \in \mathbb{R}^{m\times n}$.

Việc chuyển đổi hệ cơ sở sử dụng ma trận trực giao có thể được coi như 1 phép xoay trục tọa độ. Nhìn theo 1 cách khác, đây cũng chính là 1 phép xoay vector dữ liệu theo chiều ngược lại, nếu ta coi các trục tọa độ là cố định.

Việc phân tích 1 đại lượng toán học ra thành các đại lượng nhỏ hơn mang lại nhiều hiệu quả. Phân tích 1 số thành tích các thừa số nguyên tố giúp kiểm tra 1 số có bao nhiêu ước số. Phân tích đa thức thành nhân tử giúp tìm nghiệm của đa thức. Việc phân tích 1 ma trận thành tích của các ma trận đặc biệt cũng mang lại nhiều lợi ích trong việc giải hệ phương trình tuyến tính, tính lũy thừa của ma trận, xấp xỉ ma trận, ...

Phép phân tích trị riêng. Cách biểu diễn 1 ma trận vuông A với $\mathbf{x}_i \neq \mathbf{0}$ là các vector riêng của 1 ma trận vuông A ứng với các giá trị riêng lặp hoặc phức λ_i: $A\mathbf{x}_i = \lambda_i\mathbf{x}_i$, $\forall i = 1, \dots, n$: $A = X\Lambda X^{-1}$ với $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$, $X = [\mathbf{x}_1, \dots, \mathbf{x}_n]$.

Norm – Chuẩn. Khoảng cách Euclid chính là độ dài đoạn thẳng nối 2 điểm trong mặt phẳng. Đôi khi, để đi từ 1 điểm này tới 1 điểm kia, không thể đi bằng đường thẳng vì còn phụ thuộc vào hình dạng đường đi nối giữa 2 điểm. Cf. đường trắc địa trong Hình học Vi phân – geodesics in Differential Geometry. Việc đo khoảng cách giữa 2 điểm dữ liệu nhiều chiều rất cần thiết trong ML – chính là lý do khái niệm *chuẩn* (norm) ra đời.

Trace – Vết. *Vết* (trace) của 1 ma trận vuông A được ký hiệu là $\text{trace}\, A$ là tổng tất cả các phần tử trên đường chéo chính của nó. Hàm vết xác định trên tập các ma trận vuông được sử dụng nhiều trong tối ưu vì nó có các tính chất đẹp.

Kiểm tra gradient. Việc tính gradient của hàm nhiều biến thông thường khá phức tạp & rất dễ mắc lỗi. Trong thực nghiệm, có 1 cách để kiểm tra liệu gradient tính được có chính xác không. Cách này dựa trên định nghĩa của đạo hàm cho hàm 1 biến.

1.1 [TB97; TB22]. Lloyd N. Trefethen, David Bau III. Numerical Linear Algebra

[NQBH]: There is a new version 2e 2022 but I cannot download it from Libgen or anywhere.

- Preface. Since early 1980s, 1st author has taught a graduate course in numerical linear algebra at MIT & Cornell. Alumni of this course, now numbering in hundreds, have been graduate students in all fields of engineering & physical sciences. This book is an attempt to put this course on paper.

  In field of numerical linear algebra, there is already an encyclopedic treatment on market: *Matrix Computations*, by Golub & Van Loan, now in its 3e. This book is in no way an attempt to duplicate that one. It is small, scaled to size of 1 university semester. Its aim: present fundamental ideas in as elegant a fashion as possible. Hope: every reader of this book will have access also to Golub & Van Loan for pursuit of further details & additional topics, & for its extensive references to research literature. 2 other important recent books are those of Higham & Demmel.

  Field of numerical linear algebra is more beautiful, & more fundamental, than its rather dull name may suggest. More beautiful, because it is full of powerful ideas that are quite unlike those normally emphasized in a linear algebra course in a mathematics department. (At end of semester, students invariably comment: there is more to this subject than they ever imagined.) More fundamental, because, thanks to a trick of history, "numerical" linear algebra is really *applied* linear algebra. Here one finds essential ideas that every mathematical scientist needs to work effectively with vectors & matrices. In fact, our subject is more than just vectors & matrices, for virtually everything we do carries over to functions & operators. Numerical linear algebra is really functional analysis, but with emphasis always on practical algorithmic ideas rather than mathematical technicalities.

  Book is divided into 40 lectures. Have tried to build each lecture around 1 or 2 central ideas, emphasizing unity between topics & never getting lost in details. In many places our treatment is nonstandard. This is not place to list all of these points (see Notes), but will mention 1 unusual aspect of this book. Have departed from customary practice by not starting with Gaussian elimination. That algorithm is atypical (khác biệt) of numerical linear algebra, exceptionally difficult to analyze, yet at same time tediously familiar to every student entering a course like this. Instead, begin with QR factorization, which is more important, less complicated, & a fresher idea to most students. QR factorization is thread that connects most of algorithms of numerical linear algebra, including methods for least squares, eigenvalue, & singular value problems, as well as iterative methods $\forall$ of these & also for systems of equations. Since 1970s, iterative methods have moved to center stage in scientific computing, & to them, devote last part of book.

  Hope reader will come to share our view: if any other mathematical topic is as fundamental to mathematical sciences as calculus & differential equations, it is numerical linear algebra.

- I. Fundamentals.
  - 1. Matrix-Vector Multiplication.
  - 2. Orthogonal Vectors & Matrices.
  - 3. Norms.
  - 4. Singular Value Decomposition SVD.
  - 5. More on SVD.
- II. QR Factorization & Least Squares.
  - 6. Projectors.

- 7. QR Factorization.
- 8. Gram–Schmidt Orthogonalization.
- 9. MATLAB.
- 10. Householder Triangularization.
- 11. Least Squares Problems. Least squares data-fitting has been an indispensable tool since its invention by GAUSS & LEGENDRE around 1800, with ramifications (hậu quả) extending throughout mathematical science. In language of linear algebra, problem here is solution of an overdetermined system of equaitons $A\mathbf{x} = \mathbf{b}$ – rectangular with more rows than columns. Least squares idea: "solve" such a system by minimizing 2-norm of residual $\mathbf{b} - A\mathbf{x}$.

