Mathematical Optimization – Toán Tối Ưu

Nguyễn Quản Bá Hồng*

Ngày 27 tháng 3 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:

- *Mathematical Optimization – Toán Tối Ưu.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/optimization/NQBH_mathematical_optimization.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/optimization/NQBH_mathematical_optimization.tex.

Mục lục

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

1 Adjoint

2 main types: discrete adjoint, continuous adjoint.

Resources – Tài nguyên.

1. Tan Bui-Thanh. Adjoint & Its roles in Sciences, Engineering, and Mathematics: A Tutorial.

Abstract. Synergize roles of adjoint in various disciplines of mathematics, sciences, & engineering. Though materials developed & presented here are not new – as each or some could be found in (or inferred from) publications in different fields – believe: 1st effort to systematically unify these materials on the same mathematical footing starting from the basic defs. Aim: provide a unified perspective & understanding of adjoint applications. As a result, this work could give broader views & better insights into the application of adjoint beyond a single community. By rigorously establishing general results & then developing materials specific to each application, bring forth details on how abstract concepts/defs can be translated into particular applications & connections among them. This paper is written as an interdisciplinary tutorial on adjoint with discussions & with many examples from different fields including linear algebra (e.g., eigendecomposition & SVD), ODEs (asymptotic stability of an epidemic model), PDEs (well-posedness of elliptic, hyperbolic, & Friedrichs' types), neural networks (backpropagation of feed-forward deep neural networks), least squares & inverse problems (with Tikhonov regularization), & PDE-constrained optimization. Exposition covers results & applications in both finite-dimensional & infinite-dimensional Hilbert spaces.

Keywords.Adjoint, optimization, backpropagation, eigenvalue problem, singular value decomposition, asymptotic stability, wellposedness, least squares, PDE-constrained optimization, reproduction number.

- 1. Introduction. Adjoint is ubiquitous in mathematics. History of adjoint can be traced back to as far as Lagrange in 1766, in a memoir extending the letter that he wrote to D'Alembert in Jun 1765 discussing his new method of solving nth-order differential equations. Term "adjoint equation", 1st used to call corresponding equation developed from Lagrange memoir, is due to Fuchs. Adjoint operator was introduced by Riesz in his seminal paper (Riesz also established *functional analysis* as a new mathematical discipline) to study inverse of linear operators.

  Since then adjoint has been pervasive in vast literature across mathematics, engineering, & sciences disciplines. This is not surprising as adjoint has many appealing features including:

  (a) adjoint operator typically possesses nicer properties than the original operator (e.g. adjoint of a densely defined linear operator is a closed operator though the original operator may not)

  (b) adjoint equation is always linear even when the original equation is not.

  Though a comprehensive survey on adjoint accounting for its application in many disciplines/fields (& their sub-disciplines) could be desirable to appreciate crucial role that adjoint plays, it is perhaps an impossible task. Or more precisely, it is rather the task for a book than for a paper.

  Goal 1. *Provide a window into the adjoint & its crucial role in certain subsets of computational science, engineering, & mathematics.*

  Exposition is necessarily personal & biased based on topics familiar with. Materials developed & presented are not new, as each or some could be found in (or inferred from) publications in different fields.

  Goal 2. *Systematically unify these materials on the same mathematical footing starting from basic defs.*

  This expectantly provides a more unified perspective on usefulness of adjoint in variety of applications. As a result, this work could give broader views & better insights into applications of adjoint beyond 1 field. By establishing general results & then developing materials specific to each application, bring forth details on how abstract concepts/defs can be translated into particular applications & connections among them. This paper is written as a tutorial on adjoint with many examples presented with discussions. Though trike for a self-contained expositor, necessary to state a few results without proof to keep length of paper manageable & to focus on adjoint & its roles.

  Structure.

  ◦ Sect. 2: Introduce various notations, defs, & some examples.

- Sect. 3: Part I in finite dimensions. Start with celebrated Riesz representation theorem that is then used to prove existence of adjoints of continuous linear operators. Followed by closed range theorem that will be useful in many places later. Built upon these basic materials, shall develop several applications of adjoint.
  - Subsect. 3.1: highlight role of adjoint in assessing solvability of linear operator equations before solving them.
  - Subsect. 3.2: 1 of most important applications of adjoint is in study of eigenvalue problems. Main result: spectral decomposition of self-adjoiont operators in finite dimensions. Tight relationship between orthogonality of a projection & its self-adjointness, immediately leading to a generalized Pythagorean theorem.
  - Subsect. 3.3: start with classical projection theorem & then deploy it together with close range theorem to find necessary & sufficient condition for optimality of an abstract linear least squares problem.
  - Subsect. 3.4: Next important application: SVD, in which employ spectral decomposition to establish an SVD decomposition for general linear operator in finite dimensions. This SVD decomposition is then deployed to provide trivial proofs for closed range theorem, rank-nullity theorem, & fundamental theorem of linear algebra for abstract linear operators. Discuss equivalence of SVD of an abstract linear operator & SVD of its matrix representation.
  - Subsect. 3.5: optimization with equality constraints. Expose at length role of adjoint in optimization theory valid for both finite & infinite dimensions. Accomplished by working with Fréchet derivative & its Riesz representation counterpart as the gradient. Though can be further developed (e.g. to 2nd-order optimality conditions) focus on 1st-order optimality condition. Recall an implicit function theorem & use it to prove an abstract inverse function theorem, then deployed to derive 1st-order optimality condition for abstract optimization problems with equality constraints. Important role of adjoint comes into picture when prove an abstract Lagrangian multiplier theorem using closed range theorem. Importance of adjoint is further amplified when study constrained optimization problems with separable structure. Here, adjoint facilitates an efficient gradient-based optimization algorithm in unconstrained reduced subspace while ensuring feasibility of constraints at all time.
  - Subsect. 3.6: show that when applying this reduced space approach for optimization problem arisen from training deep neural networks (DNNs), recover backpropagation from reduced space approach provides further insights into algorithm.
  - Subsect. 3.7: stability of autonomous ODEs. Main goal: exhibit vital role of adjoint in establishing necessary & sufficient conditions for asymptotic stability (in sense of Lyapunov) of ODEs' equilibria.
- Sect. 4: Part II in infinite dimensions. Start with a more general adjoint definition for densely defined linear operators. Useful for all subsects. except Subsect. 4.1.
  - Subsect. 4.1: illposedness (in Hadamard's sense) nature of inverting compact operators. Extend spectral theorem & SVD decomposition theorem to Hilbert-Schmidt theorem & a general SVD theorem for compact (linear) operators in infinite dimensions. Main result: a Picard theorem stating necessary & sufficient conditions under which task of inverting a compact operator is solvable. Often employed to show that inverting a compact operator violates at least stability condition of well-posedness. Deploy Riesz-Fredholm theory to show how Tikhonov regularization can restore well-posedness at expense of getting a nearby solution.
  - Subsect. 4.2: Discuss role of adjoint in establishing well-posedness of abstract linear operator equations with application to PDEs. 2 main results developed: Banach-Nečas-Babuška theorem, Lax-Milgram lemma.
  - Subsect. 4.3: study Sturm-Liouville eigenvalue problem & generalized Fourier series in L^2. For closed linear operators. Using Hilbert-Schmidt theorem & Lax-Milgram lemma, establish a spectral decomposition theorems for quite general linear operators, & then apply them to Sturm-Liouville eigenvalue problems to derive Fourier series & its generalizations.
  - Subsect. 4.4: PDE-constrained optimization. Show how to rigorously translate abstract Lagrangian multiplier theorem to derive adjoint equation & reduced gradient for prototype elliptic & hyperbolic PDEs. Show: differential operators of adjoint equations are indeed adjoint operators derived at beginning of Sect. 4.

To keep exposition succinct, defs & results valid for both cases are presented/proved once & when that happens we will explicitly state so. Most of our developments start from abstract operator settings & then reduce to standard finite-dimensional settings in $\mathbb{R}^d$ as a special case. In some cases, e.g. optimization, order is reversed as we believe it is more natural that way. Each sect. of paper is equipped with examples on which we show how to apply preceding abstract theoretical results. Make an effort to include practical examples from different fields including linear algebra (e.g., eigendecomposition & SVD), ODEs (an epidemic model), PDEs (of elliptic, hyperbolic, & Friedrichs' types), neural networks (feed-forward deep neural networks), least squares & inverse problems (with Tikhonov regularization), PDE-constrained optimization, etc. Due to interdisciplinary nature of paper, do not attempt to provide an exhaustive list of refs but a few for each sect. to keep references at a manageable length.

- 2. Notations. boldface lower case letters, e.g., $\mathbf{u}$: vector-valued functions in $\mathbb{R}^d$. Calligraphic uppercase letters, e.g., $\mathcal{A}$: matrices, script uppercase letters: operators. Bold blackboard upper cases: spaces & sets. Lowercase letters: scalar-valued functions, also for results valid for both finite & infinite dimensional settings. Bold uppercase letters: bases of vector spaces. Frequently identify dual of any Hilbert space with itself. θ denotes either "zero" function or "zero" vector in appropriate space.

  Defs: Linear transformation/map/operator & its domain, range, kernel/null space. Any matrix is a linear operator. Integrals are operators: $\mathcal{A}f = \int_0^1 \omega(t) f(t)\,\mathrm{d}t$. Differentiation is a linear operator: $\mathcal{A}u = \frac{d^2}{dt^2}u(t)$.
- 3. Part I: Adjoint operators in finite dimensional Hilbert spaces. Assume X, Y are finite dimensional vector spaces, $\dim X = n < \infty, \dim Y = m < \infty$. If $A \in L(X,Y)$ & $\dim X < \infty$, then $A \in \mathcal{B}(X,Y)$. Let $E = \{e_i\}_{i=1}^n, G = \{g_i\}_{i=1}^m$ be orthonormal

bases for X, Y, resp. $\forall u \in X$, denote by u^E the unique vector of coordinates of u in E, then $(u^E, v^E)_{\mathbb{F}^d} = (u, v)_X$. Matrix representation of A w.r.t. bases E, G is denoted as A^{EG}. When there is no ambiguity on bases referred to, simply ignore superscripts for both coordinate vector & matrix representation. Denote ith element of a vector $\mathbf{u}$ as $\mathbf{u}(i)$ & element at ith row & jth column of a matrix A as $A(i, j)$. Also use $\mathbf{u}_i \coloneqq \mathbf{u}(i)$. Use square brackets to express matrices & vectors with a finite number of components. Unless otherwise stated, vectors with finite number of components are column vectors. Organize: celebrated Riesz representation theorem & closed range theorem, upon which develop several applications of adjoint.

Theorem 1 (Riesz representation)**.** *Let L be a bounded linear functional on a Hilbert space X. There exists a unique $u \in X$ s.t. $L(v) = (u, v)_X$, $\forall v \in X$. Furthermore, operator norm of L is given as $\|L\| \coloneqq \sup_{v \in X} \frac{|L(v)|}{\|v\|_X} = \|u\|_X$.*

Definition 1 (Adjoint operator)**.** *Let $A \in B(X, Y)$. Say that $A^\star : Y \to X$ is the adjoint of A iff $(Au, v)_Y = (u, A^\star v)_X$, $\forall u \in X, v \in Y$.*

Proposition 1. *Let $A \in B(X, Y)$. Then $A^\star$ exists & is unique. Furthermore, it is linear with $\|A^\star\| = \|A\|$, where operator norm is defined as $\|A\| \coloneqq \sup_{u \in X} \frac{\|Au\|_Y}{\|u\|_X} = \sup_{\|u\|_X = 1} \|Au\|_Y$.*

An M-weighted inner product $(\cdot, \cdot)_{\mathbb{R}^d, M}$ where $(\mathbf{u}, \mathbf{v})_{\mathbb{R}^d, M} \coloneqq \sum_{i,j} \mathbf{u}(i) M(i, j) \mathbf{v}(i) \coloneqq \mathbf{u}^T M \mathbf{v}$, $\forall \mathbf{u}, \mathbf{v} \in V$, M: a symmetric & positive definite matrix. Adjoint operator $A^\star = A^T M$ of a matrix A. Set $\mathbb{P}^n[0, 1]$ of complex-valued polynomial of order $\leq n$ on $[0, 1]$.

Proposition 2. *Let E, G be orthonormal bases of X, Y, resp., & $\dim X = n, \dim Y = m$. Let A, B be the matrix representations of $A, A^\star$ w.r.t. bases E, G. Then $B = A^\star$ where $A^\star$ be the conjugate transpose of A.*

Orthogonal complement $S^\perp$ of $S \subset X$ is a closed subspace of X & $S \cap S^\perp = \{0\}$. Close $\overline{S}$ of S is the smallest closed set containing S. $(S^\perp)^\perp = \overline{S}$.

Theorem 2 (Closed range)**.** *Let $A : X \to Y$. $[\mathrm{R}(A)]^\perp = \mathrm{N}A^\star, \overline{\mathrm{R}(A)} = [N(A^\star)]^\perp, [\mathrm{R}(A^\star)]^\perp = \mathrm{N}(A), \overline{\mathrm{R}(A^\star)} = [\mathrm{N}(A)]^\perp$. Consequently, $X = \mathrm{N}(A) \oplus \overline{\mathrm{R}(A^\star)}$, $Y = \mathrm{N}(A^\star) \oplus \overline{\mathrm{R}(A)}$.*

Since consider only Hilbert spaces, which are reflexive, $\overline{\mathrm{R}(A^\star)} = [\mathrm{N}(A)]^\perp$ holds. In general $\overline{\mathrm{R}(A^\star)} \subset [\mathrm{N}(A)]^\perp$. For finite dimensional vector spaces X, Y, $\mathrm{R}|(A)$, & hence $\mathrm{R}(A^\star)$, is obviously closed. Proof of closed range theorem for finite dimensional cases is trivial using SVD decomposition.

- Subsect. 3.1: Application of adjoint to solvability of linear operator equations. Solvability of linear operator equations before solving them. Existence of a solution of linear operator equation $Au = f$ where $A : X \to Y$.

  Lemma 1.

  (i) Existence: The linear equation $Au = f$ has a solution iff $y \in \mathrm{N}(A^\star)^\perp$.

  (ii) Uniqueness: The solution of the linear equation $Au = f$ is unique iff $\mathrm{N}(A) = \{0\}$.

  (iii) If $\dim X = \dim Y$, uniqueness $\Leftrightarrow$ existence.

  Existence condition can be simply $y \in \mathrm{R}(A)$. However, easier to work with $\mathrm{N}(A^\star)^\perp$ as it gives us equations to determine/characterize $\mathrm{N}(A^\star)^\perp$.

  An operator setting for problem of fitting a quadratic polynomial $u(x)$ with 2 pieces of information about $u(x)$.
- Subsect. 3.2: Application of adjoint to eigenvalue problems. Role of adjoint in study of eigenvalue problems.

  Definition 2 (Eigenvalue problem)**.** *Let $A : X \to X$ be a linear operator. $Au = \lambda u$ is called an* eigenvalue problem *if there exists a nontrivial pair (λ, x) (x is not a zero vector/function but λ could be zero). In particular: λ is called an* eigenvalue*. x is called an* eigenfunction*, associated with eigenvalue λ, of A. If X is a finite-dimensional space, i.e., $X = \mathbb{R}^d$, x is typically called* eigenvector*.*

  Definition 3 (Self-adjoint operator)**.** *If $A^\star = A$, then A is called* self-adjoint*.*

  Lemma 2. *Let $A : X \to X$ be a linear operator & A is self-adjoint. Then:*

  (i) Eigenvalues of A are real.

  (ii) Eigenfunctions corresponding to distinct eigenvalues are orthogonal to each other. I.e., if (λ, u) & (α, v) are 2 eigenpairs & $\lambda \neq \alpha$ then $(u, v)_X = 0$.

