Notes for ECNU 2007 Short Course

October 2007

0. Introduction

• Purpose of the course: to offer the students a concise introduction to modern unconstrained and constrained optimization methods, algorithms, and computer software; functional optimization and applications will also be briefly addressed.

• Background mathematics includes first-year calculus and linear algebra, some knowledge of calculus of variations and PDE are helpful but not critical.

• Familiarity with MATLAB is important.

• Most of the distributed course material (Chapters 1, 2, 4, 5, 7, 9, 10, 12, 13, 14, 16, and Appendix A) are from the book

Andreas Antoniou and Wu-Sheng Lu, P*ractical Optimization: Algorithms and Engineering Applications*, Springer, New York, March 2007.

The book has a web site: http://www.ece.uvic.ca/~optimization/

I. Numerical Optimization and Applications

1.1 Unconstrained Optimization (Chaps. 1, 2, 4, 5, 7, 9)

• Basic optimization problem:

$$\text{mimimize} \quad f(\boldsymbol{x}) \qquad \text{for} \quad \boldsymbol{x} \in R^n$$

• Assumption: $f(\boldsymbol{x})$ is twice continuously differentiable.

• An example: solve a system of algebraic equations $p_i(\boldsymbol{x}) = 0$ for $i = 1, 2, \ldots, m$.

• Basic structure of a numerical algorithm for minimizing $f(\boldsymbol{x})$

Step 1: choose an initial point x0, set a convergence tolerance ε, and set a counter $k = 0$.

Step 2: determine a search direction $\boldsymbol{d}_k$ for reducing $f(\boldsymbol{x})$ from point $\boldsymbol{x}_k$.

Step 3: determine a step size α_k such that $f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k)$ is minimized for $\alpha \geq 0$, and

construct $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$

Step 4: if $\|\alpha_k \boldsymbol{d}_k\| < \varepsilon$, stop and output a solution $\boldsymbol{x}_{k+1}$, else set $k := k + 1$ and repeat from Step 2.

• Comments:

(a) Steps 2 and 3 are key steps of an optimization algorithm.

(b) Different ways to accomplish Step 2 leads to different algorithms

(c) Step 3 is a one-dimensional optimization problem and it is often called a line search step.

• Notation and basic quantities: $\boldsymbol{x}$: vector of n variables in column

$f(\boldsymbol{x})$: objective function

$\boldsymbol{g}(\boldsymbol{x})$: gradient (vector) of $f(\boldsymbol{x})$. A popular notation for $\boldsymbol{g}(\boldsymbol{x})$: $\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$

$\boldsymbol{H}(\boldsymbol{x})$: Hessian (matrix) of $f(\boldsymbol{x})$. A popular notation for $\boldsymbol{H}(\boldsymbol{x})$: $\boldsymbol{H}(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$

Note: $\boldsymbol{H}(\boldsymbol{x})$ is a square, symmetric matrix and can be obtained as $\boldsymbol{H}(\boldsymbol{x}) = \nabla(\nabla^T f(\boldsymbol{x}))$

A point $\boldsymbol{x}$ is called a stationary point if $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$.

• First-order necessary condition: If $\boldsymbol{x}^*$ is a minimum point (minimizer), then $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$. In other words, if $\boldsymbol{x}^*$ is a minimum point, then it must be a stationary point.

• Second-order necessary condition: If $\boldsymbol{x}^*$ is a minimum point (minimizer), then $\boldsymbol{H}(\boldsymbol{x}^*)$ is a positive semidefinite matrix, i.e., $\boldsymbol{H}(\boldsymbol{x}^*) \succeq \boldsymbol{0}$.

• Second-order sufficient condition: If $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$ and $\boldsymbol{H}(\boldsymbol{x}^*)$ is a positive definite matrix, i.e., $\boldsymbol{H}(\boldsymbol{x}^*) \succ \boldsymbol{0}$, then $\boldsymbol{x}^*$ is a minimum point (minimizer).

• One-dimensional (1-D) optimization (line search)

♦ Popular 1-D methods for unimodal functions include Dichotomous search, Fibonacci search, Golden-section search, quadratic interpolation method, and cubic interpolation method.

(we illustrate the basic ideas behind the Dichotomous search, Golden-section search, and quadratic interpolation methods.)

♦ 1-D optimization for general smooth functions: the inexact line search method by Roger Fletcher. Matlab codes implementing these search methods are available for download at http://www.ece.uvic.ca/~optimization/

- Determination of a search direction $\boldsymbol{d}_k$:

♦ Basis of the methods: Taylor series of $f(\boldsymbol{x})$ in a small vicinity surrounding point $\boldsymbol{x}_k$:

$$f(\boldsymbol{x}_k+\boldsymbol{\delta}) = f(\boldsymbol{x}_k)+\nabla^T f(\boldsymbol{x}_k)\cdot\boldsymbol{\delta}+\frac{1}{2}\boldsymbol{\delta}^T\cdot\nabla^2 f(\boldsymbol{x}_k)\cdot\boldsymbol{\delta}+O\left(\|\boldsymbol{\delta}\|^3\right)$$

or

$$f(\boldsymbol{x}_k+\boldsymbol{\delta}) = f(\boldsymbol{x}_k)+\boldsymbol{g}^T(\boldsymbol{x}_k)\cdot\boldsymbol{\delta}+\frac{1}{2}\boldsymbol{\delta}^T\cdot\boldsymbol{H}(\boldsymbol{x}_k)\cdot\boldsymbol{\delta}+O\left(\|\boldsymbol{\delta}\|^3\right)$$

♦ Steepest descent method: $\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$, i.e., $\boldsymbol{d}_k = -\boldsymbol{g}(\boldsymbol{x}_k)$

□ Remarks:

(i) The concept of a descent algorithm $f(\boldsymbol{x}_{k+1}) \le f(\boldsymbol{x}_k)$ for $k = 0,1,.....$

(ii) The steepest descent algorithm is a descent algorithm.

(iii) The steepest descent algorithm is usually quite slow.

♦ Newton method: $\boldsymbol{d}_k = -\boldsymbol{H}^{-1}(\boldsymbol{x}_k)\boldsymbol{g}(\boldsymbol{x}_k)$

□ Remarks:

(i) if $\boldsymbol{H}(\boldsymbol{x}_k) = \boldsymbol{H}_k$ is positive definite for all $\boldsymbol{x}_k$, then the Newton algorithm is a descent algorithm.

(ii) if $\boldsymbol{H}_k$ is not positive definite, then the search direction in Newton method is modified to

$$\boldsymbol{d}_k = -\hat{\boldsymbol{H}}_k^{-1}\boldsymbol{g}_k \text{ where } \hat{\boldsymbol{H}}_k = \frac{\boldsymbol{H}_k+\beta\boldsymbol{I}}{1+\beta} \text{ with a } \beta \text{ such that } \boldsymbol{H}_k+\beta\boldsymbol{I} \succ \boldsymbol{0}$$

The Newton method with such modifications is a descent algorithm.

(iii) Typically Newton algorithm converges considerably faster than the steepest descent algorithm at a cost of increased computational intensity.

♦ Quasi-Newton methods: one can take a unifying look at the above two methods as

$$\boldsymbol{d}_k = -\boldsymbol{S}_k\boldsymbol{g}_k$$

where

$$\boldsymbol{S}_k = \begin{cases} \boldsymbol{I} & \text{for steepest descent} \\ \boldsymbol{H}_k^{-1} & \text{for Newton} \end{cases}$$

□ Quasi-Newton methods use gradient information to generate successive low-rank estimation of the inverse Hessian matrix. In this way, quasi-Newton algorithms provide convergent rates comparable with that of Newton algorithm with reduced complexity.