  * Problem. Consider a linear system of equations having n unknowns but $m > n$ equations. Symbolically, wish to find a vector $\mathbf{x} \in \mathbb{C}^n$ that satisfies $A\mathbf{x} = \mathbf{b}$, where $A \in \mathbb{C}^{m\times n}, b \in \mathbb{C}^m$. In general, such a problem has no solution. A suitable vector $\mathbf{x}$ exists only if $\mathbf{b}$ lies in range(A), & since $\mathbf{b}$ is an m-vector, whereas range(A) is of dimension $\leq n$, this is true only for exceptional choices of $\mathbf{b}$. Say: a rectangular system of equations with $m > n$ is *overdetermined*. Vector known as *residual*: $\mathbf{r} = \mathbf{b} - A\mathbf{x} \in \mathbb{C}^m$, can perhaps be made quite small by a suitable choice of $\mathbf{x}$, but in general it cannot be made $= 0$.
    What can it mean to solve a problem that has no solution? In case of an overdetermined system of equations, there is a natural answer to this question. Since residual r cannot be made to be 0, instead made it as small as possible. Measuring smallness of r entails choosing a norm. If choose 2-norm, problem takes following form: (11.2)

    Given $A \in \mathbb{C}^{m\times n}, m \geq n, b \in \mathbb{C}^m$, find $\mathbf{x} \in \mathbb{C}^n$ s.t. $\|\mathbf{b} - A\mathbf{x}\|_2$ is minimized.

    This is our formulation of general (linear) *least squares problem*. Choice of 2-norm can be defended by various geometric & statistical arguments, & it certainly leads to a simple algorithms – ultimately because derivative of a quadratic function, which must be set to 0 for minimization, is linear.
    2-norm corresponds to Euclidean distance, so there is a simple geometric interpretation of (11.2). Seek a vector $\mathbf{x} \in \mathbb{C}^n$ s.t. vector $A\mathbf{x} \in \mathbb{C}^m$ is closest point in range(A) to $\mathbf{b}$
  * Example: Polynomial Data-Fitting. Compare polynomial interpolation, which leads to a square system of equations, & least squares polynomial data-fitting, where system is rectangular.
    Example 1 (Polynomial Interpolation)**.** *Suppose given m distinct points $x_1, \ldots, x_m \in \mathbb{C}$ & data $y_1, \ldots, y_m \in \mathbb{C}$ at these points. Then there exists a unique* polynomial interpolant *to these data in these points, i.e., a polynomial of degree at most $m-1$, $p(x) = \sum_{i=0}^{m-1} c_i x^i$, with property: at each x_i, $p(x_i) = y_i$. Relationship of data $\{x_i\}, \{y_i\}$ to coefficients $\{c_i\}$ can be expressed by square Vandermonde system. To determine coefficients $\{c_i\}$ for a given set of data, can solve this system of equations, which is guaranteed to be nonsingular as long as points $\{x_i\}$ are distinct because $D_n = \prod_{1\leq j<i\leq n}(x_i - x_j)$.*
    Fig. 11.1: Degree 10 polynomial interpolant to 11 data points. Axis scales are not given, as these have no effect on picture. *presents an example of this process of polynomial interpolation. Have 11 data points in form of a discrete square wave, represented by crosses, & curve $p(x)$ passes through them, as it must. However, fit is not at all pleasing. near ends of interval, $p(x)$ exhibits large oscillations that are clearly an artifact (hiện vật) of interpolation process, not a reasonable reflection of data.*
    This unsatisfactory behavior is typical of polynomial interpolation. Fits it produces are often bad, & they tend to get worse rather than better if more data are utilized. Even if fit is good, interpolation process may be ill-conditioned, i.e., sensitive to perturbations of data. To avoid these problems, one can utilize a nonuniform set of interpolation points e.g. Chebyshev points in interval $[-1, 1]$. In applications, however, it will not always be possible to choose interpolation points at will.
    Example 2 (Polynomial Least Squares Fitting)**.** *Without changing data points, can do better by reducing degree of polynomial. Given $x_1, \ldots, x_m, y_1, \ldots, y_m$ again, consider now a degree $n-1$ polynomial $p(x) = \sum_{i=0}^{n-1} c_i x^i$ for some $n < m$. Such a polynomial is a least squares fit to data if it minimizes sum of squares of deviation from data,*

$$\sum_{i=1}^{m} |p(x_i) - y_i|^2.$$

    This sum of squares = square of norm of residual $\|\mathbf{r}\|_2^2$ for rectangular Vandermonde system (11.7). Fig. 11.2: Degree 7 polynomial least squares fit to same 11 data points. *illustrates what we get if fit same 11 data points from last example with a polynomial of degree 7. New polynomial does not interpolate data, but it* captures their overall behavior much better *than polynomial of Example 11.1. Though on cannot see this in figure, it is also less sensitive to perturbations.*
  * Orthogonal Projection & Normal Equations. How was Fig. 11.2 computed? How are least squares problems solved in general? Key to deriving algorithms is orthogonal projection.
    Idea is illustrated in Fig. 11.3: Formulation of least squares problem (11.2) in terms of orthogonal projection. Goal: find closest point $A\mathbf{x}$ in range(A) to $\mathbf{b}$, so that norm of residual $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ is minimized. Clear geometrically: this will occur provided $A\mathbf{x} = P\mathbf{b}$ where $P \in \mathbb{C}^{m\times m}$: orthogonal projector that maps $\mathbb{C}^m$ onto range(A). I.e., residual $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ must be orthogonal to range(A). Formulate this condition as following theorem.
    Theorem 1. *Let $A \in \mathbb{C}^{m\times n}$, $m \geq n$, & $\mathbf{b} \in \mathbb{C}^m$ be given. A vector $\mathbf{x} \in \mathbb{C}^n$ minimizes residual norm $\|\mathbf{r}\|_2 = \|\mathbf{b} - A\mathbf{x}\|_2$, thereby solving least squares problem (11.2), iff – – $\mathbf{r} \perp$ range(A), i.e., (11.8)–(11.9)*

$$A^*\mathbf{r} = \mathbf{0} \Leftrightarrow A^*A\mathbf{x} = A^*\mathbf{b} \Leftrightarrow Pb = A\mathbf{x},$$

    where $P \in \mathbb{C}^{m\times m}$: orthogonal projector onto range(A). *$n \times n$ system of equations (11.9), known as* normal equations, *is nonsingular iff A has full rank. Consequently solution $\mathbf{x}$ is unique iff A has full rank.*