  Proposition 3. *Let $A : X \to X$ be a linear operator. Then A has at least 1 eigenvalue.*

  Theorem 3. *Let $A : X \to X$ be a self-adjoint linear operator. Then, an orthonormal basis of X can be constructed from eigenfunctions of A.*

  Theorem 4 (Spectral decomposition of self-adjoint operators in finite dimensions)**.** *Let $\dim X = n$ & $A : X \to X$ be a linear & self-adjoint operator. There exists n real values $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ & orthonormal vectors $u_1, u_2, \ldots, u_n$ s.t. $Au_i = \lambda_i u_i$. $\forall x \in X$, $Ax = \sum_{i=1} \lambda_i (x, u_i)_X u_i \Rightarrow A = \sum_{i=1}^n \lambda_i (\cdot, u_i)_X u_i$, i.e., A is completely determined by its eigenpairs.*
- Subsect. 3.3. Employ classical projection theorem + closed range theorem to find necessary & sufficient condition for optimality of an abstract linear least squares problem.
- Subsect. 3.4. Singular value decomposition (SVD). Deploy SVD decomposition to provide trivial proofs for closed range theorem, rank-nullity theorem, & fundamental theorem of linear algebra for abstract linear operators.
- Subsect. 3.5. Optimization with equality constraints, expose at length role of adjoint in optimization theory valid for both finite & infinite dimensions.

- Subsect. 3.6: Application of adjoint to backpropagation in deep learning. A reduced spaced approach using adjoint reduces to backpropagation of deep neural networks. Consider standard fully connected deep neural network & use adjoint method to derive backpropagation method for computing gradient of loss function w.r.t. weights & biases of a general fully-connected deep neural work (DNN). Review papers on deep learning [LBH15], [69], history of backpropagation: [54,55]. Gradient is needed for gradient-based methods e.g. stochastic gradient descent. Extension of adjoint method for other type of neural networks e.g. ResNet & CNN are straightforward. Backpropagation is nothing more than a reduced space approach to compute gradient using gradient method.

  Definition 4 (L-layer Neural network)**.** *Given $n_l, s_0, s_1, \ldots, s_{n_l} \in \mathbb{N}$, an n_l-layer neural network is defined as the following series of composition: Input layer: $\mathbf{a}^0 - \mathbf{x} = \mathbf{0}$, The ith layer: $\mathbf{a}^i - \sigma(\mathcal{W}^i \mathbf{a}^{i-1} + \mathbf{b}^i) = \mathbf{0}$, $i = 1, \ldots, n_l$, where $\mathbf{x} \in \mathbb{R}^{s_0}, \mathcal{W}^i \in \mathbb{R}^{s_i} \times \mathbb{R}^{s_i - 1}$, $\mathbf{b}^i \in \mathbb{R}^{s_i}$, $i = 1, \ldots, n_l$, are weight matrix & bias vector of the ith layer; $\mathbf{a}^i \in \mathbb{R}^{s_i}$ is the output of the ith layer; & the activation function σ acts componentwise when its argument is a vector.*

  Define $\mathbf{u} := [\mathbf{a}^0, \ldots, \mathbf{a}^{n_l}]^T, \mathbf{z} := [\mathcal{W}^1, \mathbf{b}^1, \ldots, W^{n_l}, \mathbf{b}^{n_l}]^T, \mathbf{c}(\mathbf{u}, \mathbf{z}) = \mathbf{0}$ as concatenation of all subequation. For concreteness, consider loss (objective) function $f(\mathbf{u}, \mathbf{z}) := \frac{1}{2}\|\mathbf{a}^{\text{obs}} - \mathbf{a}^{n_l}\|^2$ where $\mathbf{a}^{\text{obs}}$ is a given data (label). Neural network training problem is constrained optimization problem in Ex 3.59.
  Using adjoint equations, backpropagate to compute adjoint solution from output layer to ith layer, & then compute gradients. From backpropagation point of view, $\mathbf{y}^i, i = 1, \ldots, n_l$ are simply temporary variables to help compute/write chain rule in a succinct manner. The adjoint approach, however, reveals their precise role as adjoint solutions – also known as Lagrangian multiliers – of adjoint equations stemming from 1st-order optimality condition using reduced space approach. DNN training problem, from adjoint point of view, is a constrained optimization problem with forward pass as forward equations. The backpropagation is thus nothing more than a reduced space approach to compute gradient using adjoint method.
- Subsect. 3.7. role of adjoint in establishing stability of autonomous ODEs. A brief view of role of adjoint in study of stability of equilibria of ODEs. Limit to autonomous systems of form $\dot{\mathbf{x}} := \frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^d, \mathbf{f} : G \subset \mathbb{R}^d \to \mathbb{R}^d$ is assumed to be continuous & locally Lipschitz.

  Definition 5 (Lyapunov stability)**.** *The equilibrium point $\mathbf{0}$ is stable in the sense of Lyapunov if for any $\varepsilon > 0, \exists \delta > 0$ s.t. for every (maximal) solution $\mathbf{x} : I \to G$ s.t. $\mathbf{x}(0) \le \delta$, have $\mathbf{x}(t) \le \varepsilon$, $\forall t \in I \cap (0, \infty)$.*

  Theorem 5 (Lyapunov direct method)**.** *If there exists an open neighborhood U of $\mathbf{0}$ & a continuous differentiable function V s.t.*
  (i) $V(\mathbf{0}) = 0$, $V(\mathbf{z}) > 0$, $\forall \mathbf{z} \in U \backslash \{\mathbf{0}\}$, &
  (ii) $V_{\mathbf{f}}(\mathbf{z}) := (\nabla V(\mathbf{z}), \mathbf{f}(\mathbf{z})) := \sum_{i=1}^n \partial_{\mathbf{z}_i} V \mathbf{f}_i(\mathbf{z}) \le 0$, $\forall \mathbf{z} \in U$.
  Then $\mathbf{0}$ is a stable equilibrium point of $\dot{\mathbf{x}} := \frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x})$.

  Definition 6 (Asymptotic stability)**.** *The equilibrium $\mathbf{0}$ is* attractive *if there exists $\delta > 0$ s.t. $\forall \mathbf{x}_0 \in G$ s.t. $\|\mathbf{x}_0\| \le \delta$, then the solution $\mathbf{x}(t) \to \mathbf{0}$ as $t \to \infty$. Say $\mathbf{0}$* asymptotically stable *in the sense of Lyapunov if it is both stable & attractive.*

  Theorem 6 (A sufficient condition for asymptotic stability)**.** *Assume that there exists a neighborhood U of $\mathbf{x}_0$ & a continuously differentiable function V s.t.*
  (i) $V(\mathbf{0}) = 0$, $V(\mathbf{z}) > 0$, $\forall \mathbf{z} \in U \backslash \{\mathbf{0}\}$, & $V_{\mathbf{f}} \le 0$, $\forall \mathbf{z} \in U$, &
  (ii) $\mathbf{0}$ is the inverse image of $V_{\mathbf{f}}(\mathbf{z}) = 0$, i.e., $V_{\mathbf{f}}^{-1}(0) = \mathbf{0}$.
  Then $\mathbf{0}$ is asymptotically stable.

  Study stability of systems of linear ODEs, & this is where the adjoint comes into picture. Infer stability of nonlinear systems using stability of their linearizations.

- 4. Part II: Adjoint operators in Infinite dimensional Hilbert spaces.
- 5. Conclusions.

2 Algorithms for Optimization – Thuật Toán Tối Ưu

2.1 [KW19]. Mykel J. Kochenderfer, Tim A. Wheeler. Algorithms for Optimization. 2019

[118 Amazon ratings]

Amazon review. A comprehensive introduction to optimization with a focus on practical algorithms for design of engineering systems.

This book offers a comprehensive introduction to optimization with a focus on practical algorithms. Book approaches optimization from an engineering perspective, where objective: design a system that optimizes a set of metrics subject to constraints. Readers will learn about computational approaches for a range of challenges, including searching high-dimensional spaces, handling problems where there are multiple competing objectives, & accommodating uncertainty in metrics. Figures, examples, & exercises convey intuition behind mathematical approaches. Text provides concrete implementation in Julia programming language.

Topics covered include derivatives & their generalization to multiple dimensions; local descent & 1st- & 2nd-order methods that inform local descent; stochastic methods, which introduce randomness into optimization process; linear constrained optimization, when both objective function & constraints are linear; surrogate models, probabilistic surrogate models, & using probabilistic surrogate models to guide optimization; expression optimization, & multidisciplinary design optimization. Appendixes offer an introduction to Julia language, test functions for evaluating algorithm performance, & mathematical concepts used in derivation

& analysis of optimization methods discussed in text. Book can be used by advanced undergraduates & graduates students in mathematics, statistics, CS, any engineering field, (including electrical engineering & aerospace engineering), & operations research, & as a reference for professionals.

- Preface. This book provides a broad introduction to optimization with a focus on practical algorithms for design of engineering systems. Cover a wide variety of optimization topics, introducing underlying mathematical problem formulations & algorithms for solving them. Figures, examples, & exercises are provided to convey intuition behind various approaches.

  This text is intended for advanced undergraduates & graduate students as well as professionals. Book requires some mathematical maturity & assumes prior exposure to multivariable calculus, linear algebra, & probability concepts. Some review material is provided in appendix. Disciplines where book would be especially useful include mathematics, statistics, CS, aerospace, electrical engineering, & operations research.

  Fundamental to this textbook are algorithms, which are all implemented in Julia programming language. Have found language to be ideal for specifying algorithms in human readable form. Permission is granted, free of charge, to use code snippets associated with this book, subject to condition: source of code is acknowledged. Anticipate (Dự đoán): others may want to contribute translations of these algorithms to other programming languages. As translations become available, link to them from book's webpage.

- 1. Introduction.
- 2. Derivatives & Gradients.
- 3. Bracketing.
- 4. Local Descent.
- 5. 1st-Order Methods.
- 6. 2nd-Order Methods.
- 7. Direct Methods.
- 8. Stochastic Methods.
- 9. Population Methods.
- 10. Constraints.
- 11. Linear Constrained Optimization.
- 12. Multiobjective Optimization.
- 13. Sampling Plans.
- 14. Surrogate Models.
- 15. Probabilistic Surrogate Models.
- 16. Surrogate Optimization.
- 17. Optimization under Uncertainty.
- 18. Uncertainty Propagation.
- 19. Discrete Optimization.
- 20. Expression Optimization.
- 21. Multidisciplinary Optimization.
- A. Julia.
- B. Test Functions.
- C. Mathematical Concepts.

3 Mathematical Optimization

Community – Cộng đồng. Marc Quincampoix, Cédric Villani.
Resources – Tài nguyên.

1. [Vil03]. Cédric Villani. *Topics in Optimal Transportation.*

- Preface. Optimal mass transportation was born in France in 1781, with a very famous paper by Gaspard Monge, *Mémoire sur la théorie des déblais et des remblais.* Since, become a classical subject in probability theory, economics, & optimization. Gained extreme popularity, since many researches in different areas of mathematics understood that optimal mass transportation was strongly linked to their subject. A precise birthdate for this revival: 1987 note by Yann Brenier, *Décomposition polaire et réarrangement des champs de vecteurs* paved the way towards a beautiful interplay between PDEs, fluid mechanics, geometry, probability theory, & function analysis, developed over last 10 years, through contributions of a number of authors, with optimal transportation problems as a common denominator. 2 volumes of *Mass transportation problems* by Rachev & Rüschendorf depicting many applications of Monge-Kantorovich distance to various problems, together with classical theory of optimal transportation problem in a very abstract setting; survey by Evans, which can also be considered as an introduction to subject, describes several applications of L^1 theory (i.e., when cost function is a distance), extremely clear lecture notes by Ambrosio, centered on L^1 theory from point of view of calculus of variations, lecture notes by Urbas – a marvelous reference for regularity theory of Monge-Ampère equation arising in mass transportation.
- Introduction. *Formulation of optimal transportation problem.* Assume given a pile of sand, & a hole having to be completely filled up with the sand. Obviously pile & hole must have same volume. Normalize mass of pile = 1. Will model both pile & hole by probability measures μ, ν, defined respectively on some measure spaces X, Y. Whenever A, B: measurable subsets of X, Y, resp., $\mu[A]$ gives a measure of how much sand is located inside A, $\nu[B]$ of how much sand can be piled in B. Moving sand around needs some effort, modeled by a measurable *cost function* defined on $X \times Y$. Informally, $c(x, y)$ tells how much it costs to transport 1 unit of mass from location x to location y. Natural to assume at least: c is measurable & nonnegative. Should not a priori exclude the possibility that c takes infinite values, & so c should be a measurable map from $X \times Y$ to $\mathbb{R} \cup \{+\infty\}$.

  Problem 1 (Central question)**.** *How to realize transportation at minimal cost?*

  Before studying this question, have to make clear what a way of transportation, or a *transference*[1] *plan*, is. Will model transference plans by probability measures π on the product space $X \times Y$. Informally, $d\pi(x, y)$ measures amount of mass transferred from location x to location y. Do not a priori exclude the possibility that some mass located at point x may be split into several parts (several possible destination y's). For a transference plan $\pi \in P(X \times Y)$ to be admissible, of course necessary that all the mass taken from point x coincide with $d\mu(x)$, & all the mass transferred to y coincide with $d\nu(y)$, i.e., $\int_Y d\pi(x, y) = d\mu(x)$, $\int_X d\pi(x, y) = d\nu(y)$. More rigorously, require $\pi[A \times Y] = \mu[A]$, $\pi[X \times B] = \nu[B]$ for all measurable subsets $A \subset X$ & $B \subset Y$. Equivalent to stating: for all functions ϕ, ψ in a suitable class of test functions,

  $$\int_{X \times Y} \phi(x) + \psi(y)\, \mathrm{d}\pi(x, y) = \int_X \phi(x)\, \mathrm{d}\mu(x) + \int_Y \psi(y)\, \mathrm{d}\nu(y). \tag{1}$$

  In general, the natural set of admissible test functions for (ϕ, ψ) is $L^1(d\mu) \times L^1(d\nu)$, or equivalently, $L^\infty(d\mu) \times L^\infty(d\nu)$. In most situations of interest, this class can be narrowed to just $C_b(X) \times C_b(Y)$, or $C_0(X) \times C_0(Y)$. Those probability measures π satisfying $\pi[A \times Y] = \mu[A]$, $\pi[X \times B] = \nu[B]$ are said to have *marginals* μ, ν, & will be the admissible transference plans. Denote the set of all such probability measures by $\Pi(\mu, \nu) := \{\pi \in P(X \times Y); \pi[A \times Y] = \mu[A], \pi[X \times B] = \nu[B]$ holds for all measurable $A, B\} \neq \emptyset$ since tensor product $\mu \otimes \nu \in \Pi(\mu, \nu)$, corresponding to the most stupid transportation plan that one may imagine: any piece of sand, regardless of its location, is distributed over the entire hole, proportionally to the depth. A clear mathematical def of basic problem:

  Problem 2 (Kantorovich's optimal transportation problem)**.** *Minimize* $I[\pi] = \int_{X \times Y} c(x, y)\, \mathrm{d}\pi(x, y)$ *for* $\pi \in \Pi(\mu, \nu)$.

  This minimization problem was studied in 40s by Kantorovich awarded a Nobel prize for related work in economics. The optimal transference problem is related to basic questions in economics becomes clear if one thinks of μ as a density of production units, & of ν as aa density of consumers. For a given transference plan π, the nonnegative (possibly infinite) quantity $I[\pi]$ is called the *total transportation cost* associated to π. The *optimal transportation cost* between μ, ν is the value $\mathcal{T}_c(\mu, \nu) := \inf_{\pi \in \Pi(\mu, \nu)} I[\pi]$. The optimal π's, i.e., those s.t. $I[\pi] = \mathcal{T}_c(\mu, \nu)$, if they exist, will be called *optimal transference plans.*

  Translate Kantorovich problem into its probabilistic equivalent:

  Problem 3 (Probabilistic interpretation)**.** *Given 2 probability measures* μ, ν*, minimize expectation* $I(U, V) = \mathbb{E}[c(U, V)]$ *over all pairs* (U, V) *of random variables* U *in* X*,* V *in* Y*, s.t.* $\mathrm{law}(U) = \mu$*,* $\mathrm{law}(V) = \nu$*.*

  Basic probabilistic theory. a random variable U in X is a measurable map with values in X, defined on a probability space Ω equipped with a probability measure $\mathbb{P}$, that the law of U is the probability measure μ on X defined by $\mu[A] := \mathbb{P}[U^{-1}(A)]$, & the expectation stands for the integral w.r.t. $\mathbb{P}$. Transference plans $\pi \in \Pi(\mu, \nu)$ are all possible laws of the couple (U, V). Such a π is often said to be the *joint law* of random variables U, V; also says that it constitutes a *coupling* of U, V.