□ Notation: $\boldsymbol{\delta}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$, $\boldsymbol{\gamma}_k = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$

□ Requirements for $\boldsymbol{S}_k$: (i) Symmetric and positive definite, (ii) $\boldsymbol{S}_{k+1}\boldsymbol{\gamma}_k = \boldsymbol{\delta}_k$

□ Davidon-Fletcher-Powell (DFP) updating formula: $\boldsymbol{S}_0 = \boldsymbol{I}$, and

$$\boldsymbol{S}_{k+1} = \boldsymbol{S}_k + \frac{\boldsymbol{\delta}_k\boldsymbol{\delta}_k^T}{\boldsymbol{\delta}_k^T\boldsymbol{\gamma}_k} - \frac{\boldsymbol{S}_k\boldsymbol{\gamma}_k\boldsymbol{\gamma}_k^T\boldsymbol{S}_k}{\boldsymbol{\gamma}_k^T\boldsymbol{S}_k\boldsymbol{\gamma}_k}$$

□ Broyden-Fletcher-Goldfarb-Shanno (BFGS) updating formula: $\boldsymbol{S}_0 = \boldsymbol{I}$, and

$$\boldsymbol{S}_{k+1} = \boldsymbol{S}_k + \left(1+\frac{\boldsymbol{\gamma}_k^T\boldsymbol{S}_k\boldsymbol{\gamma}_k}{\boldsymbol{\gamma}_k^T\boldsymbol{\delta}_k}\right)\frac{\boldsymbol{\delta}_k\boldsymbol{\delta}_k^T}{\boldsymbol{\gamma}_k^T\boldsymbol{\delta}_k} - \frac{\boldsymbol{\delta}_k\boldsymbol{\gamma}_k^T\boldsymbol{S}_k + \boldsymbol{S}_k\boldsymbol{\gamma}_k\boldsymbol{\delta}_k^T}{\boldsymbol{\gamma}_k^T\boldsymbol{\delta}_k}$$

1.2 Constrained Optimization: The Karush-Kuhn-Tucker (KKT) Conditions (Chap. 10)

• The constrained optimization problem

$$\begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}) \\ &\text{subject to}: \; a_i(\boldsymbol{x}) = 0 \quad \text{for } i = 1,2,...,p \\ &\qquad\qquad c_j(\boldsymbol{x}) \geq 0 \quad \text{for } j = 1,2,...,q \end{aligned}$$

• Feasible region: $\{\boldsymbol{x} : a_i(\boldsymbol{x}) = 0 \text{ for } i = 1,2,...,p, \text{ and } c_j(\boldsymbol{x}) \geq 0 \text{ for } j = 1,2,...,q\}$

• A point x is said to be feasible if it is in the feasible region (inside or on the boundary). We say an inequality constraint $c_j(\boldsymbol{x}) \geq 0$ is active at a feasible point $\boldsymbol{x}$, if $c_j(\boldsymbol{x}) = 0$. And inequality constraint $c_j(\boldsymbol{x}) \geq 0$ is said to be inactive at a feasible point $\boldsymbol{x}$ if $c_j(\boldsymbol{x}) > 0$.

• KKT conditions (first-order necessary conditions for point $\boldsymbol{x}^*$ to be a minimum point):
If $\boldsymbol{x}^*$ is a local minimizer of the constrained problem, then

(a) $a_i(\boldsymbol{x}^*) = 0 \quad \text{for } i = 1,2,...,p$

(b) $c_j(\boldsymbol{x}^*) \geq 0 \text{ for } j = 1,2,...,q$

(c) there exist Lagrange multipliers λ_i^* for $1 \leq i \leq p$ and μ_j^* for $1 \leq j \leq q$ such that

$$\nabla f(\boldsymbol{x}^*) = \sum_{i=1}^{p} \lambda_i^* \nabla a_i(\boldsymbol{x}^*) + \sum_{j=1}^{q} \mu_j^* \nabla c_i(\boldsymbol{x}^*)$$

(d) $\mu_j^* c_j(\boldsymbol{x}^*) = 0 \text{ for } j = 1,2,...,q$

(e) $\mu_j^* \geq 0 \text{ for } j = 1,2,...,q$

Remarks:

♦ Conditions (a) and (b) are merely the constraints imposed in the problem.

♦ Condition (c) says at a local minimizer, the gradient of the objective function is a linear combination of the gradient of the constraint functions, and the coefficients of the combination are Lagrange multipliers. See Example 10.10 and Fig. 10.8 for illustration.

♦ Define Lagrangian $L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) - \sum_{i=1}^{p} \lambda_i a_i(\boldsymbol{x}) - \sum_{j=1}^{q} \mu_j c_j(\boldsymbol{x})$, the equation in (c) can be expressed as

$$\nabla_x L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0}$$

Also note that the equality and inequality constraints can be expressed in terms of Lagrangian as

$$\nabla_{\boldsymbol{\lambda}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{0}$$

and

$$\nabla_{\boldsymbol{\mu}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \mathbf{0}$$

♦ Conditions in (d) are called *complementarity conditions*. It follows that if the *k*th inequality constraint is in active at $\boldsymbol{x}^*$, then $\mu_k = 0$. In other words, the equation in (c) only contains those terms $\nabla c_j(\boldsymbol{x}^*)$ that are active at $\boldsymbol{x}^*$.

♦ The total number of equations *p* (from (a)) + *n* (from (c)) + *q* (from (d)) are equal to the total number of unknowns $\{\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$.

1.3 Convex Sets, Convex Functions, and Convex Programming (CP) (Chaps. 2, 10, 12, 13, 14)

♦ A set $\mathcal{R}$ is said to be convex if for every pair of points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{R}$ and every real α with $0 \leq \alpha \leq 1$, point $\boldsymbol{x} = \alpha \boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2$ belongs to $\mathcal{R}$ for all $0 \leq \alpha \leq 1$.

♦ A function $f(\boldsymbol{x})$ defined over a convex set $\mathcal{R}$ is said to be convex if for every pair of points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{R}$ and every real α with $0 \le \alpha \le 1$,

$$f(\alpha \boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2) \le \alpha f(\boldsymbol{x}_1) + (1-\alpha) f(\boldsymbol{x}_2)$$

♦ Suppose $f(\boldsymbol{x}) \in C^1$. Function $f(\boldsymbol{x})$ is convex over a convex set $\mathcal{R}$ if and only if

$$f(\boldsymbol{x}_1) \ge f(\boldsymbol{x}) + g(\boldsymbol{x})^T (\boldsymbol{x}_1 - \boldsymbol{x})$$

for all $\boldsymbol{x}$ and $\boldsymbol{x}_1 \in \mathcal{R}$.

♦ Using the above property, it can be shown that a function $f(\boldsymbol{x}) \in C^2$ is convex over a convex set $\mathcal{R}$ if anf only if the Hessian $\boldsymbol{H}(\boldsymbol{x})$ of $f(\boldsymbol{x})$ is positive semidefinite for $\boldsymbol{x} \in \mathcal{R}$.

♦ The problem of minimizing a convex function $f(\boldsymbol{x})$ (with no constraints) is not very complicated, because the first-order necessary condition becomes a sufficient condition, and any local minimizer of a convex function is a global minimizer.

♦ A constrained minimization problem

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) \\ &\text{subject to:} && a_i(\boldsymbol{x}) = 0 \quad \text{for } i = 1, 2, ..., p \\ &&& c_j(\boldsymbol{x}) \ge 0 \quad \text{for } j = 1, 2, ..., q \end{aligned}$$

is said to be a convex programming (CP) problem if the objective function is convex and the feasible region defined by the constraints, i.e., $\{\boldsymbol{x} : a_i(\boldsymbol{x}) = 0 \text{ for } i = 1, 2, ..., p, \text{ and } c_j(\boldsymbol{x}) \ge 0 \text{ for } j = 1, 2, ..., q\}$ is a convex set. In words, a CP problem is a problem of minimizing a convex function over a convex region. It can be shown that the feasible region $\{\boldsymbol{x} : a_i(\boldsymbol{x}) = 0 \text{ for } i = 1, 2, ..., p, \text{ and } c_j(\boldsymbol{x}) \ge 0 \text{ for } j = 1, 2, ..., q\}$ is convex if (i) all equality constraint functions $a_i(\boldsymbol{x})$ are linear, and (ii) all inequality functions $-c_j(\boldsymbol{x})$ are convex.

♦ An important property of the class of CP problems is that the KKT conditions become both necessary and sufficient. This implies that any $\{\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$ satisfying the KKT conditions gives a local solution (minimizer) $\boldsymbol{x}^*$ for the CP problem and any local minimizer of a CP problem is a global minimizer.