    - Pseudoinverse.
    - Normal Equations.
    - QR Factorization.
    - SVD.
    - Comparison of Algorithms.
- III. Conditioning & Stability.
    - 12. Conditioning & Condition Numbers.
    - 13. Floating Point Arithmetic.
    - 14. Stability.
    - 15. More on Stability.
    - 16. Stability of Householder Triangularization.
    - 17. Stability of Back Substitution.
    - 18. Conditioning of Least Squares Problems.
    - 19. Stability of Least Squares Algorithms.
- IV. Systems of Equations.
    - 20. Gaussian Elimination.
    - 21. Pivoting.
    - 22. Stability of Gaussian Elimination.
    - 23. Cholesky Factorization.
- V. Eigenvalues.
    - 24. Eigenvalue Problems.
    - 25. Overview of Eigenvalue Algorithms.
    - 26. Reduction to Hessenberg or Tridiagonal Form.
    - 27. Rayleigh Quotient, Inverse Iteration.
    - 28. QR Algorithm without Shifts.
    - 29. QR Algorithm with Shifts.
    - 30. Other Eigenvalue Algorithms.
    - 31. Computing SVD.
- VI. Iterative Methods.
    - 32. Overview of Iterative Methods.
    - 33. Arnoldi Iteration.
    - 34. How Arnoldi Locates Eigenvalues.
    - 35. GMRES.
    - 36. Lanczos Iteration.
    - 37. From Lanczos to Gauss Quadrature.
    - 38. Conjugate Gradients.
    - 39. Biorthogonalization Methods.
    - 40. Preconditioning.
- Appendix. Def of Numerical Analysis.

1.2 [Tsu+23]. Makoto Tsukada, Yuji Kobayashi, Hiroshi Kaneko, Sin-Ei Takahasi, Kiyoshi Shirayanagi, Masato Noguchi. **Linear Algebra with Python: Theory & Applications**

[1 citations]

Amazon review. This textbook is for those who want to learn linear algebra from basics. After a brief mathematical introduction, it provides standard curriculum of linear algebra based on an abstract linear space. It covers, among other aspects: linear mappings & their matrix representations, basis, & dimension; matrix invariants, inner products, & norms; eigenvalues & eigenvectors; & Jordan normal forms. Detailed & self-contained proofs as well as descriptions are given ∀ theorems, formulas, & algorithms.

A unified overview of linear structures is presented by developing linear algebra from perspective of functional analysis. Advanced topics e.g. function space are taken up, along with Fourier analysis, Perron–Frobenius theorem, linear differential equations, state transition matrix & generalizied inverse matrix, SVD, tensor products, & linear regression models. These all provide a bridge to more specialized theories based on linear algebra in mathematics, physics, engineering, economics, & social sciences.

Python is used throughout book to explain linear algebra. Learning with Python interactively, readers will naturally become accustomed to Python coding. By using Python's libraries NumPy, Matplotlib, VPython, & SymPy, readers can easily perform large-scale matrix calculations, visualization of calculation results, & symbolic computations. All codes in this book can be executed on both Windows & macOS & also on Raspberry Pi.

About Author. Makoto Tsukada has been studied in field of functional analysis. He has been teaching linear algebra, analysis, & probability theory for many years. Also, he has taught programming language courses using Pascal, Prolog, C, Python, etc. Yuji Kobayashi, Hiroshi Kaneko, Sin-Ei Takahasi, Kiyoshi Shirayanagi, & Masato Noguchi are specialists in algebra, analysis, statistics, & computers.

- Preface. Wonderland of linear algebra. World of linear algebra is full of many elegant theories that lead to powerful applications. Explore this exciting realm using programming language Python as a tool.
  - **About this book.** A textbook for learning basic theory of linear algebra with awareness of its potential applications. Can better grasp specific uses & applications of linear algebra by understanding mathematical theory that underpins it.

    Linear algebra begins with theory of vectors & matrices. In high school mathematics, vectors appear in plane or space geometry & apply to areas of physics e.g. mechanics. However, this is just 1 aspect of vectors. Functions e.g. polynomials & trigonometric functions can also be regarded as vectors. In this book, reader will learn how to treat even sounds & images as vectors. Calculations related to vectors & matrices are collectively called *linear calculations*, & differentiation & integration of functions can be regarded as types of linear calculation. In physics, state of a system is formulated by a differential equation, & in order to solve it, eigenvalue problem in linear algebra must be addressed. In engineering, a technique called Fourier analysis is used in fields e.g. speech processing, image processing, communication theory, & control theory. It is closely related to notion of orthogonality defined through inner product of vectors. Calculations used in probability theory & statistics are mainly integral calculations & are associated with simultaneous linear equations & eigenvalue problems.

    Today, often encounter terms e.g. AI & big data. In order to properly understand these new fields, a solid grasp of linear algebra & its uses is becoming more & more important. In this book, build theory of linear algebra from rudiments (những điều cơ bản) to essential concepts, occasionally referring to applications mentioned.

    For undergraduate students in science or engineering departments, calculus (analysis) & linear algebra are required subjects. It might be easy to imagine utility of concrete examples e.g. velocity & acceleration for differentiation & area & volume for integration. However, harder to understand usefulness of linear algebra concepts s.t. matrix multiplication, determinants, inverse matrices, & eigenvalues. Students learn manual computational techniques in their linear algebra classes. However, as can easily imagine from above application examples, need large-scale calculations, most of which are beyond scale of manual calculations. This is why computers are both effective & indispensable in linear algebra.

    May devise an effective computer program to solve a problem if know how to solve it by hand. Today, however, there are various linear computing tools that are already coded. In many cases, do not even have to write own code. Ability to identify most appropriate tools & use them effectively is more important. Need to know various cases in which linear algebra is applied, & more importantly, need to understand principles underlying utility of tool in each individual case.