[1] the process of moving something from one place, person or use to another.

Kantorovich's problem is a relaxed version of the original mass transportation problem considered by Monge. Monge's problem is just the same as Kantorovich's, except for 1 thing: additionally required: *no mass be split*. I.e., to each location x is associated a unique destination y. In terms of random variables, this requirement means that we ask for V to be a function of U in $I(U,V) = \mathbb{E}[c(U,V)]$. In terms of transference plans, it means that we ask for π in $I[\pi]$ to have special form $d\pi(x,y) = d\pi_T(x,y) \equiv d\mu(x)\delta[y = T(x)]$, where T is a measurable map $X \to Y$.

- Chap. 1: Kantorovich Duality.
- Chap. 2: Geometry of Optimal Transportation.
- Chap. 3: Brenier's Polar Factorization Theorem.
- Chap. 4: Monge-Ampère Equation.
- Chap. 5: Displacement Interpolation & Displacement Convexity.
- Chap. 6: Geometric & Gaussian Inequalities.
- Chap. 7: Metric Side of Optimal Transportation.
- Chap. 8: A Differential Point of View on Optimal Transportation.
- Chap. 9: Entropy Production & Transportation Inequalities.
- Chap. 10: Problems.

4 Dynamic Programming – Quy Hoạch Động

5 Linear Programming – Quy Hoạch Tuyến Tính

Definition 7 (Linear programming)**.** *"*Linear programming (LP)*, also called* linear optimization*, is a method to achieve the best outcome, e.g., maximum profit or lower cost, in a mathematical model whose requirements & objective are represented by linear relationships. Linear programming is a special case of mathematical programming ≡ mathematical optimization."* – *Wikipedia/linear programming*

More formally, linear programming is a technique for the optimization of a linear linear objective function, subject to linear equality & linear inequality constraints. Its feasible region is a convex polytope, which is a set defined as the intersection of finitely many half spaces, each of which is defined by a linear inequality. Its objective function is a real-valued affine (linear) function defined on this polytope. A linear programming algorithm finds a point in the polytope where this function has the largest (or smallest) value if such a point exists.

Linear programs are problems that can be expressed in standard form as

$$\text{Find a vector } \mathbf{x} \text{ that maximizes/minimizes } \mathbf{c}^\top \mathbf{x} \text{ subject to } A\mathbf{x} \leq \mathbf{b} \ \& \ \mathbf{x} \geq \mathbf{0}. \tag{lp}$$

Here the components of $\mathbf{x}$ are the variables to be determined, $\mathbf{b}, \mathbf{c}$ are given vectors, & A is a given matrix. The function whose value is to be maximized ($\mathbf{x} \mapsto \mathbf{c}^\top \mathbf{x}$ in this case) is called the objective function. The constraint $A\mathbf{x} \leq \mathbf{x}$ & $\mathbf{x} \geq \mathbf{0}$ specify a convex polytope over which the objective function is to be optimized.

Linear programming can be applied to various fields of study, which is widely used in mathematics &, to a lesser extent, in business, economics, & to some engineering problems. There is a close connection between linear programs, eigenequations, John von Neumann's general equilibrium model, & structural equilibrium models (see dual linear program). Industries using linear programming models include transportation, energy, telecommunications, & manufacturing. It has proven useful in modeling diverse types of problems in planning, routing, scheduling, assignment, & design.

Định nghĩa 1 (Quy hoạch tuyến tính)**.** *Bài toán* quy hoạch tuyến tính *là bài toán tìm* GTLN/GTNN *của* hàm mục tiêu *trong điều kiện hàm mục tiêu là hàm bậc nhất đối với các biến & mỗi 1 điều kiện ràng buộc là bất phương trình bậc nhất đối với các biến (không kể điều kiện ràng buộc biến thuộc tập số nào, e.g.,* $\mathbb{N}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$*.*

Ta có thể viết bài toán quy hoạch tuyến tính 2 biến x, y về dạng sau:

$$\max T := \alpha x + \beta y \text{ s.t } a_i x + b_i y \leq c_i, \ \forall i = 1, 2, \ldots, n, \tag{lp2max}$$
$$\min T := \alpha x + \beta y \text{ s.t } a_i x + b_i y \leq c_i, \ \forall i = 1, 2, \ldots, n, \tag{lp2min}$$

trong đó các điều kiện ràng buộc đều là các bất phương trình bậc nhất đối với x, y. See also:

- *Problem: Inequation & Linear System of Inequations – Bài Tập: Bất Phương Trình & Hệ Bất Phương Trình.*

  Folder: Elementary STEM & Beyond/Elementary Mathematics/grade 10/linear system inequations/problem: [pdf[2]][TEX[3]].

[2] URL: https://github.com/NQBH/elementary_STEM_beyond/blob/main/elementary_mathematics/grade_10/linear_system_inequations/problem/NQBH_linear_system_inequations_problem.pdf.

[3] URL: https://github.com/NQBH/elementary_STEM_beyond/blob/main/elementary_mathematics/grade_10/linear_system_inequations/problem/NQBH_linear_system_inequations_problem.tex.

- *Problem & Solution: Inequation & Linear System of Inequations – Bài Tập & Lời Giải: Bất Phương Trình & Hệ Bất Phương Trình.*
  Folder: Elementary STEM & Beyond/Elementary Mathematics/grade 10/linear system inequations/solution: [pdf[4]][TEX[5]].

- *Problem: Mathematical Optimization – Bài Tập: Ứng Dụng Toán Học Để Giải Quyết 1 Số Bài Toán Tối Ưu.*
  Folder: Elementary STEM & Beyond/Elementary Mathematics/grade 12/optimization/problem: [pdf[6]][TEX[7]].

- *Problem & Solution: Mathematical Optimization – Bài Tập & Lời Giải: Ứng Dụng Toán Học Để Giải Quyết 1 Số Bài Toán Tối Ưu.*
  Folder: Elementary STEM & Beyond/Elementary Mathematics/grade 12/optimization/solution: [pdf[8]][TEX[9]].

5.1 How to solve some linear programmings – Cách giải 1 số bài toán quy hoạch tuyến tính

Có thể giải 1 số bài toán quy hoạch tuyến tính dạng (lp2max) hay (lp2min) theo 2 bước:

1. Xác định miền nghiệm $S \subset \mathbb{R}^2$ của hệ bất phương trình $a_i x + b_i y \leq c_i$, $\forall i = 1, \ldots, n$.
2. Tìm điểm $(x, y) \in S$ sao cho biểu thức $T = T(x, y) = \alpha x + \beta y$ có GTLN ở bài toán (lp2max) hoặc có GTNN ở bài toán (lp2min).

   Khi miền nghiệm S là đa giác (polygon), biểu thức $T(x, y) = \alpha x + \beta y$ đạt GTLN/GTNN (gộp chung gọi là *cực trị*) tại $(x, y) \in \mathbb{R}^2$ là tọa độ 1 trong các đỉnh của đa giác đó. Khi đó, bước 2 có thể được thực hiện như sau:

   (a) Xác định tọa độ các đỉnh của đa giác đó.

   (b) Tính giá trị của biểu thức $T(x, y) = \alpha x + \beta y$ tại các đỉnh của đa giác đó.

   (c) So sánh các giá trị & kết luận.

[**CDHT_Toan_12_Canh_Dieu**].

6 Control Theory – Lý Thuyết Điều Khiển

"*Control theory* is a field of control engineering & applied mathematics that deals with the control of dynamical systems in engineered processes & machines. The objective is to develop a model or algorithm governing the application of system inputs to drive the system to a desired state, while minimizing any *delay, overshoot*, or *steady-state error* & ensuring a level of control stability; often with the aim to achieve a degree of optimality.

To do this, a *controller* with the requisite corrective behavior is required. This controller monitors the controlled process variable (PV), & compares it with the reference or set point (SP). The difference between actual & desired value of the process variable, called the *error* signal, or SP-PV error, is applied as feedback to generate a control action to bring the controlled process variable to the same value as the set point. Other aspects which are also studied are controllability & observability. Control theory is used in control system engineering to design automation that have revolutionized manufacturing, aircraft, communications, & other industries, & created new fields such as robotics.

Extensive use is usually made of a diagrammatic style known as the block diagram. In it the transfer function, also known as the system function or network function, is a mathematical model of the relation between the input & output based on the differential equations describing the system.

Control theory dates from the 19th century, when the theoretical basis for the operation of governors was 1st described by James Clerk Maxwell. Control theory was further advanced by Edward Routh in 1874, Charles Sturm & in 1895, Adolf Hurwitz, who all contributed to the establishment of control stability criteria; & from 1922 onwards, the development of PID control theory by Nicolas Minorsky. Although a major application of mathematical control theory is in control systems engineering, which deals with the design of process control systems for industry, other applications range far beyond this. As the general theory of feedback systems, control theory is useful wherever feedback occurs – thus control theory also has applications in life sciences, computer engineering, sociology, & operations research." – Wikipedia/control theory

[4] URL: https://github.com/NQBH/elementary_STEM_beyond/blob/main/elementary_mathematics/grade_10/linear_system_inequations/solution/NQBH_linear_system_inequations_solution.pdf.
[5] URL: https://github.com/NQBH/elementary_STEM_beyond/blob/main/elementary_mathematics/grade_10/linear_system_inequations/solution/NQBH_linear_system_inequations_solution.tex.
[6] URL: https://github.com/NQBH/elementary_STEM_beyond/blob/main/elementary_mathematics/grade_12/optimization/problem/NQBH_optimization_problem.pdf.
[7] URL: https://github.com/NQBH/elementary_STEM_beyond/blob/main/elementary_mathematics/grade_12/optimization/problem/NQBH_optimization_problem.tex.
[8] URL: https://github.com/NQBH/elementary_STEM_beyond/blob/main/elementary_mathematics/grade_12/optimization/solution/NQBH_optimization_solution.pdf.
[9] URL: https://github.com/NQBH/elementary_STEM_beyond/blob/main/elementary_mathematics/grade_12/optimization/solution/NQBH_optimization_solution.tex.

Open-loop & closed-loop (feedback) control. "Fundamentally, there are 2 types of control loop: open-loop control (feed-forward), & closed-loop control (feedback).

In open-loop control, the control action from the controller is independent of the "process output" (or "controlled process variable"). A good example of this is a central heating boiler controlled only by a timer, so that heat is applied for a constant time, regardless of the temperature of the building. The control action is the switching on/off of the boiler, but the controlled variable should be the building temperature, but is not because this is open-loop control of the boiler, which does not give closed-loop control of the temperature.

In closed loop control, the control action from the controller is dependent on the process output. In the case of the boiler analogy this would include a thermostat to monitor the building temperature, & thereby feed back a signal to ensure the controller maintains the building at the temperature set on the thermostat. A closed loop controller therefore has a feedback loop which ensures the controller exerts a control action to give a process output the same as the "reference input" or "set point". For this reason, closed loop controllers are also called *feedback controllers*.

The definition of a closed loop control system according to the British Standards Institutin is "a control system possessing monitoring feedback, the deviation signal formed as a result of this feedback being used to control the action of a final control element in such a way as to tend to reduce the deviation to 0."

Likewise; "A *Feedback Control System* is a system which tends to maintain a prescribed relationship of 1 system variable to another by comparing functions of these variables & using the difference as a means of control."

Classical control theory.

6.1 Application of C_0-Semigroup in Control Theory – Ứng Dụng của Nửa Nhóm C_0 Trong Lý Thuyết Điều Khiển

7 C_0 Semigroup – Nửa Nhóm C_0

Resources – Tài nguyên.

1. [AK16]. Cung Thế Anh, Trần Đình Kế. *Nửa Nhóm Các Toán Tử Tuyến Tính & Ứng Dụng.*

"Ứng dụng của lý thuyết nửa nhóm các toán tử tuyến tính trong lý thuyết điều khiển toán học, bao gồm bài toán điều khiển được, bài toán quan sát, bài toán điều khiển tối ưu, & bài toán ổn định hóa. Nhấn mạnh đến các hệ điều khiển vô hạn chiều, i.e., các hệ điều khiển sinh bởi các PDEs." – [AK16, Chap. IV: *Ứng Dụng Trong Lý Thuyết Diều Khiển*]

8 Feedback Control – Kiểm Soát Phản Hồi/Điều Khiển Hồi Tiếp

Resources – Tài nguyên.

1. [Ray07]. J.-P. Raymond. *Feedback Boundary Stabilization of 3D Incompressible NSEs.*

   Study local stabilization of 3D NSEs around an unstable stationary solution $\mathbf{w}$, by means of a feedback boundary control. 1st determine a feedback law for the linearized system around $\mathbf{w}$. Show: this feedback provides a local stabilization of NSEs. To deal with the nonlinear term, the solutions to the closed loop system must be in $H^{\frac{3}{2}+\varepsilon,\frac{3}{4}+\frac{\varepsilon}{2}}(Q)$ with $\varepsilon > 0$. Such a regularity is achieved with a feedback obtained by minimizing a functional involving a norm of the state variable strong enough. The feedback controller cannot be determined by a well posed Riccati equation. Here choose a controller at $t = 0$, is achieved by choosing a time varying control operator in a neighborhood of $t = 0$.

   Keywords: Dirichlet control; Navier–Stokes equations; Feedback control; Stabilization; Riccati equation.

   An important issue in control theory is the controllability of systems. The local stability of 3D NSEs in a neighborhood of an unstable stationary solution may be deduced from this controllability result. Another important issue is the characterization of stabilizing feedback control laws, pointwise in time.

9 Optimal Control – Điều Khiển Tối Ưu

Resources – Tài nguyên.

1. [LT00]. Irena Lasiecka, Roberto Triggiani. *Control Theory for PDEs: Continuous & Approximation Theories II: Abstract Hyperbolic-Like Systems Over a Finite Time Horizon.*

2. [Lio69]. Jacques-Louis Lions. *Quelques méthodes de résolution des problèmes aux limites nonlinéaires – Some Methods for Solving Nonlinear Boundary Problems.*

3. [Lio71]. Jacques-Louis Lions. *Optimal control of systems governed by PDEs.*

4. [Trö10]. Fredi Tröltzsch. *Optimal Control of PDEs.*

Definition 8 (Optimal control theory)**.** *"Optimal control theory is a branch of control theory that deals with finding a control for a dynamical system over a period of time s.t. an objective function is optimized. It has numerous applications in science, engineering, & operational research. E.g., the dynamical system might be a spacecraft with controls corresponding to rocket thrusters, & the objective might be to reach the Moon with minimum fuel expenditure. Or the dynamical system could be a nation's economy, with the objective to minimize unemployment; the controls in this case could be fiscal & monetary policy. A dynamical system may also be introduced to embed operations research problems within the framework of optimal control theory."* – *Wikipedia/optimal control*

"Optimal control is an extension of the calculus of variations, & is a mathematical optimization method for deriving control policies. The method is largely due to the work of Lev Pontryagin & Richard Bellman in the 1950s, after contributions to calculus of variations by Edward J. McShane. Optimal control can be seen as a control strategy in control theory." – Wikipedia/optimal control

General method. Optimal control deals with the problem of finding a control law for a given system s.t. a certain optimality criterion is achieved. A control problem includes a cost functional that is a function of state & control variables. An *optimal control* is a set of differential equations describing the paths of the control variables that minimize the cost function. The optimal control can be derived using Pontryagin's maximum principle (a necessary condition also known as Pontryagin's minimum principle or simply Pontryagin's principle), or by solving the Hamilton–Jacobi–Bellman equation0 (a sufficient condition).