1.4 Wolfe Duality (Chap. 10)

Wolfe's theorem (1991) on duality is concerned with the general CP problem

$$(\mathcal{P}) \qquad \begin{aligned} &\text{minimize} && f(\boldsymbol{x}) \\ &\text{subject to:} && a_i(\boldsymbol{x}) = \boldsymbol{a}_i^T \boldsymbol{x} - b_i = 0 \quad \text{for } i = 1, 2, ..., p \\ &&& c_j(\boldsymbol{x}) \ge 0 \quad \text{for } j = 1, 2, ..., q \end{aligned}$$

which is referred as the *primal problem*. Wolfe's theorem says that if $\{\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$ solve the KKT system for the primal problem, then $\{\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$ also solves the dual problem which is defined by

$$(\mathcal{D}) \qquad \begin{aligned} &\underset{\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}}{\text{maximize}} && L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \\ &\text{subject to:} && \nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \boldsymbol{0} \\ &&& \boldsymbol{\mu} \ge \boldsymbol{0} \end{aligned}$$

In addition, $f(\boldsymbol{x}^*) = L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$.

♦ Proof: From the KKT conditions of the primal problem, we obtain $f(\boldsymbol{x}^*) = L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ and $\boldsymbol{\mu}^* \ge \boldsymbol{0}$. For a feasible set $\{\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}\}$, we have $\boldsymbol{\mu} \ge \boldsymbol{0}$ and $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \boldsymbol{0}$, hence

$$L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = f(\boldsymbol{x}^*) \ge f(\boldsymbol{x}^*) - \sum_{i=1}^{p} \lambda_i a_i(\boldsymbol{x}^*) - \sum_{j=1}^{q} \mu_j c_j(\boldsymbol{x}^*) = L(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu})$$

With $\boldsymbol{\mu} \geq \mathbf{0}$ the Lagrangian $L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ is convex, hence

$$L(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) \geq L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) + (\boldsymbol{x}^* - \boldsymbol{x})^T \nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$$

Therefore $L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \geq L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$, i.e., set $\{\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\}$ solves the dual problem. ■

♦ If we define the duality gap $\delta(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) - L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$, then the KKT conditions imply that $\delta(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \geq 0$ for feasible $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ and $\delta(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = 0$. This is because for a feasible set $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$, we have $\delta(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) - L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \sum_{i=1}^{p} \lambda_i a_i(\boldsymbol{x}) + \sum_{j=1}^{q} \mu_j c_j(\boldsymbol{x}) = \sum_{j=1}^{q} \mu_j c_j(\boldsymbol{x}) \geq 0$, and

$$\delta(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \sum_{j=1}^{q} \mu_j^* c_j(\boldsymbol{x}^*) = 0$$

1.5 Linear Programming and Quadratic Programming (Chap. 12)

♦ Linear programming problems are of great importance in both theoretical and practical perspectives. The modern theory and algorithms of interior-point methods were initially developed for the LP problems in 1980's and the basic concepts and solution techniques for LP have since been extended to larger classes of CP and non-convex problems. On the other hand, there are a great many of practical problems that can be accurately or approximately modeled as LP problems.

♦ The standard-form linear programming (LP) problem is given by

$$(\mathcal{P}) \qquad \begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{x} \\ &\text{subject to:} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ &\qquad\qquad\quad \boldsymbol{x} \geq \mathbf{0} \end{aligned}$$

where matrix $\boldsymbol{A}$ is of size $p \times n$ with $p < n$ and $\boldsymbol{A}$ is assumed to have full row rank. It is important to realize that LP problems are convex programming problems. Hence the KKT conditions are necessary and sufficient conditions, and any local solution is a global solution.

♦ Moreover, the Wolfe theorem applies: the dual of the above problem is given by

$$(\mathcal{D}) \qquad \begin{aligned} &\text{maximize} \quad h(\boldsymbol{\lambda}) = \boldsymbol{b}^T \boldsymbol{\lambda} \\ &\text{subject to:} \quad \boldsymbol{A}^T \boldsymbol{\lambda} + \boldsymbol{\mu} = \boldsymbol{c} \\ &\qquad\qquad\quad \boldsymbol{\mu} \geq \mathbf{0} \end{aligned}$$

♦ The KKT conditions in this case are given by

$$\begin{aligned} &\boldsymbol{A}^T \boldsymbol{\lambda} + \boldsymbol{\mu} = \boldsymbol{c} \\ &\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ &x_i \mu_i = 0 \qquad \text{for} \quad 1 \leq i \leq n \\ &\boldsymbol{x} \geq \mathbf{0}, \quad \boldsymbol{\mu} \geq \mathbf{0} \end{aligned}$$

♦ A useful concept is the development of interior-point algorithms for LP is *central path*. To introduce it we first write the KKT conditions as

$$\begin{aligned} &\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \qquad \text{with} \quad \boldsymbol{x} \geq \mathbf{0} \\ &\boldsymbol{A}^T \boldsymbol{\lambda} + \boldsymbol{\mu} = \boldsymbol{c} \quad \text{with} \quad \boldsymbol{\mu} \geq \mathbf{0} \\ &\boldsymbol{X}\boldsymbol{\mu} = \mathbf{0} \end{aligned}$$

where $\boldsymbol{X} = \text{diag}\{x_1, x_2, \ldots, x_n\}$. The central path for a standard-form LP is defined by a set of vectors $\{\boldsymbol{x}(\tau), \boldsymbol{\lambda}(\tau), \boldsymbol{\mu}(\tau)\}$ that satisfy

$$(C) \qquad \begin{aligned} &\boldsymbol{Ax} = \boldsymbol{b} && \text{with } \boldsymbol{x} > \boldsymbol{0} \\ &\boldsymbol{A}^T\boldsymbol{\lambda} + \boldsymbol{\mu} = \boldsymbol{c} && \text{with } \boldsymbol{\mu} > \boldsymbol{0} \\ &\boldsymbol{X\mu} = \tau\boldsymbol{e} && \text{with } \tau > 0,\ \boldsymbol{e} = [1\ 1 \cdots 1]^T \end{aligned}$$

The duality gap on the central path is found to be

$$\begin{aligned} \delta\left[\boldsymbol{x}(\tau), \boldsymbol{\lambda}(\tau)\right] &= \boldsymbol{c}^T\boldsymbol{x}(\tau) - \boldsymbol{b}^T\boldsymbol{\lambda}(\tau) \\ &= \left[\boldsymbol{\lambda}^T(\tau)\boldsymbol{A} + \boldsymbol{\mu}^T(\tau)\right]\boldsymbol{x}(\tau) - \boldsymbol{b}^T\boldsymbol{\lambda}(\tau) \\ &= \boldsymbol{\mu}^T(\tau)\boldsymbol{x}(\tau) = n\tau \end{aligned}$$

We see that duality gap approaches zero as parameter τ approaches zero. In other words, the central path defines a parameterized trajectory approaching to the solution of both primal and dual problems as τ approaches zero.

♦ Below we describe a primal-dual path-following algorithm for LP problems. The idea behind the algorithm is to start with a strictly feasible initial set of vectors $\boldsymbol{w}_0 = \{\boldsymbol{x}_0, \boldsymbol{\lambda}_0, \boldsymbol{\mu}_0\}$ and use first-order perturbations of the equations (C) to obtain a set of increment vectors to update point $\boldsymbol{w}_k$. So in the kth iteration we seek $\boldsymbol{\delta}_w = \{\boldsymbol{\delta}_x, \boldsymbol{\delta}_\lambda, \boldsymbol{\delta}_\mu\}$ such that the next iterate

$$\boldsymbol{w}_{k+1} = \{\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1}\} = \{\boldsymbol{x}_k + \boldsymbol{\delta}_x, \boldsymbol{\lambda}_k + \boldsymbol{\delta}_\lambda, \boldsymbol{\mu}_k + \boldsymbol{\delta}_\mu\}$$

remains strictly feasible and approaches the central path defined by (C) with $\tau = \tau_{k+1}$. This leads (C) to a system of linear equations as follows

$$\begin{aligned} &\boldsymbol{A\delta}_x = \boldsymbol{0} \\ &\boldsymbol{A}^T\boldsymbol{\delta}_\lambda + \boldsymbol{\delta}_\mu = \boldsymbol{0} \\ &\boldsymbol{M\delta}_x + \boldsymbol{X\delta}_\mu = \tau_{k+1}\boldsymbol{e} - \boldsymbol{X\mu}_k \end{aligned}$$

where

$$\boldsymbol{M} = \text{diag}\{(\boldsymbol{\mu}_k)_1, (\boldsymbol{\mu}_k)_2, \cdots, (\boldsymbol{\mu}_k)_n\}$$

♦ Additional details:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k\boldsymbol{\delta}_w$$

where α_k is obtained by a line-search step to ensure the strict feasibility of point $\boldsymbol{w}_{k+1}$.