    Chaps. 9–10 cover several such cases. Before ready for these chaps, explain in Chaps. 1–8 most of theories of linear algebra in a step-by-step manner. Along way, each mathematical fact will be given as much original proof as possible, without cutting corners. In concrete numerical calculations, explain use of programming language Python as a calculator.
  - **Why Python?** Python is a free & open-source programming language that provides many useful tools for doing various calculations. It can treat fractions as fractions, calculate expressions containing character constants & variables, & display results as character expressions. Moreover, Python has many external libraries for special purposes, e.g. image processing, numerical computation, symbolic computation, graph drawing, & so on. E.g., SymPy is library for performing symbolic computations or computer algebra, e.g. solving linear or algebraic equations & factorization of polynomials. In addition to SymPy, there are many free computer algebra systems e.g. Maxima, Reduce, & SageMath, as well as commercial computer algebra systems e.g. Maple, Mathematica, & Magma. However, most of these are huge systems that aspire to be full-featured mathematics systems. On other hand, SymPy is compared & lightweight, but sufficient for performing linear algebra.

    Python has an interactive mode, in which can proceed with formula transformation while checking each calculation step as if we were using a calculator. Can also see 2D or 3D vectors displayed during calculation. Computational problems specific to linear algebra, e.g. finding determinants, inverse matrices, & eigenvalues, can also be solved in 1-line code. Indeed, many

of exercises presented in college-level linear algebra textbooks can be immediately solved using Python. Can also create new exercises, & with a little ingenuity, even exercises suitable for manual calculations with paper & pencil. 1 of authors, who has been teaching linear algebra for many years, has routinely assigned his class problems that were randomly generated in Python, so that his students will learn to solve different numerical problems of similar difficulty.

However, cannot understand true benefits of linear algebra using such exercise-level calculations. For that purpose, better to handle large-scale (high-dimensional) data e.g. audio & images. Readers find: linear algebra is more interesting if they use techniques described in this book to process their own images & sound recordings. It would not be an impractical goal to implement what we learned on a small microcomputer board e.g. Raspberry Pi to make an AI robot. Amazing: Python code as it is described in this book always always runs at practical speed.

- **About Structure & Reading of This Book.** Relationship between chaps: Fig. 1: Structure of this book. Chaps. 3–5 & 7 contain many theories peculiar to (đặc biệt đối với) finite-dimensional linear spaces, which are essence of linear algebra. Some of applications covered in this book require: build a theory in an infinite-dimensional linear space. This theory will lead us to field of functional analysis; Chap. 6 contains some topics related to this field. Reader can skip more difficult calculations of determinants & inverse matrices in Chap. 5, which might risk engendering an aversion to linear algebra (điều này có thể gây ra sự ác cảm với đại số tuyến tính). Then, after Chaps. 6–7, it may be a good idea to read Chap. 10 on applications of linear algebra 1st, bypassing more challenging Chaps. 8–9. Readers unfamiliar with Python should read Appendix 1st, where it is expanded how to set up environment for Python & how to use it. Here is an overview of each chap.
  - Chap. 1: Mathematics & Pythons. Review minimum requirements of mathematics, e.g. propositions, numbers, sets, mappings, etc., to learn concepts underpinning linear algebra. Also learn how these concepts are expressed in Python.
  - Chap. 2: Linear Spaces & Linear Mappings. Introduce concepts of linear space, which is stage set for linear algebra, & linear mapping, which plays a leading role in linear algebra. Explain linear algebra from an abstract linear space defined with axioms.
  - Chap. 3: Basis & Dimension. learn concept of basis. If there is a basis of size n in an abstract linear space, can regard it as a concrete n-dimensional space consisting of vectors with n components. A basis can be thought of as a coordinate system of a linear space, & 1 of main themes in linear algebra is how to choose a convenient coordinate system.
  - Chap. 4: Matrices. Learn: a linear mapping can be represented by a matrix. Meaning of operations e.g. matrix multiplication becomes clear here.
  - Chap. 5: Elementary Operations & Matrix Invariants. Learn about basic transformation of matrices, & use this transformation to calculate determinants & inverse matrices, or to solve simultaneous linear equations. A matrix is a representation of a linear mapping through a basis, & when basis is changed, expression of matrix changes as well. However, there are original features (invariants) associated with a linear mapping that remain unchanged under any representation. Learn how to find features of such a mapping using elementary operations.
  - Chap. 6: Inner Product & Fourier Expansion. Learn about inner product, where product of vectors is a scalar. Introduce notion of orthogonality between vectors through inner product & show how this concept develops into a method called Fourier analysis. Fourier analysis plays a major role in mathematics, engineering, physics, & more, & some of its applications are explained here.
  - Chap. 7: Eigenvalues & Eigenvectors. Learn about eigenvalues & eigenvectors of matrices, which are especially important notions in linear algebra. Eigenvalues are features that cannot be obtained merely by elementary operations of a matrix, but they appear clearly when matrix is diagonalized.
  - Chap. 8: Jordan Normal Form & Spectrum. Learn to transform & analyze a not-necessarily diagonalizable matrix into what is called *Jordan normal form*. Jordan normal form is 1 of final destinations of matrix theory. As an application of this, prove: Perron-Frobenius theorem, which often appears in applications.
  - Chap. 9: Dynamical Systems. Introduce analysis of behavior of solution of a linear differential equation as an application of Jordan normal form, & ergodic theorem of a Markov chain as an application of Perron-Frobenius theorem. Former is a system that changes deterministically over time, & latter is a system that changes stochastically.
  - Chap. 10: Applications & Development of Linear Algebra. Discuss singular value decomposition & generalized inverse matrix with some concrete applications. Rigorously explain essential parts of these application examples so that they can be completed only here. Hope this will be a stepping stone that inspires readers to consult more specialized books to examine these themes individually.
  - Appendix. Show how to prepare computer, including installation of Python, in order to read this book. Those who already have a computer environment set up can proceed to 1st chap.

  Traditionally, 1st-year college linear algebra textbooks set Jordan normal form as final goal, but do not touch on singular value decompositions or generalized inverse matrices. However, in order to understand Jordan normal form, there are hurdles (rào cản) that must be overcome, e.g., analysis of generalized eigenspace. On other hand, SVD & generalized inverse matrix are consequences of both simultaneous equation theory & eigenvalue problem, which are 2 major themes of linear algebra. Moreover, they are adaptable to any matrices including non-square matrices, & are presently used in applications more than Jordan normal form. Think these topics should be taken up not only due to their importance but also from aspect of motivation for learning linear algebra. Hope: this book will "throw a stone" (a Japanese idiom meaning "introduce a new topic for discussion") at current thinking about curriculum of linear algebra.