Example 1. *Consider a car traveling in a straight line on a hilly road. Question: How should the driver press the accelerator pedal in order to minimize the total traveling time? The term* control law *refers specifically to the way in which the driver presses the accelerator & shifts the gears. The* system *consists of both the car & the road, & the* optimality criterion *is the minimization of the total traveling time. Control problems usually include ancillary constraints. E.g., the amount of available fuel might be limited, the accelerator pedal cannot be pushed through the floor of the car, speed limits, etc.*

A proper cost function will be a mathematical expression giving the traveling time as a function of the speed, geometrical considerations, & initial conditions of the system. Constraints are often interchangeable with the cost function.

Another related optimal control problem may be to find the way to drive the car so as to minimize its fuel consumption, given that it must complete a given course in a time not exceeding some amount. Yet another related control problem may be to minimize the total monetary cost of completing the trip, given assumed monetary prices for time & fuel.

An abstract framework. Minimize the continuous-time cost functional

$$J(t_0, t_\mathrm{f}, \mathbf{x}(\cdot), \mathbf{u}(\cdot)) \coloneqq E(\mathbf{x}(t_0), t_0, \mathbf{x}(t_\mathrm{f}), t_\mathrm{f}) + \int_{t_0}^{t_\mathrm{f}} F(t, \mathbf{x}(t), \mathbf{u}(t))\,\mathrm{d}t, \tag{2}$$

subject to the 1st-order dynamic constraints (the *state equation*)

$$\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)), \tag{3}$$

the algebraic *path constraints*

$$\mathbf{h}(t, \mathbf{x}(t), \mathbf{u}(t)) \leq \mathbf{0}, \tag{4}$$

& the endpoint conditions

$$\mathbf{e}(t_0, \mathbf{x}(t_0), t_\mathrm{f}, \mathbf{x}(t_\mathrm{f})) = \mathbf{0}, \tag{5}$$

where $\mathbf{x}(t)$: the *state*, $\mathbf{u}(t)$ is the *control*, t: the independent variable (generally speaking, time), t_0: the initial time, & t_f: the terminal time. The terms E, F are called the *endpoint cost* & *running cost*, respectively. In the calculus of variations, E, F are referred to as the Mayer term & the Lagrangian, respectively. Furthermore, it is noted that the path constraints are in general *inequality* constraints & thus may not be active (i.e., $= 0$) at the optimal solution. It is also noted that the optimal control problem as stated above may have multiple solutions (i.e., the solution may not be unique). Thus, it is most often the case that any solution $(t_0^\star, t_\mathrm{f}^\star, \mathbf{x}^\star(t), \mathbf{u}^\star(t))$ to the optimal control problem is *locally minimizing/minimizer*.

9.1 Linear Quadratic Control

A special case of the general nonlinear optimal control problem is the linear quadratic (LQ) optimal control problem. The LQ problem is stated as follows. Minimize the *quadratic* continuous-time cost functional

$$J = \frac{1}{2}\mathbf{x}^\top(t_\mathrm{f})S_\mathrm{f}\mathbf{x}(t_\mathrm{f}) + \frac{1}{2}\int_{t_0}^{t_\mathrm{f}} \mathbf{x}^\top(t)Q(t)\mathbf{x}(t) + \mathbf{u}^\top(t)R(t)\mathbf{u}(t)\,\mathrm{d}t, \tag{6}$$

subject to the linear 1st-order dynamic constraints

$$\begin{cases} \dot{\mathbf{x}}(t) = A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t), \\ \mathbf{x}(t_0) = \mathbf{x}_0. \end{cases} \tag{7}$$

A particular form of the LQ problem arising in many control system problems is that of the *linear quadratic regulator* (LQR) where all of the matrices (i.e., A, B, Q, R) are constant, the initial time is arbitrarily set to 0, & the terminal time is taken in

the limit $t_\mathrm{f} \to 0$ (this last assumption is what is known as *infinite horizon*). The LQR problem is stated as follows. Minimize the infinite horizon quadratic continuous-time cost functional

$$J = \frac{1}{2}\int_0^\infty \mathbf{x}^\top(t)Q\mathbf{x}(t) + \mathbf{u}^\top(t)R\mathbf{u}(t)\,\mathrm{d}t, \tag{8}$$

subject to the *linear time-invariant* 1st-order dynamic constraints

$$\begin{cases} \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t), \\ \mathbf{x}(t_0) = \mathbf{x}_0. \end{cases} \tag{9}$$

In the finite-horizon case the matrices are restricted in that Q, R are positive semi-definite & positive definite, respectively. In the infinite-horizon case, however, the matrices Q, R are not only positive-semidefinite & positive-definite, respectively, but are also constant. These additional restrictions on Q, R in the infinite-horizon case are enforced to ensure that the cost functional remains positive. Furthermore, in order to ensure that the cost function is *bounded*, the additional restriction is imposed that the pair (A, B) is *controllable*. Note that the LQ or LQR cost functional can be thought of physically as attempting to minimize the *control energy* (measured as a quadratic form).

The infinite horizon problem, i.e., LQR, may seem overly restrictive & essentially useless because it assumes that the operator is driving the system to 0-state & hence driving the output of the system to 0. This is indeed correct. However the problem of driving the output to a desired nonzero level can be solved *after* the zero output one is. In fact, can prove that this secondary LQR problem can be solved in a very straightforward manner. It has been shown in classical optimal control theory that the LQ (or LQR) optimal control has the feedback form

$$\mathbf{u}(t) = -K(t)\mathbf{x}(t), \tag{10}$$

where $K(t)$ is a properly dimensioned matrix, given as

$$K(t) = R^{-1}B^\top S(t), \tag{11}$$

& $S(t)$ is the solution of the differential Riccati equation. The differential Riccati equation is given as

$$\dot{S}(t) = -S(t)A - A^\top S(t) + S(t)BR^{-1}B^\top S(t) - Q. \tag{12}$$

For the finite horizon LQ problem, the Riccati equation is integrated backward in time using the terminal boundary condition

$$S(t_\mathrm{f}) = S_\mathrm{f}. \tag{13}$$

For the infinite horizon LQR problem, the differential Riccati equation is replaced with the *algebraic* Riccati equation (ARE) given as

$$-SA - A^\top S + SBR^{-1}B^\top S - Q = \mathbf{0}. \tag{14}$$

Understanding that the ARE arises from infinite horizon problem, the matrices A, B, Q, R are all constant. It is noted that there are in general multiple solutions to the algebraic Riccati equation & the *positive definite* (or positive semi-definite) solution is the one that is used to compute the feedback again. The LQ (LQR) problem was elegantly solved by Rudolf E. Kálmán.

9.2 Numerical Methods for Optimal Control

"Optimal control problems are generally nonlinear & therefore, generally do not have analytic solutions, e.g., like the linear-quadratic optimal control problem. As a result, it is necessary to employ numerical methods to solve optimal control problems. In the early years of optimal control (c. 1950s–1980s) the favored approach for solving optimal control problems was that of *indirect methods*. In an indirect method, the calculus of variations is employed to obtain the 1st-order optimality conditions. These conditions result in a 2-point (or, in the case of a complex problem, a multi-point) BVP. This BVP actually has a special structure because it arises from taking the derivative of a Hamiltonian. Thus, the resulting dynamical system is a Hamiltonian system of the form

$$\begin{cases} \dot{\mathbf{x}} = \partial_{\boldsymbol{\lambda}} H, \\ \dot{\boldsymbol{\lambda}} = -\partial_{\mathbf{x}} H, \end{cases} \tag{15}$$

where $H = F + \boldsymbol{\lambda}^\top \mathbf{f} - \boldsymbol{\mu}^\top \mathbf{h}$ is the *augmented Hamiltonian* & in an indirect method, the BVP is solved (using the appropriate boundary or *transversality* conditions). The beauty of using an indirect method is that the state & adjoint, i.e., $\boldsymbol{\lambda}$, are solved for & the resulting solution is readily verified to be an extremal trajectory. The disadvantage of indirect methods is that the BVP is often extremely difficult to solve (particularly for problems that span large time intervals or problems with interior point constraints). A well-known software program that implements direct methods is BNDSCO.

The approach that has risen to prominence in numerical optimal control since the 1980s is that of so-called *direct methods*. In a direct method, the state or the control, or both, are approximated using an appropriate function approximation (e.g., polynomial approximation or piecewise constant parameterization). Simultaneously, the cost functional is approximated as a *cost function*. Then, the coefficients of the function approximations are treated as optimization variables & the problem is "transcribed" to a nonlinear optimization problem of the form

$$\min F(\mathbf{z}) \tag{16}$$

subject to the algebraic constraints

$$\mathbf{g}(\mathbf{z}) = \mathbf{0},\ \mathbf{h}(\mathbf{z}) \leq \mathbf{0}. \tag{17}$$

Depending upon the type of direct method employed, the size of the nonlinear optimization problem can be quite small (e.g., as in a direct shooting or quasilinearization method), moderate (e.g., pseudospectral optimal control) or may be quite large (e.g., a direct collocation method). In the latter case (i.e., a collocation method), the nonlinear optimization problem may be literally thousands $a \cdot 10^3$ to tens of thousands $a \cdot 10^4$ of variables & constraints. Given the size of many NLPs arising from a direct method, it may appear somewhat counter-intuitive that solving the nonlinear optimization problem is easier than solving the BVP. It is, however, the fact that the NLP is easier to solve than the BVP. The reason for the relative ease of computation, particularly of a direct collocation method, is that the NLP is *sparse* & many well-known software programs exist (e.g., SNOPT) to solve large sparse NLPs. As a result, the range of problems that can be solved via direct methods (particularly direct *collocation methods* which are very popular these days) is significantly larger than the range of problems that can be solved via indirect methods. In fact, direct methods have become so popular these days that many people have written elaborate software programs employing these methods. In particular, many such programs include DIRCOL, SOCS, OTIS, GESOP/ASTOS, DITAN., & PyGMO/PyKEP. In recent years, due to the advent of the MATLAB programming language, optimal control software in MATLAB has become more common. Examples of academically developed MATLAB software tools implementing direct methods include RIOTs, DIDO, DIRECT, FALCON.m, & GPOPs, while an example of an industry developed MATLAB tool is PROPT. These software tools have increased significantly the opportunity for people to explore complex optimal control problems both for academic research & industrial problems. Finally, it is noted that general-purpose MATLAB optimization environments such as TOMLAB have made coding complex optimal control problems significantly easier than was previously possible in languages such as C & FORTRAN.

9.3 Discrete-Time Optimal Control

"The examples thus far have shown continuous time systems & control solutions. In fact, as optimal control solutions are now often implemented digitally, contemporary control theory is now primarily concerned with discrete time systems & solutions. The theory of Consistent Approximations provides conditions under which solutions to a series of increasingly accurate discretized optimal control problem converge to the solution of the original, continuous-time problem. Not all discretization methods have this property, even seemingly obvious ones. E.g., using a variable step-size routine to integrate the problem's dynamic equations may generate a gradient which does not converge to 0 (or point in the right direction) as the solution is approached. The direct method RIOTS is based on the Theory of Consistent Approximation.

A common solution strategy in many optimal control problems is to solve for the costate (sometimes called the shadow price) $\lambda(t)$. The costate summaries in 1 number the marginal value of expanding or contracting the state variable next turn. The marginal value is not only the gains accruing to it next turn but associated with the duration of the program. It is nice when $\lambda(t)$ can be solved analytically, but usually, the most one can do is describe it sufficiently well that the intuition can grasp the character of the solution & an equation solver can solve numerically for the values.

Having obtained $\lambda(t)$, the turn-t optimal value for the control can usually be solved as a differential equation conditional on knowledge of $\lambda(t)$. Again it is infrequent, especially in continuous-time problems, that one obtains the value of the control or the state explicitly. Usually, the strategy is to solve for thresholds & regions that characterize the optimal control & use a numerical solver to isolate the actual choice values in time.

Example 2. *"Consider the problem of a mine owner who must decide at what rate to extract ore from their mine. They own rights to the ore from date 0 to date T. At date 0 there is x_0 ore in the ground, & the time-dependent amount of ore $x(t)$ left in the ground declines at the rate of $u(t)$ that the mine owner extracts it. The mine owner extracts ore at cost $\frac{u(t)^2}{x(t)}$ (the cost of extraction increasing with the square of the extraction speed & the inverse of the amount of ore left) & sells ore at a constant price p. Any ore left in the ground at time T cannot be sold & has no value (there is no "scrap value"). The owner chooses the rate of extraction varying with time $u(t)$ to maximize profits over the period of ownership with no time discounting. See discrete-time version vs. Continuous-time version." – Wikipedia/ optimal control/ finite time*

9.4 Optimal Control for PDEs – Điều Khiển Tối Ưu Cho Phương Trình Vi Phân Đạo Hàm Riêng

Resources – Tài nguyên.

1. [Lio71]. Jacques-Louis Lions. *Optimal control of systems governed by PDEs.*
2. [Trö10]. Fredi Tröltzsch. *Optimal Control of PDEs.*

9.4.1 Optimal Control for Navier–Stokes Equations – Điều Khiển Tối Ưu Cho Phương Trình Navier–Stokes

10 Stochastic Control

Community – Cộng đồng. Michael Hintermüller, Caroline Geiersbach.

Stochastic control or *stochastic optimal control* is a subfield of control theory that deals with the existence of uncertainty either in observations or in the noise that drives the evolution of the system. The system designer assumes, in a Bayesian probability-driven fashion, that random noise with known probability distribution affects the evolution & observation of the

state variables. Stochastic control aims to design the time path of the controlled variables that performs the desired control task with minimum cost, somehow defined, despite the presence of this noise. The context may be either discrete time or continuous time.

Certainty equivalence. "An extremely well-studied formulation in stochastic control is that of linear quadratic Gaussian control. Here the model is linear, the objective function is the expected value of a quadratic form, & the disturbances are purely additive. A basic result for discrete-time centralized systems with only additive uncertainty is the *certainty equivalence property*: that the optimal control solution in this case is the same as would be obtained in the absence of the additive disturbances. This property is applicable to all centralized systems with linear equations of evolution, quadratic cost function, & noise entering the model only additively; the quadratic assumption allows for the optimal control laws, which follow the certainty-equivalence property to be linear functions of the observations of the controllers.

Any deviation from the above assumptions – a nonlinear stat equation, a non-quadratic objective function, noise in the multiplicative parameters of the model, or decentralization of control – causes the certainty equivalence property not to hold. E.g., its failure to hold for decentralized control was demonstrated in Witsenhausen's counterexample." – Wikipedia/stochastic controlcertainty equivalence.

Discrete time. "In a discrete-time context, the decision-maker observes the state variable, possibly with observational noise, in each time period. The objective may be to optimize the sum of expected values of a nonlinear (possibly quadratic) objective function over all the time periods from the present to the final period of concern, or to optimize the value of the objective function as of the final period only. At each time period new observations are made, & the control variables are to be adjusted optimally. Finding the optimal solution for the present time may involve iterating a matrix Riccati equation backwards in time from the last period to the present period.

In the discrete-time case with uncertainty about the parameter values in the transition matrix (giving the effect of current values of the state variables on their own evolution) &/or the control response matrix of the state equation, but still with a linear state equation & quadratic objective function, a Riccati equation can still be obtained for iterating backward to each period's solution even though certainty equivalence does not apply. The discrete-time case of a non-quadratic loss function but only additive disturbances can also be handled, albeit with more complications.

Example 3. *A typical specification of the discrete-time stochastic linear quadratic control problem is to minimize*

$$\mathrm{E}_1 \sum_{t=1}^{S} y_t^\top Q y_t + u_t^\top R u_t, \tag{18}$$

where E_1 *is the expected value operator conditional on* y_0*, S: the time horizon, subject to the state equation* $y_t = A_t y_{t-1} + B_t u_t$*.*

Continuous time. "If the model is in continuous time, the controller knows the state of the system at each instant of time. The objective is to maximize either an integral of, e.g., a concave function of a state variable over a horizon from time 0 (the present) to a terminal time T, or a concave function of a state variable at some future date T. As time evolves, new observations are continuously made & the control variables are continuously adjusted in optimal fashion."