□ The value of parameter τ_{k+1} is determine by

$$\tau_{k+1} = \frac{\boldsymbol{\mu}_k^T\boldsymbol{x}_k}{n + \rho} \quad \textit{with } \rho > \sqrt{n}$$

□ An interior-point algorithm with nonfeasible initialization cab be developed in a similar way, see Sec. 12.5.2.

♦ LP problems may be formulated and solved in an alternative-form which are equivalent to LP problems in the standard form:

$$\begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} \\ &\text{subject to}: \quad \boldsymbol{Ax} \geq \boldsymbol{b} \end{aligned}$$

Counterpart algorithms for LP problems in alternative-form can also be developed.

1.6 Semidefinite Programming (SDP) (Chap. 14)

♦ SDP is a class of CP problems that includes LP and convex quadratic programming problems as its subclasses and a lot more other CP problems of practical usefulness, 1990's have witnessed intensive research interest in SDP that has resulted in many efficient solution methods and software tools for SDP problems.

♦ Notation and definitions

We denote the set of $n \times n$ symmetric matrices by $\mathcal{S}^n$. For $\boldsymbol{A}, \boldsymbol{B} \in \mathcal{S}^n$, the inner product of $\boldsymbol{A}$ with $\boldsymbol{B}$ is defined by

$$\boldsymbol{A} \cdot \boldsymbol{B} = \text{trace}(\boldsymbol{AB}) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{ij}$$

Matrix $\boldsymbol{A}$ being positive semidefinite is signified as $\boldsymbol{A} \succeq \boldsymbol{0}$, and $\boldsymbol{A}$ being positive definite is denoted by $\boldsymbol{A} \succ \boldsymbol{0}$.

♦ The primal SDP problem is defined as

$$(\mathcal{P}) \qquad \begin{aligned} &\text{minimize} && \boldsymbol{C} \cdot \boldsymbol{X} \\ &\text{subject to:} && \boldsymbol{A}_i \cdot \boldsymbol{X} = b_i \quad \text{for } i = 1, 2, ..., p \\ & && \boldsymbol{X} \succeq \boldsymbol{0} \end{aligned}$$

where $\boldsymbol{C}$, $\boldsymbol{X}$, and $\boldsymbol{A}_i$ are members of $\mathcal{S}^n$. Note that if $\boldsymbol{C} = \text{diag}\{\text{c}\}$, $\boldsymbol{A}_i = \text{diag}\{\boldsymbol{a}_i\}$, and $\boldsymbol{x} = \text{diag}\{\boldsymbol{X}\}$ for some vectors $\boldsymbol{c}$ and $\boldsymbol{a}_i$, then the SDP problem becomes a standard LP problem:

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{x} \\ &\text{subject to:} && \boldsymbol{Ax} = \boldsymbol{b} \\ & && \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

where $\boldsymbol{A}$ is a matrix with $\boldsymbol{a}_i^T$ as its i-th row and $\boldsymbol{b} = [b_1\ b_2\ \ldots\ b_p]^T$.

♦ The dual SDP problem assumes the form

$$(\mathcal{D}) \qquad \begin{aligned} &\text{maximize} && \boldsymbol{b}^T \boldsymbol{y} \\ &\text{subject to:} && \sum_{i=1}^{p} y_i \boldsymbol{A}_i + \boldsymbol{S} = \boldsymbol{C} \\ & && \boldsymbol{S} \succeq \boldsymbol{0} \end{aligned}$$

By eliminating $\boldsymbol{S}$ and making simple variable changes, we can write the dual problem as

$$(\mathcal{D}') \qquad \begin{aligned} &\text{minimize} && \boldsymbol{c}^T \boldsymbol{x} \\ &\text{subject to:} && \boldsymbol{F}(\boldsymbol{x}) \succeq \boldsymbol{0} \end{aligned}$$

where

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{F}_0 + \sum_{i=1}^{p} x_i \boldsymbol{F}_i$$

♦ The KKT conditions for SDP

$$\begin{aligned} &\sum_{i=1}^{p} y_i \boldsymbol{A}_i + \boldsymbol{S} = \boldsymbol{C} \\ &\boldsymbol{A}_i \cdot \boldsymbol{X} = b_i \qquad \text{for } i = 1, 2., ..., p \\ &\boldsymbol{SX} = \boldsymbol{0} \\ &\boldsymbol{X} \succeq \boldsymbol{0},\ \boldsymbol{S} \succeq \boldsymbol{0} \end{aligned}$$

♦ An example: convex quadratic programming (QP) problems
The general convex QP problems can be expressed as

$$\begin{aligned} &\text{minimize} \quad \boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} + \boldsymbol{p}^T\boldsymbol{x} \qquad \text{with} \;\; \boldsymbol{H} \succeq \boldsymbol{0} \\ &\text{subject to}: \;\; \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \end{aligned}$$

which is equivalent to

$$\begin{aligned} &\text{minimize} \quad \delta \\ &\text{subject to}: \;\; \boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} + \boldsymbol{p}^T\boldsymbol{x} \leq \delta \\ &\qquad\qquad \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \end{aligned}$$

where, by using a decomposition of $\boldsymbol{H} = \hat{\boldsymbol{H}}^T\hat{\boldsymbol{H}}$, the first constraint can be expressed as

$$\delta - \boldsymbol{p}^T\boldsymbol{x} - (\hat{\boldsymbol{H}}\boldsymbol{x})^T(\hat{\boldsymbol{H}}\boldsymbol{x}) \geq 0$$

which is equivalent to

$$\boldsymbol{G}(\delta, \boldsymbol{x}) = \begin{bmatrix} \boldsymbol{I} & \hat{\boldsymbol{H}}\boldsymbol{x} \\ (\hat{\boldsymbol{H}}\boldsymbol{x})^T & \delta - \boldsymbol{p}^T\boldsymbol{x} \end{bmatrix} \succeq \boldsymbol{0}$$

In addition, the second constraint can be written as

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{F}_0 + \sum_{i=1}^{n} x_i\boldsymbol{F}_i \succeq \boldsymbol{0}$$

where $\boldsymbol{F}_0 = -\text{diag}\{\boldsymbol{b}\}$, and $\boldsymbol{F}_i = \text{diag}\{\boldsymbol{a}_i\}$ with $\boldsymbol{a}_i$ being the i-th column of $\boldsymbol{A}$.

By defining an augmented vector $\hat{\boldsymbol{x}} = \begin{bmatrix} \delta \\ \boldsymbol{x} \end{bmatrix}$, the convex QP problem is now formulated as the SDP problem

$$\begin{aligned} &\text{minimize} \quad \hat{\boldsymbol{c}}^T\hat{\boldsymbol{x}} \\ &\text{subject to}: \;\; \boldsymbol{E}(\hat{\boldsymbol{x}}) \succeq \boldsymbol{0} \end{aligned}$$

where $\hat{\boldsymbol{c}} = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T$, and $\boldsymbol{E}(\hat{\boldsymbol{x}}) = \text{diag}\{\boldsymbol{G}(\delta, \boldsymbol{x}), \boldsymbol{F}(\boldsymbol{x})\}$.

1.7 Second-Order Cone Programming (SOCP) (Chap. 14)

♦ The class of SOCP problem is a subset of the class of SDP problems but it is still large enough to include LP, QP, and many other CP problems of practical importance. Moreover, interior-point algorithms that are specifically designed for SOCP is considerably more efficient than what the best an SDP algorithm can offer.

♦ Notation and definitions

□ A second-order cone (also known as quadratic cone or Lorentz cone) of dimension n is defined as

$$\mathcal{K} = \left\{ \begin{bmatrix} t \\ \boldsymbol{u} \end{bmatrix} : t \in R, \, \boldsymbol{u} \in R^{n-1} \text{ for } \|\boldsymbol{u}\| \leq t \right\}$$

□ Examples: $n = 1$: a ray on the t-axis starting at $t = 0$, for $n = 2$ and 3 see Fig. 14.3.