  Reader can download all of code for programs described in this book from website. In addition, they can also access code that created inset & other diagrams, as well as solutions to exercises from same site. Website also provides latest information about Python: https://www.math-game-labo.com/.

- 1. Mathematics & Python. Survey notions of propositions, real & complex numbers, sets, & mappings (functions) that are basis of mathematics needed to study linear algebra. Also learn how these concepts are expressed in Python. Learn mathematics in a practical way by using Python.
- 2. Linear Spaces & Linear Mappings.
- 3. Basis & Dimension.
- 4. Matrices. Learn matrix representation of a linear mapping, & matrix operations defined naturally through representation. Main themes in linear algebra are "choosing a suitable basis for matrix representation to become simple" & "finding properties of linear mappings that do not depend on basis choice".

  Introduce 2 libraries, now standard in Python, NumPy for numerical computation & SymPy for symbolic computation. Need to use them selectively in accordance with our problems.

  - 4.1. Matrix Operations. Consider an $m \times n$ matrix $A = (a_{ij})_{i,j=1}^{m,n}$: a 2D arrangement of elements of $\mathbb{K}$. Taking out columns $\mathbf{a}_1, \ldots, \mathbf{a}_n$ from it in order, write it as $A = [\mathbf{a}_1\ \mathbf{a}_2\ \cdots\ \mathbf{a}_n]$. Entry a_{ij} at intersection of ith row & jth column is called (i,j)-*element* of A. In particular, a matrix of shape (n,n) with same number of rows & columns is called a *square matrix* of *order* n, & its (i,i)-elements a_{ii}, $i = 1, \ldots, n$, are called *diagonal elements*. A square matrix is called a *diagonal matrix*, if all its elements other than diagonal elements are 0. 2 matrices A, B are equal $\Leftrightarrow$ they are of same shape & corresponding elements are equal.

    For matrix calculation, use `array` of NumPy in Python as standard.

    ```
    from numpy import array
    A = [[1 , 2 , 3] , [4 , 5 , 6]]
    print(A)
    ```

    Matrix A is expressed by a nested list. Note: a list expresses only an arrangement of elements of a matrix, so in order to use it for various matrix calculations described later, need to implement them by ourselves.

    ```
    A = array(A)
    print(A)
    ```

    Cast list `A` to an array & let this array be `A` again. Without this assignment, `A` remains a list.

    ```
    [[1 2 3]
     [4 5 6]]
    ```

    Function `print` outputs matrix.

    ```
    L = A.tolist();
    print(L)
    ```

    Make list from array using method `tolist`.

    ```
    B = A.copy()
    print(B)
    ```

    Copying `A`, make another array `B` with same contents. On other hand, `B = A` means: **A, B** point to same object, & change of elements of `A` is also reflected in `B`. (So to speak, this is difference between pointing at a matrix written at some place in a notebook & rewriting same matrix with a pencil in another place.) This happens because an array is a mutable object.

    ```
    print(A == B)
    [[ True  True  True]
     [ True  True  True]]
    print((A == B).all())
    B[0, 1] = 1
    print((A == B).all())
    ```

    `A == B` compares arrays & returns array of Boolean values of componentwise comparison. Method `all` decides equality of matrices. This method return `True` if all Boolean values of comparison are `True`, whereas method `any` returns `True` if at least 1 value is `True`.

    When write a vector $\mathbf{x} \in \mathbb{K}^n$ in column form $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top$, it is a matrix of shape $(n,1)$ & is called a *column vector*. On other hand, a matrix $[x_1, \ldots, x_n]$ of shape $(1,n)$ is called a *row vector*. They are treated as follows in NumPy.

```
A = array([1, 2, 3])
print(A)
B = A.reshape((1, 3))
print(B)
C = B.reshape((3, 1))
print(C)
D = C.reshape((3,))
print(D)
A[0] = 0
print(A)
print(B[0, 0], C[0, 0], D[0])
```

Row vector & column vector can be converted to each other by method `reshape` of array. However, note: when converted by method `reshape`, corresponding element in each expression point to same thing. (Method `reshape` just changes way of reference to same memory by subscripts of array.) E.g., if some element in `A` is changed, corresponding elements in `B, C, D` are affected. If want to avoid such effects, need to make `B, C, D` are copies of reshaped arrays.

Set $M_{\mathbb{K}}(m,n)$ of all matrices of same shape (m,n) whose elements belong to $\mathbb{K}$ forms a linear space over $\mathbb{K}$. For 2 matrices $A = [\mathbf{a}_1\ \mathbf{a}_2\ \cdots\ \mathbf{a}_n], B = [\mathbf{b}_1\ \mathbf{b}_2\ \cdots\ \mathbf{b}_n]$ of shape (m,n) & for $c \in \mathbb{K}$, define

$$A + B \coloneqq [\mathbf{a}_1 + \mathbf{b}_1\ \mathbf{a}_2 + \mathbf{b}_2\ \cdots\ \mathbf{a}_n + \mathbf{b}_n],$$
$$cA \coloneqq [c\mathbf{a}_1\ c\mathbf{a}_2\ \cdots\ c\mathbf{a}_n].$$

I.e., vector sum is defined by summing each pair of corresponding elements, & scalar multiple is defined by multiplying each element by same scalar. Matrix playing role of zero vector is one whose all elements are 0, called *zero matrix*. Matrix for inverse vector is obtained by inverting signs of all elements. (Do not call this matrix inverse matrix. This name will be used later for another meaning. Recall for a number x, inverse of x is $\frac{1}{x}$ instead of $-x$.)

In NumPy, sum & scalar multiple of matrices are expressed in usual way. Define `A, B` as arrays. If let `A, B` be lists, calculations later would not work.

```
A = array([[1 , 2 , 3], [4 , 5 , 6]])
B = array([[1 , 3 , 5], [2 , 4 , 6]])
A + B
print(2 * A, A / 2, A * B, 0 * A, -1 * A)
```

Calculate scalar multiple. Can calculate `A * 2, A / 2` too. Can calculate Â * B (Similar to matrix sum, this is a componentwise product. This product is called *Schular product* or *Hadamard product*. There are various products of matrices other than this.), but it is different from matrix product. Multiplying `A` by 0 gives zero matrix. Multiplying `A` by -1 gives matrix corresponding to inverse vector.