Stochastic model predictive control. "In the literature, there are 2 types of MPCs for stochastic systems; Robust model predictive control & Stochastic Model Predictive Control (SMPC). Robust model predictive control is a more conservative method which considers the worst scenario in the optimization procedure. However, this method, similar to other robust controls, deteriorates the overall controller's performance & also is applicable only for systems with bounded uncertainties. The alternative method, SMPC, considers soft constraints which limit the risk of violation by a probabilistic inequality.

Example 4 (Stochastic model predictive control in finance)**.** *"In a continuous time approach in a finance context, the state variable in the stochastic differential equation is usually wealth or net worth, & the controls are the shares placed at each time in the various assets. Given the asset allocation chosen at any time, the determinants of the change in wealth are usually the stochastic returns to assets & the interest rate on the risk-free asset. The field of stochastic control has developed greatly since the 1970s, particularly in its applications to finance. Robert Merton used stochastic control to study optimal portfolios of safe & risky assets. Merton's portfolio problem & that of Black Scholes changed the nature of the finance literature. Influential mathematical textbook treatments were by Fleming & Rishel, & by Fleming & Soner. These techniques were applied by Stein to the financial crisis of 2007–08.*

The maximization, say of the expected logarithm of net work at a terminal date T*, is subject to stochastic processes on the components of wealth. In this case, in continuous time Itô's equation is the main tool of analysis. In the case where the maximization is an integral of a concave function of utility over an horizon* $(0, T)$*, dynamic programming is used. There is no certainty equivalence as in the older literature, because the coefficients of the control variables – i.e., the returns received by the chosen shares of assets – are stochastic.*

11 Shape Optimization – Tối Ưu Hình Dạng

Resources – Tài nguyên.

1. [AH01]. Grégoire Allaire, Antoine Henrot. *On some recent advances in shape optimization.*
2. [Aze20]. Hideyuki Azegami. *Shape Optimization Problems.*
3. [BW23]. Catherine Bandle, Alfred Wagner. *Shape Optimization: Variations of Domains & Applications.*
4. [DZ01; DZ11]. Michael C. Delfour, Jean-Paul Zolésio. *Shapes & Geometries.*
5. [HM03]. J. Haslinger, R. A. E. Mäkinen. *Introduction to Shape Optimization.*
6. [MP10]. Bijan Mohammadi, Olivier Pironneau. *Applied Shape Optimization for Fluids.*
7. [MZ06]. Marwan Moubachir, Jean-Paul Zolésio. *Moving Shape Analysis & Control.*
8. Stephan Schmidt. Master course: *Shape & Geometry.* Humboldt University of Berlin. [written in German, taught in English & German].
9. [SZ92]. Jan Sokołowski, Jean-Paul Zolésio. *Introduction to Shape Optimization.*
10. [Wal15]. Shawn W. Walker. *The Shapes of Things.*

Differential equations on surfaces. Differential geometry is useful for understanding mathematical models containing geometric PDEs, e.g., surface/manifold version of the standard Laplace equation, which requires the development of the surface gradient & surface Laplacian operators – the usual gradient ∇ & Laplacian $\Delta = \nabla \cdot \nabla$ operators defined on a surface (manifold) instead of standard Euclidean space $\mathbb{R}^n$. *Advantage*: provide alternative formulas for geometric quantities, e.g., the summed (mean) curvature, that are much clearer than the usual presentation of texts on differential geometry.

Differentiating w.r.t. Shape. The approach to differential geometry is advantageous for developing the framework of *shape differential calculus* – the study of how quantities change w.r.t. changes of independent "shape variable".

"The framework of shape differential calculus provides the tools for developing the equations of mean curvature flow & Willmore flow, which are geometric flows occurring in many applications such as fluid dynamics & biology." – [Wal15, p. 2]

The shape perturbation $\delta J(\Omega; V)$ is similar to the gradient operator, which is a directional derivative, analogous to $V \cdot \nabla f$ where V is a given direction, providing information about the local slope, or the sensitivity of a quantity w.r.t. some parameters.

It takes only 2 or 3 numbers to specify a point (x, y) in 2D & a point (x, y, z) in 3D, whereas an "infinite" number of coordinate pairs is needed to specify a domain Ω. V is a 2D/3D vector in the scalar function setting; for a shape functional, V is a full-blown function requiring definition at every point in Ω. This "infinite dimensionality" is the reason for using the notation $\delta J(\Omega; V)$ to denote a shape perturbation. $\delta J(\Omega; V)$ indicates how we should change Ω to decrease J, similarly to how $\nabla f(x, y)$ indicates how the coordinate pair (x, y) should change to decrease f, which opens up the world of shape optimization.

3 schools of shape optimization. Cf. engineering shape optimization vs. applied shape optimization [MP10] vs. theoretical shape optimization [SZ92; DZ11].

Shape perturbations allow us to "climb down the hill" in the infinite dimensional setting of shape, which is a powerful tool for producing sophisticated engineering designs in an automatic way.

Extrinsic vs. intrinsic point of views. To make the discussion as clear as possible, we adopt the *extrinsic* point of view: curves & surfaces are assumed to lie in a Euclidean space of higher dimension. The ambient space is 3D Euclidean space. Alternatively, there is the *intrinsic* point of view, i.e., the surface is not assumed to lie in an ambient space, i.e., one is not allowed to reference anything "outside" of the surface when defining it. Moreover, no mathematical structures "outside" of the surface can be utilized. Walker [Wal15] did not adopt the intrinsic view or consider higher dimensional manifolds, general embedding dimensions, etc. for the reasons: [Wal15] is meant as a *practical guide* to differential geometry & shape differentiation that can be used by researchers in other fields.

- [Wal15] is meant to be used as background information for deriving physical models where geometry plays a critical role. Because most physical problems of interest take place in 3D Euclidean space, the extrinsic viewpoint is sufficient.
- Many of the proofs & derivations of differential geometry relations simplify dramatically for 2D surfaces in 3D & require only basic multivariable calculus & linear algebra.
- The concepts of *normal vectors* & *curvature* are harder to motivate with the intrinsic viewpoint. *What does it meant for a surface to "curve through space" if you cannot talk about the ambient space?*
- Walker wants to keep in mind applications of this machinery to geometric PDEs, fluid dynamics, numerical analysis, optimization, etc. An interesting application of this methodology is for the development of numerical methods for mean curvature flow & surface tension driven fluid flow. Ergo (= therefore), the extrinsic viewpoint is often more convenient for computational purposes.
- Walker wants his framework to be useful for analyzing & solving *shape optimization* problems, i.e., optimization problems where geometry/shape is the control variable.

Prerequisites. "When reading any mathematical text, the reader must have a certain level of mathematical "maturity" in order to efficiently learn what is in the text.

Example 5 ([Wal15], Sect. 1.2.1, pp. 1–2)**.** *Let $f = f(r,\theta)$ be a smooth function defined on the disk $B_{R,2}(0,0)$ of radius R in terms of polar coordinates. The integral of f over $B_{2,R}(0,0)$ $J \coloneqq \int_{B_{2,R}(0,0)} f\,\mathrm{d}\mathbf{x} = \int_0^{2\pi}\int_0^R f(r,\theta)\,\mathrm{d}r\,\mathrm{d}\theta$ depends on R. Assume f also depends on R, i.e., $f = f(r,\theta,R)$ with a physical example: J is the* net flow rate *of liquid through a pipe with cross-section Ω, then f is the flow rate per unit area & could be the solution of a PDE defined on Ω, e.g., a Navier–Stokes fluid flowing in a circular pipe. Advantageous to know the* sensitivity *of J w.r.t. R, e.g., for optimization purposes. Differentiate J w.r.t. R:*

$$\frac{d}{dR}J = \int_0^{2\pi}\left(\frac{d}{dR}\int_0^R f(r,\theta;R)r\,\mathrm{d}r\right)\mathrm{d}\theta = \int_0^{2\pi}\int_0^R f'(r,\theta;R)r\,\mathrm{d}r\,\mathrm{d}\theta + \int_0^{2\pi} f(R,\theta;R)\,\mathrm{d}\theta.$$

The dependence of f on R can more generally be viewed as dependence on $B_{R,2}(0,0)$, i.e., $f(\cdot;R) \equiv f(\cdot;B_{R,2}(0,0))$. Rewriting d/dRJ using Cartesian coordinates $\mathbf{x}$:

$$\frac{d}{dR}J = \int_{B_{R,2}(0,0)} f'(\mathbf{x};\Omega)\,\mathrm{d}\mathbf{x} + \int_{S_{R,2}(0,0)} f(\mathbf{x};\Omega)\,\mathrm{d}S(\mathbf{x}), \tag{19}$$

where $\mathrm{d}\mathbf{x}$ is the volume measure, $\mathrm{d}S(\mathbf{x})$ is the surface area measure.

Example 6 (Surface height function of a hill)**.** *Let $f = f(x,y)$ be a function describing the surface height of the hill, where (x,y) are the coordinates of our position. Then, by using basic multivariate calculus, finding a direction that will move us downhill is equivalent to computing the gradient (vector) of f & moving in the opposite direction to the gradient. In this sense, we do not need to "see" the whole function. We just need to* locally *compute the gradient ∇f, analogous to feeling the ground beneath.*

Example 7 (Engineering shape optimization: minimizing drag with Navier–Stokes flow of fluid past a rigid body, [Wal15], pp. 3–5)**.** *A shape functional representing the drag:*

$$J_{\mathrm{d}}(\Omega) \coloneqq -\mathbf{u}_{\mathrm{out}}\cdot\int_{\Gamma_0}\boldsymbol{\sigma}(\mathbf{u},p)\mathbf{n}\,\mathrm{d}\Gamma = \frac{2}{\mathrm{Re}}\int_\Omega |\boldsymbol{\varepsilon}(\mathbf{u})|^2\,\mathrm{d}\mathbf{x} \geq 0, \tag{20}$$

which physically represents the net force that must be applied to Ω_{B} to keep it stationary while being acted upon by the imposed flow field & represents the total amount of viscous dissipation of energy (per unit of time) in the fluid domain Ω. Using the machinery of shape perturbations, $\delta J_{\mathrm{d}}(\Omega;V)$ indicates how J_{d} changes when we perturb Ω in the direction V. Hence, we can use this information to change Ω in small steps so as to slowly deform Ω into a shape that has better (lower) drag characteristics. A numerical computation: 2 large vortices appear behind the body, which indicate a large amount of viscous dissipation, i.e., large drag. The optimization process then computes $\delta J_{\mathrm{d}}(\Omega^0;V)$ for many different choices of V & chooses the one that drives down J_{d} the most. This choice of V is used to deform Γ_{B}^0 into a new shape Γ_{B}^1 at iteration 1, with only a small difference between Γ_{B}^0 & Γ_{B}^1. This process is repeated many times. Note how the vortices are eliminated by the more slender shape.

- Condition (V)***
- Family of transformations $\{T_s : 0 \leq s \leq \tau\}$.
- Perturbed domain $\Omega_s = \Omega_s(V) = T_s(V)(\Omega)$.
- Assume that the velocity field V satisfies V_D and, in addition, $V \in C^0([0,\tau];C^1_{\mathrm{loc}}(\mathbb{R}^d,\mathbb{R}^d))$ and $\tau > 0$ is small enough such that the Jacobian J_s is strictly positive: $J_s(X) \coloneqq \det DT_s(X) > 0$, $\forall s \in [0,\tau]$, where $DT_s(X)$ is the *Jacobian matrix* of the transformation $T_s = T_s(V)$ associated with the velocity vector field V.

11.1 Domain Integrals

- Given $\varphi \in W^{1,1}_{\mathrm{loc}}(\mathbb{R}^d)$, consider for $s \in [0,\tau]$ the volume integral

$$J(\Omega_s(V)) \coloneqq \int_{\Omega_s(V)}\varphi\,\mathrm{d}\mathbf{x} = \int_\Omega \varphi\circ T_s J_s\,\mathrm{d}\mathbf{x}. \tag{21}$$

$$dJ(\Omega;V) = \frac{d}{ds}J(\Omega_s(V))|_{s=0} = \int_\Omega \nabla\varphi\cdot V(0) + \varphi\,\mathrm{div}\,V(0)\,\mathrm{d}\mathbf{x} = \int_\Omega \mathrm{div}(\varphi V(0))\,\mathrm{d}\mathbf{x}. \tag{22}$$

If Ω has a Lipschitzian boundary, by Stokes's theorem:

$$dJ(\Omega;V) = \int_\Gamma \varphi V(0)\cdot\mathbf{n}\,\mathrm{d}\Gamma. \tag{23}$$

Theorem 7 ([DZ11], Thm. 4.1, pp. 482–483)**.** *Let $\varphi \in W^{1,1}_{\mathrm{loc}}(\mathbb{R}^d)$. Assume that the vector field $V = \{V(s) : 0 \leq s \leq \tau\}$ satisfies condition (V).*

(i) For each $s \in [0,\tau]$ the map $\varphi \mapsto \varphi\circ T_s : W^{1,1}_{\mathrm{loc}}(\mathbb{R}^d) \to W^{1,1}_{\mathrm{loc}}(\mathbb{R}^d)$ and its inverse are both locally Lipschitzian and $\nabla(\varphi\circ T_s) = {}^*DT_s\nabla\varphi\circ T_s$.

(ii) If $V \in C^0([0,\tau]; C^1_{\mathrm{loc}}(\mathbb{R}^d,\mathbb{R}^d))$, then the map $s \mapsto J_s : [0,\tau] \to C^0_{\mathrm{loc}}(\mathbb{R}^d)$ is differentiable and

$$\frac{dJ_s}{ds} = [\nabla \cdot V(s)] \circ T_s J_s \in C^0_{\mathrm{loc}}(\mathbb{R}^d). \tag{24}$$

Hence the map $s \mapsto J_s$ belongs to $C^1([0,\tau]; C^0_{\mathrm{loc}}(\mathbb{R}^d))$.

$$\frac{d}{ds} DT_s(X) = DV(s, T_s(X)) DT_s(X),\ DT_0(X) = I, \tag{25}$$

$$\frac{d}{ds} \det DT_s(X) = \operatorname{tr} DV(s, T_s(X)) \det DT_s(X) = \nabla \cdot V(s, T_s(X)) \det DT_s(X),\ \det DT_0(X) = 1. \tag{26}$$

Theorem 8 ([DZ11], Thm. 4.2, pp. 483–484)**.** *Assume that there exists $\tau > 0$ such that the velocity field $V(t)$ satisfies conditions (V) and $V \in C^0([0,\tau]; C^1_{\mathrm{loc}}(\mathbb{R}^d,\mathbb{R}^d))$. Given a function $\varphi \in C(0,\tau; W^{1,1}_{\mathrm{loc}}(\mathbb{R}^d)) \cap C^1(0,\tau; L^1_{\mathrm{loc}}(\mathbb{R}^d))$ and a bounded measurable domain Ω with boundary Γ, the semiderivative of the function $J_V(s) := \int_{\Omega_s(V)} \varphi(s)\,\mathrm{d}\mathbf{x}$ at $s = 0$ is given by*

$$dJ_V(0) = \int_\Omega \varphi'(0) + \operatorname{div}(\varphi(0)V(0))\,\mathrm{d}\mathbf{x}, \tag{27}$$

where $\varphi(0)(\mathbf{x}) := \varphi(0,\mathbf{x})$ and $\varphi'(0)(\mathbf{x}) := \partial_t \varphi(0,\mathbf{x})$. In addition, Ω is an open domain with a Lipschitzian boundary Γ, then

$$dJ_V(0) = \int_\Omega \varphi'(0)\,\mathrm{d}\mathbf{x} + \int_\Gamma \varphi(0)V(0)\cdot\mathbf{n}\,\mathrm{d}\mathbf{x}. \tag{28}$$

11.2 Boundary Integrals

11.3 Material derivatives

Let $\Omega \subset D$ be a bounded domain,

12 Topology Optimization – Tối Ưu Tôpô

13 Wikipedia

13.1 Wikipedia/adjoint

"In mathematics, the term *adjoint* applies in several situations. Several of these share a similar formalism: if A is adjoint to B, then there is typically some formula of the type $(Ax, y) = (x, By)$. Specifically, *adjoint* or *adjunction* may mean:

1. Adjoint of a linear map, also called its transpose in case of matrices
2. Hermitian adjoint (adjoint of a linear operator) in functional analysis
3. Adjoint endomorphism of a Lie algebra
4. Adjoint endomorphism of a Lie algebra
5. Adjoint functors in category theory
6. Adjunction (field theory)
7. Adjunction formula (algebraic geometry)
8. Adjunction space in topology
9. Conjuate transpose of a matrix in linear algebra
10. Adjugate matrix, related to its inverse
11. Adjoint equation
12. The upper & lower adjoints of a Galois connection in order theory
13. The adjoint of a differential operator with general polynomial coefficients
14. Kleisli adjunction
15. Monoidal adjunction
16. Qhillen adjunction
17. Axiom of adjunction in set theory
18. Adjunction (rule of inference)" – Wikipedia/adjoint

13.2 Wikipedia/dynamic programming

"Finding shortest path in a graph using optimal substructure; a straight line indicates a single edge; a wavy line indicates a shortest path between 2 vertices it connects (among other paths, not shown, sharing same 2 vertices); bold line is overall shortest path from start to goal. *Dynamic programming* is both a mathematical optimization method & an algorithmic paradigm. Method was developed by Richard Bellman in 1950s & has found applications in numerous fields, from aerospace engineering to economics.