□ Note that the second-order cone is convex in R^n.

♦ The primal SOCP problem is given by

$$(\mathcal{P}) \qquad \begin{aligned} &\text{minimize} \quad \sum_{i=1}^{q} \hat{\boldsymbol{c}}_i^T\boldsymbol{x}_i \\ &\text{subject to:} \quad \sum_{i=1}^{q} \hat{\boldsymbol{A}}_i\boldsymbol{x}_i = \boldsymbol{b} \\ &\qquad\qquad \boldsymbol{x}_i \in K_i \quad \text{for} \quad i = 1, 2, ..., q \end{aligned}$$

where $\hat{\boldsymbol{c}}_i \in R^{ni\times 1}$, $\boldsymbol{x}_i \in R^{ni\times 1}$, $\hat{\boldsymbol{A}}_i \in R^{m\times ni}$, $\boldsymbol{b} \in R^{m\times 1}$, and K_i is a second-order cone of dimension n_i.

□ We note an analogy between SOCP and LP: both involve a linear objective function and a set of linear equality constraints. While the variable $\boldsymbol{x}$ in an LP problem is constrained to the region $\{\boldsymbol{x} : \boldsymbol{x} \geq \boldsymbol{0}\}$ which is a convex cone, each variable vector $\boldsymbol{x}_i$ in an SOCP problem is constrained to the second-order cone K_i.

♦ The dual SOCP problem assumes the form

$$
\begin{array}{lll}
 & \text{maximize} & \boldsymbol{b}^T \boldsymbol{y} \\
(\mathcal{D}) & \text{subject to:} & \hat{\boldsymbol{A}}_i^T \boldsymbol{y} + \boldsymbol{s}_i = \hat{\boldsymbol{c}}_i \quad \text{for} \quad i = 1, 2, ..., q \\
 & & \boldsymbol{s}_i \in K_i \quad \text{for} \quad i = 1, 2, ..., q
\end{array}
$$

□ We also notice a similar analogy between the dual SOCP and dual LP problems.

♦ Now if we let

$$
\boldsymbol{x} = -\boldsymbol{y},\ \hat{\boldsymbol{A}}_i^T = \begin{bmatrix} \boldsymbol{b}_i^T \\ \boldsymbol{A}_i^T \end{bmatrix} \ \text{and} \ \hat{\boldsymbol{c}}_i = \begin{bmatrix} d_i \\ \boldsymbol{c}_i \end{bmatrix}
$$

then the dual SOCP problem can be expressed as

$$
\begin{array}{lll}
 & \text{minimize} & \boldsymbol{b}^T \boldsymbol{x} \\
(\mathcal{D}') & \text{subject to:} & \left\| \boldsymbol{A}_i^T \boldsymbol{x} + \boldsymbol{c}_i \right\| \leq \boldsymbol{b}_i^T \boldsymbol{x} + d_i \quad \text{for} \quad i = 1, 2, ..., q
\end{array}
$$

□ An example: Linear fractional problem

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{i=1}^{p} \frac{1}{\boldsymbol{a}_i^T \boldsymbol{x} + c_i} & \\
\text{subject to:} & \boldsymbol{a}_i^T \boldsymbol{x} + c_i > 0 & \text{for } i = 1, 2, ..., p \\
 & \boldsymbol{b}_i^T \boldsymbol{x} + d_i \geq 0 & \text{for } i = 1, 2, ..., q
\end{array}
$$

By introducing auxiliary constraints

$$
\frac{1}{\boldsymbol{a}_i^T \boldsymbol{x} + c_i} \leq \delta_i
$$

i.e.,

$$
\delta_i (\boldsymbol{a}_i^T \boldsymbol{x} + c_i) \geq 1
$$

the linear fractional problem can be formulated as

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{i=1}^{p} \delta_i & \\
\text{subject to:} & \delta_i (\boldsymbol{a}_i^T \boldsymbol{x} + c_i) \geq 1 & \text{for } i = 1, 2, ..., p \\
 & \delta_i \geq 0 & \\
 & \boldsymbol{b}_i^T \boldsymbol{x} + d_i \geq 0 &
\end{array}
$$

We can show that $w^2 \leq uv, u \geq 0, v \geq 0$ if and only if

$$
\left\| \begin{bmatrix} 2w \\ u - v \end{bmatrix} \right\| \leq u + v
$$

Hence the first two constraints in the above problem are equivalent to

$$\left\| \begin{bmatrix} 2 \\ \boldsymbol{a}_i^T \boldsymbol{x} + c_i - \delta_i \end{bmatrix} \right\| \le \boldsymbol{a}_i^T \boldsymbol{x} + c_i + \delta_i \quad \text{for } i = 1, 2, ..., p$$

and the linear fractional problem can be converted into

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{p} \delta_i \\ \text{subject to:} \quad & \left\| \begin{bmatrix} 2 \\ \boldsymbol{a}_i^T \boldsymbol{x} + c_i - \delta_i \end{bmatrix} \right\| \le \boldsymbol{a}_i^T \boldsymbol{x} + c_i + \delta_i \quad \text{for } i = 1, 2, ..., p \\ & \boldsymbol{b}_i^T \boldsymbol{x} + d_i \ge 0 \end{aligned}$$

which is an SOCP problem.

Use SeDuMi to Solve LP, SDP and SCOP Problems: Remarks and Examples*

* This file was prepared by Wu-Sheng Lu, Dept. of Electrical and Computer Engineering, University of Victoria, revised on January 29, 2007.

The name of the toolbox, *SeDuMi*, stands for *self-dual minimization* as it implements a self-dual embedding technique for optimization over self-dual homogeneous cones. Below we give comments on the usage of this software for the solution of LP, SDP, and SOCP problems. Numerical examples are included for illustration purposes. Some of the examples are from the book:
A. Antoniou and W.-S. Lu, *Optimization: Algorithms and Applications*, Springer, 2007.

0. Presently the official web site for public-domain software SeDuMi is http://sedumi.mcmaster.ca/ As of January 2007, the latest version is SeDuMi version 1.1R3. However, this version clashes with MATLAB R2006b. So for users of MATLAB R2006b, the best version of SeDuMi right now appears to be SeDuMi 1.1R2 which can also be downloaded from the above site.

I. LP Problems

The solution of the primal standard-form LP problem

$$\begin{aligned} &\text{minimize} && \boldsymbol{c}^T\boldsymbol{x} && \text{(1a)}\\ &\text{subject to:} && \boldsymbol{A}\boldsymbol{x}=\boldsymbol{b} && \text{(1b)}\\ & && \boldsymbol{x}\geq\boldsymbol{0} && \text{(1c)} \end{aligned}$$

can be obtained by using command x = sedumi(A,b,c); Alternatively, both the solution of the problem in (1) and the solution of the dual problem

$$\begin{aligned} &\text{maximize} && \boldsymbol{b}^T\boldsymbol{y} && \text{(2a)}\\ &\text{subject to:} && -\boldsymbol{A}^T\boldsymbol{y}\geq-\boldsymbol{c} && \text{(2b)} \end{aligned}$$

can be found by using command [x, y, info] = sedumi(A,b,c); where "info" contains information about validity of the solutions obtained:

(1) pinf = dinf = 0: $\boldsymbol{x}$ is an optimal solution and $\boldsymbol{y}$ certifies optimality, viz. $\boldsymbol{b}^T\boldsymbol{y} = \boldsymbol{c}^T\boldsymbol{x}$ and $\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y} \geq 0$. Stated otherwise, $\boldsymbol{y}$ is an optimal solution to maximize $\boldsymbol{b}^T\boldsymbol{y}$ such that $\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y} \geq 0$.

If size(A,2) = length(b), then $\boldsymbol{y}$ solves the linear program maximize $\boldsymbol{b}^T\boldsymbol{y}$ such that $\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y} \geq 0$.

(2) pinf = 1: there cannot be $\boldsymbol{x} \geq \boldsymbol{0}$ with $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, and this is certified by $\boldsymbol{y}$, viz. $\boldsymbol{b}^T\boldsymbol{y} > 0$ and $\boldsymbol{A}^T\boldsymbol{y} \leq 0$. Thus $\boldsymbol{y}$ is a Farkas solution.