Problem 1. *Following program outputs a calculation problem of matrices in a form of a code in math mode of LaTeX. Typsetting obtained TeX code, obtained* Fig. 4.1: LaTeX output image with no margins using template.tex in Appendix. *Create another problem by changing seed* 2021 *& solve it.*

```
from numpy.random import seed , randint , choice
from sympy import Matrix , latex

seed(2021)
m, n = randint(2, 4, 2)
X = [-3, -2, -1, 1, 2, 3, 4, 5]
A = Matrix(choice(X, (m, n)))
B = Matrix(choice(X, (m, n)))
print(f'{latex(A)} + {latex(B)} = ')
```

```
\left[\begin{matrix}-2 & -3 & 3\\4 & 4 & 2\end{matrix}\right] + \left[\begin{matrix}5 & 4 & 1\\3 & 3 & 3
```

- 4.2. Matrices & Linear Mappings. For a given linear mapping $f : V \to W$, impossible to thoroughly investigate $\mathbf{y} \in W$ satisfying $\mathbf{y} = f(\mathbf{x})$, $\forall \mathbf{x} \in V$. (Even if V is finite dimensional, it has an infinite number of vectors.) Convenient if can express relation $\mathbf{y} = f(\mathbf{x})$ by a formula in $\mathbf{x}$ like $y = ax + b$ in case of usual numbers. Consider such a method.

  Let V, W be linear spaces over $\mathbb{K}$ of dimensions n, m & let $X = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}, Y = \{\mathbf{w}_1, \ldots, \mathbf{w}_m\}$ be their bases, resp. Because

- 5. Elementary Operations & Matrix Invariants.
- 6. Inner Product & Fourier Expansion.

- 7. Eigenvalues & Eigenvectors.
- 8. Jordan Normal Form & Spectrum.
- 9. Dynamical Systems.
- 10. Applications & Development of Linear Algebra.
- Appendix.

2 Wikipedia

2.1 Wikipedia/abstract structure

"An *abstract structure* is an abstraction that might be of the geometric spaces or a set structure, or a hypostatic abstraction that is defined by a set of mathematical theorems & laws, properties, & relationships in a way that is logically if not always historically independent[1] of the structure of contingent experiences, e.g., those involving physical objects. Abstract structures are studied not only in local & mathematics but in the fields that apply them, as computer science & computer graphics, & in the studies that reflect on them, such as philosophy (especially the philosophy of mathematics). Indeed, modern mathematics has been defined in a very general sense as the study of abstract structures (by the Bourbaki group: see discussion there, at algebraic structure & also structure).

An abstract structure may be represented (perhaps with some degree of approximation) by 1 or more physical objects – this is called an implementation or instantiation of the abstract structure. But the abstract structure itself is defined in a way that is not dependent on the properties of any particular implementation.

An abstract structure has a richer structure than a concept or an idea. An abstract structure must include precise rules of behavior which can be used to determine whether a candidate implementation actually matches the abstract structure in question, & it must be free from contradictions. Thus we may debate how well a particular government fits the concept of democracy, but there is no room for debate over whether a given sequence of moves is or is not a valid game of chess (e.g. Kasparovian approaches).

2.1.1 Examples

- A sorting algorithm is an abstract structure, but a recipe is not, because it depends on the properties & quantities of its ingredients.
- A simple melody is an abstract structure, but an orchestration is not, because it depends on the properties of particular instruments.
- Euclidean geometry is an abstract structure, but the theory of continent drift is not, because it depends on the geology of the Earth.
- A formal language is an abstract structure, but a natural language is not, because its rules of grammar & syntax are open to debate & interpretation.

2.2 Wikipedia/direct sum

"The *direct sum* is an operation between structures in abstract algebra, a branch of mathematics. It is defined differently, but analogously, for different kinds of structures. E.g., the direct sum of 2 abelian groups A, B is another abelian group $A \oplus B$ consisting of the ordered pairs (a, b) where $a \in A, b \in B$. To add ordered pairs, we define the sum $(a, b) + (c, d) \coloneqq (a + c, b + d)$, i.e., addition is defined coordinate-wise. E.g., the direct sum $\mathbb{R} \oplus \mathbb{R}$ where $\mathbb{R}$ is real coordinate space, is the Cartesian plane $\mathbb{R}^2$. A similar process can be used to form the direct sum of 2 vector spaces or 2 modules.

We can also form direct sums with any finite number of summands, e.g., $A \oplus B \oplus C$, provided A, B, C are the same kinds of algebraic structures (e.g., all abelian groups, or all vector spaces). This relies on the fact that the direct sum is associative up to isomorphism. I.e., $(A \oplus B) \oplus C \cong A \oplus (B \oplus C)$ for any algebraic structures A, B, C of the same kind. The direct sum is also commutative up to isomorphism, i.e., $A \oplus B \cong B \oplus A$ for any algebraic structures A, B of the same kind.

The direct sum of finitely many abelian groups, vector spaces, or modules is canonically isomorphic to the corresponding direct product. This is false, however, for some algebraic object, like nonabelian groups.

In the case where infinitely many objects are combined, the direct sum & direct product are not isomorphic, even for abelian groups, vector spaces, or modules. E.g., consider the direct sum & direct product of (countably) infinitely many copies of the integers. All element in the direct product is an infinite sequence, e.g., $(1, 2, 3, \ldots)$ but in the direct sum, there is a requirement that all but finitely many coordinates be zero, so the sequence $(1, 2, 3, \ldots)$ would be an element of the direct product but not of the direct sum, while $(1, 2, 0, 0, \ldots)$ would be an element of both. Often, if a + sign is used, all but finitely many coordinates must be zero, while if some form of multiplication is used, all but finitely many coordinates must be 1. In more technical language, if the summands are $(A_i)_{i \in I}$, the direct sum $\bigoplus_{i \in I} A_i$ is defined to be the set of tuples $(a_i)_{i \in I}$ with $a_i \in A_i$ s.t. $a_i = 0$ for all but

[1] However historical dependencies are partially considered in event theory as part of the combinatorics theory in Kolmogorov complexity & Kolmogorov-Khinchin equations.

finitely many i. The direct sum $\bigoplus_{i\in I} A_i$ is contained in the direct product $\prod_{i\in I} A_i$, but is strictly smaller when the index set I is infinite, because an element of the direct product can have infinitely many nonzero coordinates.