In both contexts it refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner. While some decision problems cannot be taken apart this way, decisions that span several points in time do often break apart recursively. Likewise, in CS, if a problem, if a problem can be solved optimally by breaking it into sub-problems & then recursively finding optimal solutions to sub-problems, then it is said to have optimal substructure.

If sub-problems can be nested recursively inside larger problems, so that dynamic programming methods are applicable, then there is a relation between value of larger problem & values of sub-problems. In optimization literature this relationship is called Bellmann equation.

13.2.1 Overview

1. **Mathematical optimization.** In terms of mathematical optimization, dynamic programming usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time.

   This is done by defining a sequence of *value functions* $V_1, \dots, V_n$ taking y as an argument representing state of system at times i from 1 to n.

   Def of $V_n(y)$ is value obtained in state y at last time n.

   Values V_i at earlier times $i = n-1, n-2, \dots, 2, 1$ can be found by working backwards, using a recursive relationship called Bellmann equation.

   For $i = 2, \dots, n$, V_{i-1} at any state y is calculated from V_i by maximizing a simple function (usually sum) of gain from a decision at time $i-1$ & function V_i at new state of system if this decision is made.

   Since V_i has already been calculated for needed states, above operation yields V_{i-1} for those states.

   Finally, V_1 at initial state of system is value of optimal solution. Optimal values of decision variables can be recovered, 1 by 1, by tracking back calculations already performed.

2. **Control theory.** In control theory, a typical problem: find an admissible control $\mathbf{u}^*$ which causes system $\dot{\mathbf{x}}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), t)$ to follow an admissible trajectory $\mathbf{x}^*$ on a continuous time interval $t_0 \le t \le t_1$ that minimizes a cost function

   $$J = b(\mathbf{x}(t_1), t_1) + \int_{t_0}^{t_1} f(\mathbf{x}(t), \mathbf{u}(t), t)\, \mathrm{d}t.$$

   Solution to this problem is an optimal control law or policy $\mathbf{u}^* = h(\mathbf{x}(t), t)$, which produces an optimal trajectory $\mathbf{x}^*$ & a cost-to-go function J^*. Latter obeys fundamental equation of dynamic programming:

   $$-J_t^* = \min_{\mathbf{u}} \{ f(\mathbf{x}(t), \mathbf{u}(t), t) + J_x; *^\top \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), t) \}$$

   a PDE known as Hamilton–Jacobi–Bellman equation, in which

   $$J_x^* = \frac{\partial J^*}{\partial \mathbf{x}} = [\partial_{x_1} J^*, \dots, \partial_{x_n} J^*]^\top, \;\; J_t^* = \partial_t J^*.$$

   One finds: minimizing $\mathbf{u}$ in terms of $t, \mathbf{x}$, & unknown function J_x^* & then substitutes result into Hamilton–Jacobi–Bellman equation to get PDE to be solved with boundary condition $J(t_1) = b(\mathbf{x}(t_1), t_1)$. In practice, this generally requires numerical techniques for some discrete approximation to exact optimization relationship.

   Alternatively, continuous process can be approximated by a discrete system, which leads to a following recurrence relation analog to Hamilton–Jacobi–Bellman equation:

   $$J_k^*(\mathbf{x}_{n-k}) = \min_{\mathbf{u}_{n-k}} \left\{ \hat{f}(\mathbf{x}_{n-k}, \mathbf{u}_{n-k}) + J_{k-1}^*(\hat{\mathbf{g}}(\mathbf{x}_{n-k}, \mathbf{u}_{n-k})) \right\}$$

   at kth state of n equally spaced discrete time intervals, & where $\hat{f}, \hat{\mathbf{g}}$ denote discrete approximations to $f, \mathbf{g}$. This functional equation is known as Bellman equation, which can be solved for an exact solution of discrete approximation of optimization equation.

- **Example from economics: Ramsey's problem of optimal saving.** See also: Ramsey–Cass–Koopmans model. In economics, objective is generally to maximize (rather than minimize) some dynamic social welfare function. In Ramsey's problem, this function relates amounts of consumption to levels of utility. Loosely speaking, planner faces trade-off between contemporaneous consumption & future consumption (via investment in capital stock used in production), known as intertemporal choice. Future consumption is discounted at a constant rate $\beta \in (0, 1)$. A discrete approximation to transition equation of capital is given by

  $$k_{t+1} = \hat{g}(k_t, c_t) = f(k_t) - c_t,$$

where c is consumption, k is capital, f: a production function satisfying Inada conditions. An initial capital stock $k_0 > 0$ is assumed.

Let c_t be consumption in period t, & assume consumption yields utility $u(c_t) = \ln c_t$ as long as consumer lives. Assume consumer is impatient, so that he discounts future utility by a factor b each period, where $0 < b < 1$. Let k_t be capital in period t. Assume initial capital is a given amount $k_0 > 0$, & suppose: this period's capital & consumption determine next period's capital as $k_{t+1} = Ak_t^a - c_t$, where A: a positive constant & $0 < a < 1$. Assume capital cannot be negative. Then consumer's decision problem can be written as follows:

$$\max \sum_{t=0}^{T} b^t \ln c_t \text{ subject to } k_{t+1} = Ak_t^a - c_t \geq 0, \ \forall t = 0, 1, \ldots, T.$$

Written this way, problem looks complicated, because it involves solving for all choice variables $c_0, c_1, \ldots, c_T$. (Capital k_0 is not a choice variable – consumer's initial capital is taken as given.)

DP approach to solve this problem involves breaking it apart into a sequence of smaller decisions. To do so, define a sequence of *value functions* $V_t(k)$, for $t = 0, 1, \ldots, T, T+1$ which represent value of having any amount of capital k at each time t. There is (by assumption) no utility from having capital after death, $V_{T+1}(k) = 0$.

Value of any quantity of capital at any previous time can be calculated by backward induction using Bellman equation. In this problem, for each $t = 0, 1, \ldots, T$, Bellman equation is

$$V_t(k_t) = \max(\ln c_t + bV_{t+1}(k_{t+1})) \text{ subject to } k_{t+1} = Ak_t^a - c_t \geq 0.$$

This problem is much simpler than the once wrote down before, because it involves only 2 decision variables, c_t & k_{t+1}. Intuitively, instead of choosing his whole lifetime plan at birth, consumer can take things 1 step at a time. At time t, his current capital k_t is given, & he only needs to choose current consumption c_t & saving k_{t+1}.

To actually solve this problem, work backwards. For simplicity, current level of capital is denoted as k. $V_{T+1}(k)$ is already known, so using Bellman equation once can calculate $V_T(k)$, & so on until get to $V_0(k)$, which is *value* of initial decision problem for whole lifetime. I.e., once know $V_{T-j+1}(k)$, can calculate $V_{T_j}(k)$, which is maximum of $\ln c_{T-j} + bV_{T-j+1}(Ak^a - c_{T-j})$, where c_{T-j} is choice variable & $Ak^a - c_{T-j} \geq 0$.

Working backwards, it can be shown: value function at time $t = T - j$ is

$$V_{T-j}(k) = a \sum_{i=0}^{j} a^i b^i \ln k + v_{T-j},$$

where each v_{T-j} is a constant, & optimal amount to consume at time $t = T - j$ is

$$c_{T-j}(k) = \frac{1}{\sum_{i=0}^{j} a^i b^i} Ak^a.$$

See: optimal to consume a larger fraction of current wealth as one gets older, finally consuming all remaining wealth in period T, the last period of life.

3. **CS.** There are 2 key attributes that a problem must have in order for DP to be applicable: optimal substructure & overlapping sub-problems. If a problem can be solved by combining optimal solutions to *non-overlapping* sub-problems, strategy is called "divide & conquer" instead. This is why merge sort & quick sort are not classified as DP problems.

   Optimal substructure means: solution to a given optimization problem can be obtained by combination of optimal solutions to its sub-problems. Such optimal substructures are usually described by means of recursion. E.g., given a graph $G = (V, E)$, shortest path p from a vertex u to a vertex v exhibits optimal substructure: take any intermediate vertex w on this shortest path p. If p is truly shortest path, then it can be split into sub-paths p_1 from u to w & p_2 from w to v s.t. these, in turn, are indeed shortest paths between corresponding vertices (by simple cut-&-paste argument described in *Introduction to Algorithms*). Hence, one can easily formulate solution for finding shortest paths in a recursive manner, which is what Bellman–Ford algorithm or Floyd–Warshall algorithm does.

   Overlapping sub-problems means: space of sub-problems must be small, i.e., any recursive algorithm solving problem should solve same sub-problems over & over, rather than generating new sub-problems. E.g., consider recursive formulation for generating Fibonacci sequence $F_i = F_{i-1} + F_{i-2}$, with base case $F_1 = F_2 = 1$. Then $F_{43} = F_{42} + F_{41}$, & $F_{42} = F_{41} + F_{40}$. Now F_{41} is being solved in recursive sub-trees of both F_{43} as well as F_{42}. Even though total number of sub-problems is actually small (only 43 of them), end up solving same problems over & over if adopt a naive recursive solution e.g. this. DP takes account of this fact & solves each sub-problem only once.

   Fig. Subproblem graph for Fibonacci sequence. Fact: it is not a tree indicates overlapping subproblems. This can be achieved in either of 2 ways:

   - *Top-down approach*: This is direct fall-out of recursive formulation of any problem. if solution to any problem can be formulated recursively using solution to its sub-problems, & if its sub-problems are overlapping, then one can easily memoize or store solutions to sub-problems in a table (often an array or hastable in practice). Whenever attempt to solve a new sub-problem, 1st check table to see if already solved. If a solution has been recorded, can use it directly, otherwise solve sub-problem & add its solution to table.

- *Bottom-up approach*: Once formulate solution to a problem recursively as in terms of its sub-problems, can try reformulating problem in a bottom-up fashion: try solving sub-problems 1st & use their solutions to build-on & arrive at solutions to bigger sub-problems. This is also usually done in a tabular form by iteratively generating solutions to bigger & bigger sub-problems by using solutions to small sub-problems. E.g., if already know values of F_{41} & F_{40}, can directly calculate value of F_{42}.

Some programming languagues can automatically memoize result of a function call with a particular set of arguments, in order to speed up call-by-name evaluation (this mechanism is referred to as *call-by-need*). Some languages make it possible portably (e.g., Scheme, Common Lisp, Perl, & D). Some languages have automatic memoization built in, e.g. tabled Prolog & J, which supports memoization with *M.* adverb. In any case, this is only possible for a referentially transparent function. Memoization is also encountered as an easily accessible design pattern within term-rewrite based languages e.g. Wolfram Language.

4. **Bioinformatics.** DP is widely used in bioinformatics for tasks e.g. sequence alignment, protein folding, RNA structure prediction & protein-DNA binding. 1st DP algorithms for protein-DNA binding were developed in 1970s independently by Charles DeLisi in US & by Georgii Gurskii & Alexander Zasedatelev in Soviet Union. Recently these algorithms have become very popular in bioinformatics & computational biology, particularly in studies of nucleosome positioning & transcription factor binding.

13.2.2 Examples: computer algorithms

1. **Dijkstra's algorithm for shortest path problem.** From a DP point of view, Dijkstra's algorithm for shortest path problem is a successive approximation scheme that solves DP functional equation for shortest path problem by *Reaching* method.

   In fact, Dijkstra's explanation of logic behind algorithm, namely

   Problem 4. *Find path of minimum total length between 2 given nodes P, Q.*

   Use fact: if R is a node on minimal path from P to Q, knowledge of latter implies knowledge of minimal path from P to R.

   is a paraphrasing of Bellman's famous Principle of Optimality in context of shortest path problem.

2. **Fibonacci sequence.** Using DP in calculation of nth member of Fibonacci sequence improves its performance greatly. Here is a naive implementation, based directly on mathematical def:

```
function fib(n)
    if n <= 1 return n
    return fib(n - 1) + fib(n - 2)
```

   Notice if call, say, `fib(5)`, produce a call tree that calls function on same value many different times [...]. In particular, `fib(2)` was calculated 3 times from scratch. In larger examples, many more values of `fib`, or *subproblems*, are recalculated, leading to an exponential time algorithm.

   Now, suppose have a simple map object m which maps each value of `fib` that has already been calculated to its result, & modify our function to use it & update it. Resulting function requires only $O(n)$ time instead of exponential time (but requires $O(n)$ space):

```
var m := map(0 -> 0, 1 -> 1)
function fib(n)
    if key n is not in map m
        m[n] := fib(n - 1) + fib(n - 2)
    return m[n]
```

   This technique of saving values that have already been calculated is called memoization; this is top-down approach, since 1st break problem into subproblems & then calculate & store values.

   In *bottom-up* approach, calculate smaller values of `fib` 1st, then build larger values from them. This method also uses $O(n)$ time since it contains a loop that repeats $n-1$ times, but it only takes constant $O(1)$ space, in contrast to top-down approach which requires $O(n)$ space to store map.

```
function fib(n)
    if n = 0
        return 0
    else
        var previousFib := 0, currentFib := 1
        repeat n - 1 times // loop is skipped if n = 1
            var newFib := previousFib + currentFib
```

```
            previousFib := currentFib
            currentFib := newFib
        return currentFib
```

In both examples, only calculate `fib(2)` 1 time, & then use it to calculate both `fib(4)` & `fib(3)`, instead of computing it every time either of them is evaluated.

3. **A type of balanced 0–1 matrix.** Consider problem of assigning values, either 0 or 1, to positions of an $n \times n$ matrix, with n even, so that each row & each column contains exactly $\frac{n}{2}$ 0s & $\frac{n}{2}$ 1s. Ask how many different assignments there are for a given n. E.g., when $n = 4$, 5 possible solutions are [...] There are at least 3 possible approaches: brute force, backtracking, & DP.

   Brute force consists of checking all assignments of 0s & 1s & counting those that have balanced rows & columns ($\frac{n}{2}$ 0s & $\frac{n}{2}$ 1s). As there are 2^{n^2} possible assignments & $\binom{n}{\frac{n}{2}}^2$ sensible assignments, this strategy is not practical except maybe up to $n = 6$.