(3) dinf = 1: there cannot be $\boldsymbol{y}$ such that $\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y} \geq 0$, and this is certified by $\boldsymbol{x}$, viz. $\boldsymbol{c}^T\boldsymbol{x} < 0$, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, $\boldsymbol{x} \geq \boldsymbol{0}$. Thus $\boldsymbol{x}$ is a Farkas solution.

Example 1 Consider the standard-form LP problem in Example 11.9 [Antoniou and Lu]:

$$
\begin{aligned}
\text{minimize} \quad & f(x) = 2x_1 + 9x_2 + 3x_3 \\
\text{subject to:} \quad & -2x_1 + 2x_2 + x_3 - x_4 = 1 \\
& x_1 + 4x_2 - x_3 - x_5 = 1 \\
& x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0
\end{aligned} \tag{3}
$$

The MATLAB code listed below uses SeDuMi to solve the problem:

```
A = [-2 2 1 -1 0; 1 4 -1 0 -1];
b = [1 1]';
c = [2 9 3 0 0]';
x = sedumi(A,b,c);
```

which gives x = [0 0.3333 0.3333 0 0]'.

Example 2 Now let us consider the LP problem in Example 11.2 [Antoniou and Lu]:

$$
\begin{aligned}
\text{minimize} \quad & -x_1 - 4x_2 \\
\text{subject to:} \quad & x_1 \geq 0 \\
& -x_1 \geq -2 \\
& x_2 \geq 0 \\
& -x_1 - x_2 + 3.5 \geq 0 \\
& -x_1 - 2x_2 + 6 \geq 0
\end{aligned} \tag{4}
$$

By changing the variable $\boldsymbol{x}$ to $\boldsymbol{y}$ and defining

$$
\boldsymbol{A} = \begin{bmatrix} -1 & 1 & 0 & 1 & 1 \\ 0 & 0 & -1 & 1 & 2 \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}, \ \boldsymbol{c} = \begin{bmatrix} 0 & 2 & 0 & 3.5 & 6 \end{bmatrix}^T
$$

the problem in (4) can be expressed as the one in (2). Hence the LP problem in (4) can be solved by using the following MATLAB code:

```
b = [1 4]';
A = [-1 1 0 1 1; 0 0 -1 1 2];
c = [0 2 0 3.5 6]';
[x, y, info] = sedumi(A,b,c);
```

where y can be taken as the solution of the problem in (4). The result is given by y = [0 3]'. In addition, output "info" provides the following information:

```
    cpusec: 0.0938
      iter: 4
 feasratio: 1
      pinf: 0
      dinf: 0
   numerr: 0
```

II. SOCP Problems

We now consider the SOCP formulation given by Eq. (14.104) (see [Antoniou and Lu]), i.e.,

$$\text{minimize} \quad \boldsymbol{b}^T \boldsymbol{x} \tag{5a}$$

$$\text{subject to:} \quad \left\| \boldsymbol{A}_i^T \boldsymbol{x} + \boldsymbol{c}_i \right\| \le \boldsymbol{b}_i^T \boldsymbol{x} + d_i \quad \text{for } i = 1, ..., q \tag{5b}$$

where $\boldsymbol{b} \in R^{m\times 1}, \boldsymbol{x} \in R^{m\times 1}, \boldsymbol{A}_i \in R^{m\times(n_i-1)}, \boldsymbol{b}_i \in R^{m\times 1}, \boldsymbol{c}_i \in R^{(n_i-1)\times 1}$, and $d_i \in R$ for $1 \le i \le q$. In order to use the toolbox to solve the problem in (5), we define matrix $\boldsymbol{A}_t$, vectors $\boldsymbol{b}_t$ and $\boldsymbol{c}_t$ as follows:

$$\begin{aligned}
\boldsymbol{A}_t &= \begin{bmatrix} \boldsymbol{A}_t^{(1)} & \boldsymbol{A}_t^{(2)} & \cdots & \boldsymbol{A}_t^{(q)} \end{bmatrix} \\
\boldsymbol{A}_t^{(i)} &= -\begin{bmatrix} \boldsymbol{b}_i & \boldsymbol{A}_i \end{bmatrix} \\
\boldsymbol{b}_t &= -\boldsymbol{b} \\
\boldsymbol{c}_t &= \begin{bmatrix} \boldsymbol{c}_t^{(1)}; & \boldsymbol{c}_t^{(2)}; & \cdots & \boldsymbol{c}_t^{(q)} \end{bmatrix} \\
\boldsymbol{c}_t^{(i)} &= \begin{bmatrix} d_i; & \boldsymbol{c}_i \end{bmatrix}
\end{aligned}$$

where $\boldsymbol{A}_t \in R^{m\times n}, \boldsymbol{b}_t \in R^{m\times 1}$, and $\boldsymbol{c}_t \in R^{n\times 1}$ with $n = \sum_{i=1}^{q} n_i$.

In addition, define a q-dimensional vector $\boldsymbol{q} = [n_1 \;\; n_2 \;\; \ldots \; n_q]$ which describes the dimensions of the q conic constraints involved in (5b).

Once the data set $\{\boldsymbol{A}_t, \boldsymbol{b}_t, \boldsymbol{c}_t\}$ and vector $\boldsymbol{q}$ have been prepared, the MATLAB commands for solving the SOCP problem in (5) are as follows:

```
K.q = q;
[xs,ys,info] = sedumi(At,bt,ct,K);
info
x = ys;
```

Example 3 Consider the problem described in Example 14.5 in [Antoniou and Lu]: Find the shortest distance between the two ellipses which are defined by

$$c_1(\boldsymbol{x}) = -\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} \frac{1}{4} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix} + \frac{3}{4} \ge 0$$

$$c_2(\boldsymbol{x}) = -\frac{1}{8}\begin{bmatrix} x_3 & x_4 \end{bmatrix} \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} x_3 & x_4 \end{bmatrix} \begin{bmatrix} \frac{11}{2} \\ \frac{13}{2} \end{bmatrix} - \frac{35}{2} \ge 0$$

The problem can be formulated as the constrained minimization problem

$$\begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}) = (x_1 - x_3)^2 + (x_2 - x_4)^2 \\ &\text{subject to:} \quad c_1(\boldsymbol{x}) \geq 0 \ \text{ and } \ c_2(\boldsymbol{x}) \geq 0 \end{aligned}$$

By introducing an upper bound δ for the square root of the objective function, the above problem can be expressed as

$$\begin{aligned} &\text{minimize} \quad \delta \\ &\text{subject to:} \quad \left[(x_1 - x_3)^2 + (x_2 - x_4)^2\right]^{1/2} \leq \delta \\ &\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1/4 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1/2 \\ 0 \end{bmatrix} \leq \frac{3}{4} \\ &\begin{bmatrix} x_3 & x_4 \end{bmatrix} \begin{bmatrix} 5/8 & 3/8 \\ 3/8 & 5/8 \end{bmatrix} \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} - \begin{bmatrix} x_3 & x_4 \end{bmatrix} \begin{bmatrix} 11/2 \\ 13/2 \end{bmatrix} \leq -\frac{35}{2} \end{aligned}$$