2.2.1 Examples

The xy-plane, a 2D vector space, can be thought of as the direct sum of 2 1D vector spaces, namely the x & y axes. In this direct sum, the x, y axes intersect only at the origin (the zero vector). Addition is defined coordinate-wise, i.e., $(x_1, y_1) + (x_2, y_2) \coloneqq (x_1 + x_2, y_1 + y_2)$, which is the same as vector addition.

Given 2 structures A, B, their direct sum is written as $A \oplus B$. Given an indexed family of structures A_i, indexed with $i \in I$, the direct sum may be written $A = \bigoplus_{i\in I} A_i$. Each A_i is called a *direct summand* of A. If the index set is finite, the direct sum is the same as the direct product. In the case of groups, if the group operation is written as $+$ the phrase "direct sum" is used, while if the group operation is written $*$ the phrase "direct product" is used. When the index set is infinite, the direct sum is not the same as the direct product since the direct sum has the extra requirement that all but finitely many coordinates must be 0.

Internal & external direct sums. A distinction is made between internal & external direct sums, though the 2 are isomorphic. If the summands are defined 1st, & then the direct sum is defined in terms of the summands, we have an external direct sum. E.g., if we define the real numbers $\mathbb{R}$ & then define $\mathbb{R} \oplus \mathbb{R}$ the direct sum is said to be *external*.

If, on the other hand, 1st define some algebraic structure S & then write S as a direct sum of 2 substructures V, W, then the direct sum is said to be internal. In this case, each element of S is expressible uniquely as an algebraic combination of an element of V & an element of W. For an example of an internal direct sum, consider $\mathbb{Z}_6$ (the integers modulo 6), whose elements are $\{0, 1, 2, 3, 4, 5\}$. This is expressible as an internal direct sum $\mathbb{Z}_6 = \{0, 2, 4\} \oplus \{0, 3\}$.

2.2.2 Types of direct sum

[...]

2.2.3 Homomorphisms

The direct sum $\bigoplus_{i\in I} A_i$ comes equipped with a *projection* homomorphism $\pi_j : \bigoplus_{i\in I} A_i \to A_j$ for each $j \in I$ & a *coprojection* $\alpha_j : A_j \to \bigoplus_{i\in I} A_i$ for each $j \in I$. Given another algebraic structure B (with the same additional structure) & homomorphisms $g_j : A_j \to B$, $\forall j \in I$, there is a unique homomorphism $g : \bigoplus_{i\in I} A_i \to B$, called the sum of the g_j, s.t. $g\alpha_j = g_j$, $\forall j$. Thus the direct sum is the coproduct in the appropriate category." – Wikipedia/direct sum

2.3 Wikipedia/mathematical structure

"In mathematics, a structure on a set (or on some sets) refers to providing it (or them) with certain additional features (e.g. an operation, relation, metric, or topology). The additional features are attached or related to the set (or to the sets), so as to provide it (or them) with some additional meaning or significance.

A partial list of possible structures are measures, algebraic structures (groups, fields, etc.), topologies, metric structures (geometries), orders, graphs, events, equivalence relations, differential structures, & categories.

Sometimes, a set is endowed with > 1 feature simultaneously, which allows mathematicians to study the interaction between the different structures more richly. E.g., an ordering imposes a rigid form, shape, or topology on the set, & if a set has both a topology feature & a group feature, s.t. these 2 features are related in a certain way, then the structure becomes a topological group.

Map between 2 sets with the same type of structure, which preserve this structure [morphism: structure in the domain is mapped properly to the (same type) structure in the codomain] is of special interest in many fields of mathematics. E.g.: homomorphisms, which preserve algebraic structures; continuous functions, which preserve topological structures; & differential functions, which preserve differential structures.

2.3.1 History

In 1939, the French group with the pseudonym Nicolas Bourbaki saw structures as the root of mathematics. They 1st mentioned them in their "Fascicule" of *Theory of Sets* & expanded it into Chap. IV of the 1957 edition. They identified 3 *mother structures*: algebraic, topological, & order.

2.3.2 Example: the real numbers

"The set of real numbers has several standard structures:

- An order: each number is either less than or greater than any other number.
- Algebraic structure: there are operations of addition & multiplication, the 1st of which makes it into a group & the pair of which together make it into a field.
- A measure: intervals of the real line have a specific length, which can be extended to the Lebesgue measure on many of its subsets.

- A metric: there is a notion of distance between points.
- A geometry: it is equipped with a metric & is flat.
- A topology: there is a notion of open sets.

There are interfaces among these:

- Its order &, independently, its metric structure induce its topology.
- Its order & algebraic structure make it into an ordered field.
- Its algebraic structure & topology make it into a Lie group, a type of topological group." – Wikipedia/mathematical structure

2.4 Wikipedia/numerical linear algebra

"*Numerical linear algebra*, sometimes called *applied linear algebra*, is the study of how matrix operations can be used to create compute algorithms which efficiently & accurately provide approximate answers to questions in continuous mathematics. It is a subfield of numerical analysis, & a type of linear algebra. Computers use floating-point arithmetic & cannot exactly represent irrational data, so when a computer is applied to a matrix of data, it can sometimes increase the difference between a number stored in the computer & the true number that it is an approximation of. Numerical linear algebra uses properties of vectors & matrices to develop computer algorithms that minimize the error introduced by the computer, & is also concerned with ensuring that the algorithm is as efficient as possible.

Numerical linear algebra aims to solve problems of continuous mathematics using finite precise computers, so its applications to the natural science & social sciences are as vast as the applications of continuous mathematics. It is often a fundamental part of engineering & computational science problems, e.g., image processing & signal processing, telecommunication, computational finance, materials science simulations, structural biology, data mining, bioinformatics, & fluid dynamics. Matrix methods are particularly used in FDMs, FEMs, & the modeling of differential equations. Noting the broad applications of numerical linear algebra, Lloyd N. Trefethen & David Bau III argue that it is "as fundamental to the mathematical sciences as calculus & differential equations", even though it is a comparatively small field. Because many properties of matrices & vectors also apply to functions & operators, numerical linear algebra can also be viewed as a type of functional analysis which has a particular emphasis on practical algorithms.