   Backtracking for this problem consists of choosing some order of matrix elements & recursively placing 1s or 0s, while checking: in every row & column number of elements that have not been assigned plus number of 1s or 0s are both at least $\frac{n}{2}$. While more sophisticated than brute force, this approach will visit every solution once, making it impractical for $n > 6$, since number of solutions is already 116,963,796,250 for $n = 8$.

   DP makes it possible to count number of solutions without visiting them all. Imagine backtracking values for 1st row – what information would we require about remaining rows, in order to be able to accurately count solutions obtained for each 1st row value? Consider $k \times n$ boards, where $1 \leq k \leq n$, whose k rows contain $\frac{n}{2}$ 0s & $\frac{n}{2}$ 1s. Function f to which memoization is applied maps vectors of n pairs of integers to number of admissible boards (solutions). There is 1 pair for each column, & its 2 components indicate respectively number of 0s & 1s that have yet to be placed in that column. Seek value of $f\left(\left(\frac{n}{2}, \frac{n}{2}\right), \left(\frac{n}{2}, \frac{n}{2}\right), \ldots, \left(\frac{n}{2}, \frac{n}{2}\right)\right)$ (n arguments or 1 vector of n elements). Process of subproblem creation involves iterating over every 1 of $\binom{n}{\frac{n}{2}}$ possible assignments for top row of board, & going through every column, subtracting 1 from appropriate element of pair for that column, depending on whether assignment for top row contained a 0 or a 1 at that position. If any 1 of results is negative, then assignment is invalid & does not contribute to set of solutions (recursion stops). Otherwise, have an assignment for top row of $k \times n$ board & recursively compute number of solutions to remaining $(k-1) \times n$ board, adding numbers of solutions for every admissible assignment of top row & returning sum, which is being memoized. Base case is trivial subproblem, which occurs for a $1 \times n$ board. Number of solutions for this board is either 0 or 1, depending on whether vector is a permutation of $\frac{n}{2}$ $(0, 1)$ & $\frac{n}{2}$ $(1, 0)$ pairs or not.

   E.g., in 1st 2 boards shown above sequences of vectors would be [...] The number of solutions (sequence) A058527 in OEIS) is $1, 2, 90, 297200, 116963796250, 67362182874304607 52, \ldots$ Links to MAPLE implementation of DP approach may be found among external links.

4. **Checkerboard.** Consider a checkerboard with $n \times n$ squares & a cost function `c(i, j)` which returns a cost associated with square `(i,j)` i: row, j: column. E.g., on a [5×5 checkerboard]. Say there was a checker that could start at any square on 1st rank (i.e., row) & wanted to know shortest path (sum of minimum costs at each visited rank) to get to last rank; assuming checker could move only diagonally left forward, diagonally right forward, or straight forward. I.e., a checker on $(1, 3)$ can move to $(2, 2), (2, 3)$, or $(2, 4)$. This problem exhibits *optimal substructure*. I.e., solution to entire problem relies on solutions to subproblems. Define a function $q(i, j) \coloneqq$ minimum cost to reach square (i, j). Starting at rank n & descending to rank 1, compute value of this function for all squares at each successive rank. Picking square that holds minimum value at each rank gives shortest path between rank n & rank 1.

   Function $q(i, j) =$ minimum cost to get to any of 3 squares below it (since those are the only squares that can reach it) plus $c(i, j)$. E.g.: $q(A) = \min(q(B), q(C), q(D)) + c(A)$. Define $q(i, j)$ in somewhat more general terms:

$$q(i,j) = \begin{cases} \infty & \text{if } j < 1 \text{ or } j > n, \\ c(i,j) & \text{if } i = 1, \\ \min(q(i-1,j-1), q(i-1,j), q(i-1,j+1)) + c(i,j) & \text{otherwise.} \end{cases}$$

   1st line of this equation deals with a board modeled as squares indexed on 1 at lowest bound & n at highest bound. 2nd line specifies what happens at 1st rank; providing a base case. 3rd line, recursion, is important part. It represents A, B, C, D terms in example. From this def can derive straightforward recursive code for $q(i, j)$. In following pseudo code, n: size of board, $c(i, j)$: cost function, min() returns minimum of a number of values.

```
function minCost(i, j)
    if j < 1 or j > n
        return infinity
    else if i = 1
        return c(i, j)
    else
        return min(minCost(i - 1, j - 1), minCost(i - 1, j), minCost(i - 1, j + 1)) + c(i, j)
```

This function only computes path cost, not actual path. Discuss actual path. This, like Fibonacci-numbers example, is horribly slow because it too exhibits *overlapping subproblems* attribute. I.e., it recomputes same path costs over & over. However, can compute it much faster in a bottom-up fashion if store path costs in a 2D array `q[i, j]` rather than using a function. This avoids recomputation; all values needed for array `q[i, j]` are computed ahead of time only once. Precomputed values for (i, j) are simply looked up whenever needed.

Also need to know what actual shortest path is. To do this, use another array `p[i, j]`; a *predecessor array*. This array records path to any square s. Predecessor of s is modeled as an offset relative to index (in `q[i, j]`) of precomputed path cost of s. To reconstruct complete path, lookup predecessor of s, then predecessor of that square, then predecessor of that square, & so on recursively, until reach starting square. Consider following pseudocode:

```
function computeShortestPathArrays()
    for x from 1 to n
        q[1, x] := c(1, x)
    for y from 1 to n
        q[y, 0] := infinity
        q[y, n + 1] := infinity
    for y from 2 to n
        for x from 1 to n
            m := min(q[y-1, x-1], q[y-1, x], q[y-1, x+1])
            q[y, x] := m + c(y, x)
            if m = q[y-1, x-1]
                p[y, x] := -1
            else if m = q[y-1, x]
                p[y, x] :=  0
            else
                p[y, x] :=  1
```

Now the rest is a simple matter of finding minimum & printing it.

```
function computeShortestPath()
    computeShortestPathArrays()
    minIndex := 1
    min := q[n, 1]
    for i from 2 to n
        if q[n, i] < min
            minIndex := i
            min := q[n, i]
    printPath(n, minIndex)

function printPath(y, x)
    print(x)
    print("<-")
    if y = 2
        print(x + p[y, x])
    else
        printPath(y-1, x + p[y, x])
```

5. **Sequence alignment.** In genetics, sequence alignment is an important application where DP is essential. Typically, problem consists of transforming 1 sequence into another using edit operations that replace, insert, or remove an element. Each operation has an associated cost, & goal: find sequence of edits with lowest total cost.

   Problem can be stated naturally as a recursion, a sequence A is optimally edited into a sequence B by either:

   (a) inserting 1st character of B, & performing an optimal alignment of A & tail of B

   (b) deleting 1st character of A, & performing optimal alignment of tail of A & B

   (c) replacing 1st character of A with 1st character of B, & performing optimal alignments of tails of A & B.

   Partial alignments can be tabulated in a matrix, where cell(i, j) contains cost of optimal alignment of A[1..i] to B[1..j]. Cost in cell(i, j) can be calculated by adding cost of relevant operations to cost of its neighboring cells, & selecting optimum.

   Different variants exist, see Smith–Waterman algorithm & Needleman–Wunsch algorithm.

6. **Tower of Hanoi puzzle.** Tower of Hanoi or Towers of Hanoi is a mathematical game or puzzle. It consists of 3 rods, & a number of disks of different sizes which can slide onto any rod. Puzzle starts with disks in a neat stack in ascending order of size on 1 rod, smallest at top, thus making a conical shape.

   Objective of puzzle: move entire stack to another rod, obey rules:

   (a) Only 1 disk may be moved at a time.

   (b) Each move consists of taking upper disk from 1 of rods & sliding it onto another rod, on top of other disks that may already be present on that rod.

   (c) No disk may be placed on top of a smaller disk.

   DP solution consists of solving functional equation

$$S(n,h,t) = S(n-1,h,not(h,t)); S(1,h,t); S(n-1,not(h,t),t)$$

   where n denotes number of disks to be moved, h denotes home rod, t denotes target rod, $not(h,t)$ denotes 3rd rod (neither h nor t), ";" denotes concatenation, & $S(n,h,t) :=$ solution to a problem consisting of n disks that are to be moved from rod h to rod t. For $n = 1$ problem is trivial, namely $S(1,h,t) =$ "move a disk from rod h to rod t" (there is only 1 disk left).

   Number of moves required by this solution is $2^n - 1$. If objective is to *maximize* number of moves (without cycling) then DP functional equation is slightly more complicated & $3^n - 1$ moves are required.

7. **Egg dropping puzzle.** A description of instance of this famous puzzle involving $N = 2$ eggs & a building with $H = 36$ floors:

   Problem 5. *Suppose wish to know which stories in a 36-story building are safe to drop eggs from, & which will cause eggs to break on landing (using U.S. English terminology, in which 1st floor is at ground level). Make a few assumptions:*

   - *An egg that survives a fall can be used again.*
   - *A broken egg must be discarded.*
   - *Effect of a fall is same for all eggs.*
   - *If an egg breaks when dropped, then it would break if dropped from a higher window.*
   - *If an egg survives a fall, then it would survive a shorter fall.*
   - *It is not ruled out: 1st-floor windows break eggs, nor is it ruled out that eggs can survive 36th-floor windows.*

   If only 1 egg is available & wish to be sure of obtaining right result, experiment can be carried out in only 1 way. Drop egg from 1st-floor window; if it survives, drop it from 2nd-floor window. Continue upward until it breaks. In worst case, this method may require 36 droppings. Suppose 2 eggs are available. What is lowest number of egg-droppings that is guaranteed to work in all cases?

   To derive a DP functional equation for this puzzle, let *state* of DP model be a pair $s = (n,k)$, where $n =$ number of test eggs available, $n = 0, 1, , \ldots, N-1$, $k =$ number of (consecutive) floors yet to be tested, $k = 0, 1, \ldots, H-1$. E.g., $s = (2,6)$ indicates: 2 test eggs are available & 6 (consecutive) floors ar yet to be tested. Initial state of process is $s = (N,H)$ where N denotes number of test eggs available at commencement of experiment. Process terminates either when there are no more test eggs $n = 0$ or when $k = 0$, whichever occurs 1st. If termination occurs at state $s = (0,k)$ & $k > 0$, then test failed. Now let $W(n,k) =$ minimum number of trials required to identify value of critical floor under worst-case scenario given that process is in state $s = (n,k)$. Then it can be shown: $W(n,k) = 1 + \min\{\max(W(n-1,x-1), W(n,k-x)) : x = 1, \ldots, k\}$ with $W(n,0) = 0$, $\forall n \in \mathbb{N}^\star$ & $W(1,k) = k$ for all k. Easy to solve this equation iteratively by systematically increasing values of n,k.

   - **Faster DP solution using a different parametrization.** Notice above solution takes $O(nk^2)$ time with a DP solution. This can be improved to $O(nk \log k)$ time by binary searching on optiaml x in above recurrence, since $W(n-1,x-1)$ is increasing in x while $W(n,k-x)$ is decreasing in x, thus a local minimum of $\max(W(n-1,x-1), W(n,k-x))$ is a global minimum. Also, by storing optimal x for each cell in DP table & referring to its value for prev cell, optimal x for each cell can be found in constant time, improving it to $O(nk)$ time. However, there is an even faster solution that involves a different parametrization of problem: Let k: total number of floors s.t. eggs break when dropped from kth floor (example above is equivalent to taking $k = 37$). Let m: minimum floor from which egg must be dropped to be broken. Let $f(t,n)$: maximum number of values of m that are distinguishable using t tries & n eggs. Then $f(t,0) = f(0,n) = 1$ $\forall t, n \geq 0$. Let a: floor from which 1st egg is dropped in optimal strategy. If 1st egg broke, m is from 1 to a & distinguishable using at most $t-1$ tries & $n-1$ eggs. If 1st eggs did not break, m is from $a+1$ to k & distinguishable using $t-1$ tries & n eggs. Therefore, $f(t,n) = f(t-1,n-1) + f(t-1,n)$. Then problem is equivalent to finding minimum x s.t. $f(x,n) \geq k$. To do so, could compute $\{f(t,i) : 0 \leq i \leq n\}$ in order of increasing t, which would take $O(nx)$ time. Thus, if separately handle case of $n = 1$, algorithm would take $O(n\sqrt{k})$ time. But recurrence relation can in fact be solved, giving $f(t,n) = \sum_{i=0}^{n} \binom{t}{i}$, which can be computed in $O(n)$ time using identity $\binom{t}{i+1} = \binom{t}{i}\frac{t-i}{i+1}$, $\forall i \geq 0$. Since $f(t,n) \leq f(t+1,n)$, $\forall t \geq 0$, can binary search on t to find x, giving an $O(n \log k)$ algorithm.

8. **Matrix chain multiplication.** Wikipedia/matrix chain multiplication. Matrix chain multiplication is a well-known example that demonstrates utility of DP. E.g., engineering applications often have to multiply a chain of matrices. It is not surprising to find matrices of large dimensions, e.g. 100×100. Therefore, our task: multiply matrices $A_1, \ldots, A_n$. Matrix multiplication is not commutative, but is associative; & can multiply only 2 matrices at a time. So, can multiply this chain of matrices in many different ways, e.g.: [. . .] There are numerous ways to multiply this chain of matrices. They will all produce same final result, however they will take more or less time to compute, based on which particular matrices are multiplied. If matrix A has dimensions $m \times n$ & matrix B has dimension $n \times q$, then matrix $A = AB$ will have dimensions $m \times Q$, & will require mnq scalar multiplications (using a simplistic matrix multiplication algorithm for purposes of illustration).

   E.g., multiply matrices A, B, C. Assume: their dimensions are $m \times n, n \times p, p \times s$, resp. Matrix ABC will be of size $m \times s$ & can be calculated in 2 ways shown:

   (a) $A(BC)$: this order of matrix multiplication will require $nps + mns$ scalar multiplications.

   (b) $(AB)C$: this order of matrix multiplication will require $mnp + mps$ scalar calculations.

   Assume $m = 10, n = 100, p = 10, s = 1000$. So, 1st way to multiply chain will require $10^6 + 10^6$ calculations. 2nd way will require only $10000 + 100000$ calculations. Obviously, 2nd way is faster, & should multiply matrices using that arrangement of parenthesis.

   Therefore, our conclusion: order of parenthesis matters, & our task: find optimal order of parenthesis.

   At this point, have several choices, 1 of which: design a DP algorithm that will split problem into overlapping problems & calculate optimal arrangement of parenthesis. DP solution is presented below.

   Call $m[i, j]$ minimum number of scalar multiplications needed to multiply a chain of matrices from matrix i to matrix j, i.e., $A_i \cdots A_j$, i.e., $i \leq j$. Split chain at some matrix k, s.t. $i \leq k \leq j$, & try to find out which combination produces minimum $m[i, j]$. Formula is:

```
if i = j, m[i,j]= 0
if i < j, m[i,j]= min over all possible values of k (m[i,k]+m[k+1,j] + p[i-1]*p[k]*p[j]
```

   where k ranges from i to $j - 1$, p_{i-1}: row dimension of matrix i, p_k: column dimension of matrix k, p_j: column dimension of matrix j. This formula can be coded as shown below, where input parameter "chain" is chain of matrices, i.e., $A_1, \ldots, A_n$:

```
function OptimalMatrixChainParenthesis(chain)
    n = length(chain)
    for i = 1, n
        m[i,i] = 0    // Since it takes no calculations to multiply one matrix
    for len = 2, n
        for i = 1, n - len + 1
            j = i + len - 1
            m[i,j] = infinity      // So that the first calculation updates
            for k = i, j - 1
                q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
                if q < m[i, j]     // The new order of parentheses is better than what we had
                    m[i, j] = q    // Update
                    s[i, j] = k    // Record which k to split on, i.e. where to place the parenthesis
```

   So far, have calculated values for all possible $m[i, j]$, minimum number of calculations to multiply a chain from matrix i to matrix j, & have recorded corresponding "split point" $s[i, j]$. E.g., if multiplying chain $A_1A_2A_3A_4$, & it turns out $m[1, 3] = 100$ & $s[1, 3] = 2$, i.e. optimal placement of parenthesis for matrices 1 to 3 is $(A_1A_2)A_3$ & to multiply those matrices will require 100 scalar calculations.