If we augment the decision vector by including the upper bound δ in it, i.e., $\boldsymbol{x} = [\delta \ x_1 \ x_2 \ x_3 \ x_4]^T$, then the above problem is equivalent to the problem in (5) with $q = 3$,

$$\boldsymbol{b} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

$$\boldsymbol{A}_1^T = \begin{bmatrix} 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 \end{bmatrix}, \ \boldsymbol{A}_2^T = \begin{bmatrix} 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{A}_3^T = \begin{bmatrix} 0 & 0 & 0 & -0.7071 & -0.7071 \\ 0 & 0 & 0 & -0.3536 & 0.3536 \end{bmatrix}$$

$$\boldsymbol{b}_1 = \boldsymbol{b}, \ \boldsymbol{b}_2 = \boldsymbol{0}, \ \boldsymbol{b}_3 = \boldsymbol{0}$$

$$\boldsymbol{c}_1 = \boldsymbol{0}, \ \boldsymbol{c}_2 = \begin{bmatrix} -0.5 & 0 \end{bmatrix}^T, \ \boldsymbol{c}_3 = \begin{bmatrix} 4.2426 & -0.7071 \end{bmatrix}^T$$

$$d_1 = 0, \ d_2 = 1, \ \text{and} \ d_3 = 1.$$

The MATLAB code listed below solves the SOCP problem using SeDuMi.

```
b = [1 0 0 0 0]';
A1 = [0 -1 0 1 0; 0 0 1 0 -1]';
A2 = [0 0.5 0 0 0; 0 0 1 0 0]';
A3 = [0 0 0 -0.7071 -0.7071; 0 0 0 -0.3536 0.3536]';
b1 = b;
b2 = zeros(5,1);
b3 = b2;
c1 = [0 0]';
c2 = [-0.5 0]';
c3 = [4.2426 -0.7071]';
d1 = 0;
d2 = 1;
d3 = 1;
At1 = -[b1 A1];
At2 = -[b2 A2];
At3 = -[b3 A3];
```

```
At = [At1 At2 At3];
bt = -b;
ct1 = [d1; c1];
ct2 = [d2; c2];
ct3 = [d3; c3];
ct = [ct1; ct2; ct3];
K.q = [size(At1,2)  size(At2,2)  size(At3,2)];
[xs, ys, info] = sedumi(At,bt,ct,K);
x = ys;
r_s = x(2:3);
s_s = x(4:5);
disp('solution points in regions R and S are:')
[r_s s_s]
disp('minimum distance:')
norm(r_s – s_s)
```

It was found that r_s = [2.0447 0.8527]' and s_s = [2.5448 2.4857]' which give the minimum distance 1.7078. Additional information provided by the toolbox is as follows:

```
info =
      cpusec: 0.0469
        iter: 9
   feasratio: 1.0000
        pinf: 0
        dinf: 0
      numerr: 0
```

III. SOCP Problems with Linear Constraints

SeDuMi can be used to solve SOCP problems with additional linear constraints. As an example, we consider the following SOCP problem

$$\begin{aligned} &\text{minimize} && \boldsymbol{b}^T \boldsymbol{x} && (6a)\\ &\text{subject to:} && \boldsymbol{D}^T \boldsymbol{x} + \boldsymbol{f} \geq \boldsymbol{0} && (6b)\\ & && \left\| \boldsymbol{A}_i^T \boldsymbol{x} + \boldsymbol{c}_i \right\| \leq \boldsymbol{b}_i^T \boldsymbol{x} + d_i \quad \text{for } i = 1,...,q && (6c) \end{aligned}$$

where $\boldsymbol{D} \in R^{m \times p}$ and $\boldsymbol{f} \in R^{p \times 1}$ and other entries are defined in part II. In order to use the toolbox to solve the problem in (6), we define matrix $\boldsymbol{A}_t$, vectors $\boldsymbol{b}_t$ and $\boldsymbol{c}_t$, which are obtained by augmenting the corresponding quantities used for the problem in (5), as follows:

$$\begin{aligned} &\boldsymbol{A}_t = \left[-\boldsymbol{D} \quad \boldsymbol{A}_t^{(1)} \quad \boldsymbol{A}_t^{(2)} \quad \cdots \quad \boldsymbol{A}_t^{(q)} \right] \\ &\boldsymbol{A}_t^{(i)} = -\left[\boldsymbol{b}_i \quad \boldsymbol{A}_i \right] \\ &\boldsymbol{b}_t = -\boldsymbol{b} \\ &\boldsymbol{c}_t = \left[\boldsymbol{f}; \quad \boldsymbol{c}_t^{(1)}; \quad \boldsymbol{c}_t^{(2)}; \quad \cdots \quad \boldsymbol{c}_t^{(q)} \right] \\ &\boldsymbol{c}_t^{(i)} = \left[d_i; \quad \boldsymbol{c}_i \right] \end{aligned}$$

Now define K.l = p where p is the number of linear constraints in (6b). And as before, define a q-dimensional vector $\boldsymbol{q} = [n_1 \;\; n_2 \;\; \ldots \;\; n_q]$ to describe the dimensions of the q conic constraints in (6c). The MATLAB commands to solve the SOCP problem in (5) are as follows:

```
K.l = m;
K.q = q;
[xs,ys,info] = sedumi(At,bt,ct,K);
info
x = ys;
```

Example 4 Consider a problem similar to the one described in Example 14.5 of [Antoniou and Lu], to that we now add three additional linear constraints for variables x_3 and x_4 as follows:

$$-x_3 + 2.4 \geq 0$$
$$x_4 - 2.4 \geq 0$$
$$1.5x_3 - x_4 + 2.4 \geq 0$$

It can be readily verified that the problem at hand is a minimum distance problem for the ellipse $\mathcal{R}$ and convex body $\mathcal{S}$ shown in the next figure.

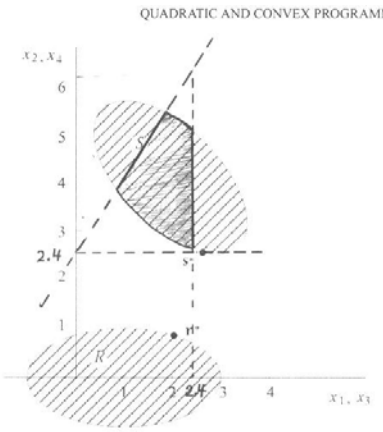


Figure 13.5. Distance between two ellipses (Example 13.5).

The MATLAB code listed below solves this SOCP problem using SeDuMi.

```
D = [0 0 0; 0 0 0; 0 0 0; -1 0 1.5; 0 1 -1];
f = [2.4 -2.4 2.4]';
b = [1 0 0 0 0]';
A1 = [0 -1 0 1 0; 0 0 1 0 -1]';
A2 = [0 0.5 0 0 0; 0 0 1 0 0]';
A3 = [0 0 0 -0.7071 -0.7071; 0 0 0 -0.3536 0.3536]';
b1 = b;
b2 = zeros(5,1);
b3 = b2;
c1 = [0 0]';
c2 = [-0.5 0]';
c3 = [4.2426 -0.7071]';
d1 = 0;
```

```
d2 = 1;
d3 = 1;
At1 = -[b1 A1];
At2 = -[b2 A2];
At3 = -[b3 A3];
At = [-D At1 At2 At3];
bt = -b;
ct1 = [d1; c1];
ct2 = [d2; c2];
ct3 = [d3; c3];
ct = [f; ct1; ct2; ct3];
K.l = size(D,2);
K.q = [size(At1,2)  size(At2,2)  size(At3,2)];
[xs, ys, info] = sedumi(At,bt,ct,K);
x = ys;
r_s = x(2:3);
s_s = x(4:5);
disp('solution points in regions R and S are:')
[r_s s_s]
disp('minimum distance:')
norm(r_s – s_s)
```

It was found that r_s = [1.9518 0.8795]' and s_s = [2.4000 2.5362]' which give the minimum distance 1.7163. Additional information provided by the toolbox are as follows:

```
info =
      cpusec: 0.6406
        iter: 13
   feasratio: 1.0000
        pinf: 0
        dinf: 0
      numerr: 0
```

IV. SDP Problems

Consider the "standard" dual SDP problem

$$\text{minimize} \quad \boldsymbol{c}^T \boldsymbol{y} \tag{7a}$$

$$\text{subject to: } \boldsymbol{F}_0 + \boldsymbol{y}_1 \boldsymbol{F}_1 + \cdots \boldsymbol{y}_p \boldsymbol{F}_p \succeq \boldsymbol{0} \tag{7b}$$

where vector $\boldsymbol{y} = [y_1 \;\; y_2 \;\; \ldots \;\; y_p]^T$, and $\boldsymbol{F}_i$ are symmetric matrices. In order to use SeDuMi to solve the SDF problem, we define

$\boldsymbol{b}_t = -\boldsymbol{c}$;
$\boldsymbol{c}_t = \text{vec}(\boldsymbol{F}_0)$;
$\boldsymbol{A}_t(:, i) = -\text{vec}(\boldsymbol{F}_i)$ for $i = 1, 2, \ldots, p$

where vec($\boldsymbol{F}$) converts matrix $\boldsymbol{F}$ into a column vector by stacking all columns of $\boldsymbol{F}$. The MATLAB code using commands from SeDuMi is as follows.

```
p = length(c);
bt = -c;
ct = vec(F0);
for i = 1:p, At(:,i) = -vec(Fi); end
K.s = size(F0,1);
[x, y, info] = sedumi(At,bt,ct,K);
info
```

where y gives the solution of the problem in (7) and info provides information about the validity of the solution.