Common problems in numerical linear algebra include obtaining matrix decompositions like singular value decomposition, QR factorization, LU factorization, or eigendecomposition, which can then be used to answer common linear algebraic problems like solving linear systems of equations, locating eigenvalues, or least squares optimization. Numerical linear algebra's central concern with developing algorithms that do not introduce errors when applied to real data on a finite precision computer is often achieved by iterative methods rather than direct ones.

2.4.1 History

Numerical linear algebra was developed by computer pioneers like John von Neumann, Alan Turing, James H. Wilkinson, Alston Scott Householder, George Forsythe, & Heinz Rutishauser, in order to apply the earliest computers to problems in continuous mathematics, e.g. ballistics problems & the solutions to systems of PDEs. The 1st serious attempt to minimize computer error in the application of algorithms to real data is John von Neumann & Herman Goldstine's work in 1947. The field has grown as technology has increasingly enabled researchers to solve complex problems on extremely large high-precision matrices, & some numerical algorithms have grown in prominence as technologies like parallel computing have made them practical approaches to scientific problems.

2.4.2 Matrix decompositions

Main article: Wikipedia/matrix decomposition.

- **Partitioned matrices.** Main article: Wikipedia/block matrix. For many problems in applied linear algebra, it is useful to adopt the perspectives of a matrix as being a concatenation of column vectors. E.g., when solving the linear system $\mathbf{x} = A^{-1}\mathbf{b}$, rather than understanding $\mathbf{x}$ as the product A^{-1} with $\mathbf{b}$, it is helpful to think of $\mathbf{x}$ as the vector of coefficients in the linear expansion of $\mathbf{b}$ in the basis formed by the columns of A. Thinking of matrices as a concatenation of columns is also a practical approach for the purposes of matrix algorithms. This is because matrix algorithms frequently contain 2 nested loops: one over the columns of a matrix A, & another over the rows of A. E.g., for matrices $A^{m\times n}$ & vectors $x^{n\times 1}, y^{m\times 1}$, we could use the column partitioning perspective to compute $y \coloneqq Ax + y$ as

```
for q = 1:n
    for p = 1:m
        y(p) = A(p,q)*x(q) + y(p)
    end
end
```

- **Singular value decomposition.** Main article: Wikipedia/singular value decompostion. The singular value decomposition of a matrix $A^{m\times n}$ is $A = U\Sigma V^\star$ where U, V are unitary, & Σ is diagonal. The diagonal entries of Σ are called the singular values of A. Because singular values are the square roots of the eigenvalues of $AA^\star$, there is a tight connection between the singular value decomposition & eigenvalue decompositions, i.e., most methods for computing the singular value decomposition are similar to eigenvalue methods; perhaps the most common method involves Householder procedures.
- **QR factorization.** Main article: Wikipedia/QR decompostion. The QR factorization of a matrix $A^{m\times n}$ is a matrix $Q^{m\times m}$ & a matrix $R^{m\times n}$ so that $A = QR$, where Q is orthogonal & R is upper triangular. The 2 main algorithms for computing QR factorizations are the Gram–Schmidt process & the Householder transformation. The QR factorization is often used to solve linear least-squares problems, & eigenvalue problems (by way of the iterative QR algorithm).
- **LU factorization.** Main article: Wikipedia/LU decomposition. An LU factorization of a matrix A consists of a lower triangular matrix L & an upper triangular matrix U so that $A = LU$. The matrix U is found by an upper triangularization procedure which involves left-multiplying A by a series of matrices $M_1, \dots, M_{n-1}$ to form the product $M_{n-1}\cdots M_1 A = U$, so that equivalently $L = M_1^{-1}\cdots M_{n-1}^{-1}$.
- **Eigenvalue decomposition.** Main article: Wikipedia/eigendecomposition of a matrix. The eigenvalue decomposition of a matrix $A^{m\times m}$ is $A = X\Lambda X^{-1}$, where the columns of X are the eigenvectors of A, & Λ is a diagonal matrix the diagonal entries of which are the corresponding eigenvalues of A. There is no direct method for finding the eigenvalue decomposition of an arbitrary matrix. Because it is not possible to write a program that finds the exact roots of an arbitrary polynomial in finite time, any general eigenvalue solver must necessarily be iterative.

2.4.3 Algorithms

1. **Gaussian elimination.**
2. **Solutions of linear systems.**
3. **Least squares optimization.**

2.4.4 Conditioning & stability

2.4.5 Iterative methods

2.4.6 Software

Main article: Wikipedia/list of numerical analysis software. Several programming languages use numerical linear algebra optimization techniques & are designed to implement numerical linear algebra algorithms. These languages include MATLAB, Analytica, Maple, & Mathematica. Other programming languages which are not explicitly designed for numerical linear algebra have libraries that provide numerical linear algebra routines & optimization; C & Fortran have packages like Basic Linear Algebra Subprograms & LAPACK, Python has the library NumPy, & Perl has the Perl Data Language. Many numerical linear algebra commands in R rely on these more fundamental libraries like LAPACK. More libraries can be found on the list of numerical libraries." – Wikipedia/numerical linear algebra

3 Miscellaneous

Tài liệu

[Hưn22] Nguyễn Hữu Việt Hưng. *Đại Số Tuyến Tính*. Tái bản lần thứ 4. Nhà Xuất Bản Đại Học Quốc Gia Hà Nội, 2022, p. 335.

[TB22] Lloyd N. Trefethen and David Bau III. *Numerical linear algebra*. 25th anniversary edition [of 1444820], With a foreword by James G. Nagy. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, [2022] ©2022, pp. xvi+370. ISBN: 978-1-611977-15-8; [9781611977165].

[TB97] Lloyd N. Trefethen and David Bau III. *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997, pp. xii+361. ISBN: 0-89871-361-7. DOI: 10.1137/1.9780898719574. URL: https://doi.org/10.1137/1.9780898719574.

[Tiệ25] Vũ Khắc Tiệp. *Machine Learning Cơ Bản*. 2025, p. 422.

[Tsu+23] Makoto Tsukada, Yuji Kobayashi, Hiroshi Kaneko, Sin-Ei Takahasi, Kiyoshi Shirayanagi, and Masato Noguchi. *Linear Algebra with Python: Theory and Applications*. Springer Undergraduate Texts in Mathematics and Technology. Springer, 2023, p. 324.