   This algorithm will produce "tables" $m[\cdot, \cdot], s[\cdot, \cdot]$ that will have entries for all possible values of i, j. Final solution for entire chain is $m[1, n]$, with corresponding split at $s[1, n]$. Unraveling solution will be recursive, starting from top & continuing until reach base case, i.e., multiplication of single matrices.

   Therefore, next step: actually split chain, i.e., place parenthesis where they (optimally) belong. For this purpose could use algorithm:

```
function PrintOptimalParenthesis(s, i, j)
    if i = j
        print "A"i
    else
        print "("
        PrintOptimalParenthesis(s, i, s[i, j])
```

```
        PrintOptimalParenthesis(s, s[i, j] + 1, j)
        print ")"
```

Of course, this algorithm is not useful for actual multiplication. This algorithm is just a user-friendly way to see what result looks like. To actually multiply matrices using proper splits, need algorithm:

```
function MatrixChainMultiply(chain from 1 to n)       // returns the final matrix, i.e. A1×A2×... ×An
    OptimalMatrixChainParenthesis(chain from 1 to n)   // this will produce s[ . ] and m[ . ] "tables"
    OptimalMatrixMultiplication(s, chain from 1 to n)  // actually multiply
function OptimalMatrixMultiplication(s, i, j)   // returns the result of multiplying a chain of matrices fr
    if i < j
        // keep on splitting the chain and multiplying the matrices in left and right sides
        LeftSide = OptimalMatrixMultiplication(s, i, s[i, j])
        RightSide = OptimalMatrixMultiplication(s, s[i, j] + 1, j)
        return MatrixMultiply(LeftSide, RightSide)
    else if i = j
        return Ai   // matrix at position i
    else
        print "error, i <= j must hold"
  function MatrixMultiply(A, B)    // function that multiplies two matrices
      if columns(A) = rows(B)
          for i = 1, rows(A)
              for j = 1, columns(B)
                  C[i, j] = 0
                  for k = 1, columns(A)
                      C[i, j] = C[i, j] + A[i, k]*B[k, j]
                  return C
      else
          print "error, incompatible dimensions."
```

13.2.3 History of name

Term *dynamic programming* was originally used in 1940s by Richard Bellman to describe process of solving problems where one needs to find best decisions one after another. By 1953, he refined this to modern meaning, referring specifically to nesting smaller decision problems inside larger decisions, & field was therefore recognized by IEEE as a system analysis & engineering topic. Bellman's contribution is remembered in name of Bellman equation, a central result of DP which restates an optimization problem in recursive form.

Bellman explains reasoning behind term DP in his autobiography, *Eye of the Hurricane: An Autobiography*:

> "I spent Fall quarter (of 1950) at RAND. My 1st task was to find a name for multistage decision processes. An interesting question is, "Where did name, DP, come from?" 1950s were not good years for mathematical research. We had a very interesting gentleman in Washing named Wilson. He was Secretary of Defense, & actually had a pathological fear & hatred of word "research". I'm not using term lightly; I'm using it precisely. His face would suffuse, he would turn red, & he would get violent if people used term research in his presence. Can imagine how he felt, then, about term mathematical. RAND Corporation was employed by Air Force, & Air Force had Wilsn as its boss, essentially. Hence, I felt I had to do sth to shield Wilson & Air Force from fact I was really doing mathematics inside RAND Corporation. What title, what name, could I choose? In 1st place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use word "programming". I wanted to get across idea that this was dynamic, this was multistage, this was time-varying. I thought, let's kill 2 birds with 1 stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in classical physical sense. It also has a very interesting property as an adjective, & i.e., impossible to use word dynamic in a pejorative sense (ý nghĩa miệt thị). Try thinking of some combination that will possibly give it a pejorative meaning. Impossible. Thus, I thought dynamic programming was a good name. It was sth not even a Congressman could object to. So I used it as an umbrella for my activities." – Richard Bellman, *Eye of the Hurricane: An Autobiography* (1984, p. 159)

Word *dynamic* was chosen by Bellman to capture time-varying aspect of problems, & because it sounded impressive. Word *programming* referred to use of method to find an optimal *program*, in sense of a military schedule for training or logistics. This usage is same as that in phrases linear programming & mathematical programming, a synonym for mathematical optimization.

Above explanation of origin of term may be inaccurate: According to Russell & Norvig, above story "cannot be strictly true, because his 1st paper using the term (Bellman1952) appeared before Wilson became Secretary of Defense in 1953." Also, Harold J. Kushner stated in a speech: "On other hand, when I asked Bellman same question, he replied: he was trying to upstage Dantzig's linear programming by adding dynamic. Perhaps both motivations were true."" – Wikipedia/dynamic programming

13.3 Wikipedia/optimal substructure

"In CS, a problem is said to have *optimal substructure* if an optimal solution can be constructed from optimal solutions of its subproblems. This property is used to determine usefulness of greedy algorithms for a problem.

Typically, a greedy algorithm is used to solve a problem with optimal substructure if it can be proven by induction that this is optimal at each step. Otherwise, provided problem exhibits overlapping subproblems as well, divide-&-conquer methods or DP may be used. If there are no appropriate greedy algorithms & problem fails to exhibit overlapping subproblems, often a lengthy but straightforward search of solution space is best alternative.

In application of DP to mathematical optimization, Richard Bellman's Principle of Optimality is based on idea that in order to solve a dynamic optimization problem from some starting period t to some ending period T, one implicitly has to solve subproblems starting from later dates s, where $t < s < T$. This is an example of optimal substructure. Principle of Optimality is used to derive Bellman equation, which shows how value of problem starting from t is related to value of problem starting from s.

13.3.1 Example

Consider finding a shortest path for traveling between 2 cities by car, as illustrated in Fig. 1. Such an example is likely to exhibit optimal substructure. I.e., if shortest route from Seattle to Los Angeles passes through Portland & then Sacramento, then shortest route from Portland to Los Angeles must pass through Sacramento too. I.e., problem of how to get from Portland to Los Angeles is nested inside problem of how to get from Seattle to Los Angeles. (Wavy lines in graph represent solutions to subproblems.)

As an example of a problem that is unlikely to exhibit optimal substructure, consider problem of finding cheapest airline ticket from Buenos Aires to Moscow. Even if that ticket involves stops in Miami & then London, can't conclude: cheapest ticket from Miami to Moscow stops in London, because price at which an airline sells a multi-flight trip is usually not sum of prices at which it would sell individual flights in trip.

13.3.2 Def

A slightly more formal def of optimal substructure can be given. Let a "problem" be a collection of "alternatives", & let each alternative have an associated cost $c(a)$. Task: find a set of alternatives that minimizes $c(a)$. Suppose: alternatives can be partitioned into subsets, i.e., each alternative belongs to only 1 subset. Suppose each subset has its own cost function. Minima of each of these cost functions can be found, as can minima of global cost function, *restricted to same subsets*. If these minima match for each subsets, then it's almost obvious that a global minimum can be picked not out of full set of alternatives, but out of only set that consists of minima of smaller, local cost functions we have defined. If minimizing local functions is a problem of "lower order", & (specifically) if, after a finite number of these reductions, problem becomes trivial, then problem has an optimal substructure.

13.3.3 Problems with optimal substructure

- Longest common subsequence problem
- Longest increasing subsequence
- Longest palindromic substring
- All-Pairs Shortest Path
- Any problem that can be solved by DP.

13.3.4 Problems without optimal substructure

- Longest path problem
- Addition-chain exponentiation
- *Least-cost airline fare.* Using online flight search, will frequently find: cheapest flight from airport A to airport B involves a single connection through airport C, but cheapest flight from airport A to airport C involves a connection through some other airport D. However, if problem takes maximum number of layovers as a parameter, then problem has optimal substructure. Cheapest flight from A to B that involves at most k layovers is either direct flight; or cheapest flight from A to some airport C that involves at most t layovers for some integer t with $0 \leq t < k$, plus cheapest flight from C to B that involves at most $k - 1 - t$ layovers." – Wikipedia/optimal substructure

13.4 Wikipedia/overlapping subproblems

"In CS, a computational problem is said to have *overlapping subproblems* if problem can be broken down into subproblems which are reused several times or a recursive algorithm for problem solves same subproblem over & over rather than always generating new subproblems.

– Trong CS, 1 vấn đề tính toán được gọi là có *các vấn đề con chồng lấn* nếu vấn đề có thể được chia nhỏ thành các vấn đề con được sử dụng lại nhiều lần hoặc một thuật toán đệ quy để giải quyết cùng 1 vấn đề con nhiều lần thay vì luôn tạo ra các vấn đề con mới.

E.g., problem of computing Fibonacci sequence exhibits overlapping subproblems. Problem of computing nth Fibonacci number $F(n)$, can be broken down into subproblems of computing $F(n-1), F(n-2)$, & then adding the 2. Subproblem of computing $F(n-1)$ can itself be broken down into a subproblem that involves computing $F(n-2)$. Therefore, computation of $F(n-2)$ is reused, & Fibonacci sequence thus exhibits overlapping subproblems.

A naive recursive approach to such a problem generally fails due to an exponential complexity. If problem also shares an optimal substructure property, DP is a good way to work it out.

13.4.1 Fibonacci sequence example

In following 2 implementations for calculating Fibonacci sequence `fibonacci` uses regular recursion & `fibonacci_mem` uses memoization. `fibonacci_mem` is much more efficient as value for any particular n is computed only once.

```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)
def fibonacci_mem(n, cache):
    if n <= 1:
        return n
    if n in cache:
        return cache[n]
    cache[n] = fibonacci_mem(n - 1, cache) + fibonacci_mem(n - 2, cache)
    return cache[n]
print(fibonacci_mem(5, {}))  # 5
print(fibonacci(5))  # 5
```

When executed, `fibonacci` function computes value of some of numbers in sequence many times over, whereas `fibonacci_mem` reuses value of n which was computed previously [Figs.] Difference in performance may appear minimal with an n value of 5; however, as n increases, computational complexity of original `fibonacci` function grows exponentially. In contrast, `fibonacci_mem` version exhibits a more linear increase in complexity." – Wikipedia/overlapping subproblems

14 Miscellaneous

Tài liệu

[AH01] Grégoire Allaire and Antoine Henrot. "On some recent advances in shape optimization". In: *C. R. Acad. Sci. Paris* t. 329, Série II b (2001), pp. 383–396.

[AK16] Cung Thế Anh and Trần Đình Kế. *Nửa Nhóm Các Toán Tử Tuyến Tính & Ứng Dụng*. Nhà Xuất Bản Đại Học Sư Phạm, 2016, p. 222.

[Aze20] Hideyuki Azegami. *Shape optimization problems*. Vol. 164. Springer Optimization and Its Applications. Springer, Singapore, 2020, pp. xxiii+646. ISBN: 978-981-15-7618-8; 978-981-15-7617-1. DOI: 10.1007/978-981-15-7618-8. URL: https://doi.org/10.1007/978-981-15-7618-8.

[BW23] Catherine Bandle and Alfred Wagner. *Shape Optimization: Variations of Domains and Applications*. Vol. 42. De Gruyter Series in Nonlinear Analysis and Applications. De Gruyter, 2023, pp. xi+278. DOI: 10.1515/9783111025438-201. URL: https://doi.org/10.1515/9783111025438-201.

[DZ01] M. C. Delfour and J.-P. Zolésio. *Shapes and geometries*. Vol. 4. Advances in Design and Control. Analysis, differential calculus, and optimization. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001, pp. xviii+482. ISBN: 0-89871-489-3.

[DZ11] M. C. Delfour and Jean-Paul Zolésio. *Shapes and geometries*. Second. Vol. 22. Advances in Design and Control. Metrics, analysis, differential calculus, and optimization. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2011, pp. xxiv+622. ISBN: 978-0-898719-36-9. DOI: 10.1137/1.9780898719826. URL: https://doi.org/10.1137/1.9780898719826.

[HM03] J. Haslinger and R. A. E. Mäkinen. *Introduction to shape optimization*. Vol. 7. Advances in Design and Control. Theory, approximation, and computation. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2003, pp. xviii+273. ISBN: 0-89871-536-9. DOI: 10.1137/1.9780898718690. URL: https://doi.org/10.1137/1.9780898718690.

[KW19] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for optimization*. MIT Press, Cambridge, MA, 2019, pp. xx+500. ISBN: 978-0-262-03942-0.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[Lio69] J.-L. Lions. *Quelques méthodes de résolution des problèmes aux limites non linéaires*. Dunod, Paris; Gauthier-Villars, Paris, 1969, pp. xx+554.

[Lio71] J.-L. Lions. *Optimal control of systems governed by partial differential equations*. Die Grundlehren der mathematischen Wissenschaften, Band 170. Translated from the French by S. K. Mitter. Springer-Verlag, New York-Berlin, 1971, pp. xi+396.

[LT00] Irena Lasiecka and Roberto Triggiani. *Control theory for partial differential equations: continuous and approximation theories. II*. Vol. 75. Encyclopedia of Mathematics and its Applications. Abstract hyperbolic-like systems over a finite time horizon. Cambridge University Press, Cambridge, 2000, i–xxii and 645–1067 and I1–I4. ISBN: 0-521-58401-9. DOI: 10.1017/CBO9780511574801.002. URL: https://doi.org/10.1017/CBO9780511574801.002.

[MP10] Bijan Mohammadi and Olivier Pironneau. *Applied shape optimization for fluids*. Second. Numerical Mathematics and Scientific Computation. Oxford University Press, Oxford, 2010, pp. xiv+277. ISBN: 978-0-19-954690-9.

[MZ06] Marwan Moubachir and Jean-Paul Zolésio. *Moving shape analysis and control*. Vol. 277. Pure and Applied Mathematics (Boca Raton). Applications to fluid structure interactions. Chapman & Hall/CRC, Boca Raton, FL, 2006, pp. xx+291. ISBN: 978-1-58488-611-2; 1-58488-611-0. DOI: 10.1201/9781420003246. URL: https://doi.org/10.1201/9781420003246.

[Ray07] J.-P. Raymond. "Feedback boundary stabilization of the three-dimensional incompressible Navier-Stokes equations". In: *J. Math. Pures Appl. (9)* 87.6 (2007), pp. 627–669. ISSN: 0021-7824. DOI: 10.1016/j.matpur.2007.04.002. URL: https://doi.org/10.1016/j.matpur.2007.04.002.

[SZ92] Jan Sokołowski and Jean-Paul Zolésio. *Introduction to shape optimization*. Vol. 16. Springer Series in Computational Mathematics. Shape sensitivity analysis. Springer-Verlag, Berlin, 1992, pp. ii+250. ISBN: 3-540-54177-2. DOI: 10.1007/978-3-642-58106-9. URL: https://doi.org/10.1007/978-3-642-58106-9.

[Trö10] Fredi Tröltzsch. *Optimal control of partial differential equations*. Vol. 112. Graduate Studies in Mathematics. Theory, methods and applications, Translated from the 2005 German original by Jürgen Sprekels. American Mathematical Society, Providence, RI, 2010, pp. xvi+399. ISBN: 978-0-8218-4904-0. DOI: 10.1090/gsm/112. URL: https://doi.org/10.1090/gsm/112.

[Vil03] Cédric Villani. *Topics in optimal transportation*. Vol. 58. Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, 2003, pp. xvi+370. ISBN: 0-8218-3312-X. DOI: 10.1090/gsm/058. URL: https://doi.org/10.1090/gsm/058.

[Wal15] Shawn W. Walker. *The shapes of things*. Vol. 28. Advances in Design and Control. A practical guide to differential geometry and the shape derivative. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2015, pp. ix+154. ISBN: 978-1-611973-95-2. DOI: 10.1137/1.9781611973969.ch1. URL: https://doi.org/10.1137/1.9781611973969.ch1.