Example 5 Solve the problem in Example 14.1 of [Antoniou and Lu]: Find scalars y_1, y_2, and y_3 such that the maximum eigenvalue of $\boldsymbol{F} = \boldsymbol{A}_0 + y_1\boldsymbol{A}_1 + y_2\boldsymbol{A}_2 + y_3\boldsymbol{A}_3$ with

$$\boldsymbol{A}_0 = \begin{bmatrix} 2 & -0.5 & -0.6 \\ -0.5 & 2 & 0.4 \\ -0.6 & 0.4 & 3 \end{bmatrix}, \ \boldsymbol{A}_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{A}_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \boldsymbol{A}_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

is minimized. This problem can be formulated as the SDP problem

$$\text{minimize} \quad t \tag{8a}$$

$$\text{subject to:} \quad t\boldsymbol{I} - (\boldsymbol{A}_0 + y_1\boldsymbol{A}_1 + y_2\boldsymbol{A}_2 + y_3\boldsymbol{A}_3) \succeq \boldsymbol{0} \tag{8b}$$

With $\boldsymbol{y} = [y_1 \ \ y_2 \ \ y_3 \ \ t]^T$, $F_0 = -\boldsymbol{A}_0$, $F_1 = -\boldsymbol{A}_1$, $F_2 = -\boldsymbol{A}_2$, and $\boldsymbol{c} = [0 \ \ 0 \ \ 0 \ \ 1]^T$, problem (8) becomes the standard SDP problem in (7). The MATLAB code listed below uses SeDuMi to solve the problem in (8).

```
A0 = [2 -0.5 -0.6; -0.5 2 0.4; -0.6 0.4 3];
A1 = [0 1 0; 1 0 0; 0 0 0];
A2 = [0 0 1; 0 0 0; 1 0 0];
A3 = [0 0 0; 0 0 1; 0 1 0];
F0 = -A0;
F1 = -A1;
F2 = -A2;
F3 = -A3;
F4 = eye(3);
At = -[vec(F1) vec(F2) vec(F3) vec(F4)];
bt = -[0 0 0 1]';
ct = vec(F0);
K.s = size(F0,1);
[x,y,info] = sedumi(At,bt,ct,K);
info
```

The result obtained is given by y = [0.5 0.6 -0.4 3]'; which says the minimum of the largest eigenvalue is 3 that can be achieved by the choice $y_1 = 0.5$, $y_2 = 0.6$, and $y_3 = -0.4$. The 'info' gives the following:

```
cpusec: 0.5000
iter: 5
feasratio: 1.0000
pinf: 0
dinf: 0
numerr: 0
```

V. SDP Problems with Linear Constraints

In principal linear constraints can be formulated as SDP constraints (see Chap. 14 of [Antoniou and Lu]), but one can take the advantage of SeDuMi that allows both linear and SDP constraints simultaneously to make the MATLAB code more efficient. To this end, we consider the problem

$$\begin{aligned} &\text{minimize} \quad \boldsymbol{c}^T \boldsymbol{y} && \text{(9a)} \\ &\text{subject to:} \quad \boldsymbol{A}\boldsymbol{y} \geq \boldsymbol{b} && \text{(9b)} \\ &\qquad\qquad \boldsymbol{F}_0 + \boldsymbol{y}_1\boldsymbol{F}_1 + \cdots \boldsymbol{y}_p\boldsymbol{F}_p \succeq \boldsymbol{0} && \text{(9c)} \end{aligned}$$

Similar to that in part (IV), we define

$\boldsymbol{b}_t = -\boldsymbol{c}$;
$\boldsymbol{c}_t = \text{vec}(\boldsymbol{F}_0)$;
$\boldsymbol{A}_t(:, i) = -\text{vec}(\boldsymbol{F}_i)$ for $i = 1, 2, \ldots, p$

The MATLAB code using commands from SeDuMi is as follows.

```
p = length(c);
btt = -c;
ctt = [-b; vec(F0)];
for i = 1:p, At(:,i) = -vec(Fi); end
Att  = [-A; At];
K.l = size(A,1);
K.s = size(F0,1);
[x, y, info] = sedumi(At,bt,ct,K);
info
```

where y gives the solution of the problem in (9) and info provides information about the validity of the solution.

Example 6 We now consider a problem similar to the one in Example 5, but we add additional constraints that parameters y_1, y_2, y_3 satisfy $0.7 \leq y_1 \leq 1$, $0 \leq y_2 \leq 0.3$, and $y_3 \geq 0$. These additional linear constraints can be put into a matrix form $\boldsymbol{A}\boldsymbol{y} \geq \boldsymbol{b}$ with

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 0.7 \\ -1 \\ 0 \\ -0.3 \\ 0 \end{bmatrix}$$

where vector y is defined as in Example 5, i.e., $\boldsymbol{y} = [y_1 \;\; y_2 \;\; y_3 \;\; t]^T$. The MATLAB code solving this problem using SeDuMi is as follows:

```
A = [1 0 0 0; -1 0 0 0; 0 1 0 0; 0 -1 0 0; 0 0 1 0];
b = [0.7 -1 0 -0.3 0]';
A0 = [2 -0.5 -0.6; -0.5 2 0.4; -0.6 0.4 3];
```

```
A1 = [0 1 0; 1 0 0; 0 0 0];
A2 = [0 0 1; 0 0 0; 1 0 0];
A3 = [0 0 0; 0 0 1; 0 1 0];
F0 = -A0;
F1 = -A1;
F2 = -A2;
F3 = -A3;
F4 = eye(3);
At = -[vec(F1) vec(F2) vec(F3) vec(F4)];
Att = [-A; At];
btt = -[0 0 0 1]';
ct = vec(F0);
ctt = [-b; ct];
K.l = size(A,1);
K.s = size(F0,1);
[x,y,info] = sedumi(Att,btt,ctt,K);
info
```

The result obtained is given by y = [1.0 0.3 -0.0 3.1556]'; which says the minimum of the largest eigenvalue is 3.1556 that can be achieved by the choice $y_1 = 1.0$, $y_2 = 0.3$, and $y_3 = 0$. The info gives the following:

```
iter: 9
feasratio: 0.9967
pinf: 0
dinf: 0
numerr: 0
timing: [0.0313 0.2969 0.0156]
cpusec: 0.3438
```

VI. Other Problems

6.1 [x, y, info] = sedumi(A, b, 0) solves the feasibility problem: Find $\boldsymbol{x}$ such that $\boldsymbol{Ax} = \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$.

Example 7 With $\boldsymbol{A}$ and $\boldsymbol{b}$ given in Example 1, [x, y, info] = sedumi(A, b, 0) yields $\boldsymbol{x}$ = [1.3130 1.2353 2.8405 1.6851 2.4136]^T and

```
info =
cpusec: 0
iter: 1
feasratio: 1
pinf: 0
dinf: 0
numerr: 0
```

6.2 [x, y, info] = sedumi(A, 0, c) solves the feasibility problem: Find $\boldsymbol{y}$ such that $\boldsymbol{A}^T\boldsymbol{y} \leq \boldsymbol{c}$.

Example 8 Let

$$\boldsymbol{A} = \begin{bmatrix} -1 & 1 & 0 & 1 & 1 \\ 0 & 0 & -1 & 1 & 2 \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} 0 & 2 & 0 & 3.5 & 6 \end{bmatrix}^T$$

Applying [x, y, info] = sedumi(A, 0, c) yields

$\boldsymbol{y} = [0.9810 \;\; 1.2670]^T$ and

info =
cpusec: 0.0625
iter: 3
feasratio: 1
pinf: 0
dinf: 0
numerr: 0

Variational Methods in Optimization

References

1. I. M. Gelfand and S. V. Formin, *Calculus of Variations*, Dover Publications Inc., 2000.
2. L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D.*, vol. 60, pp. 259-268, 1992.
3. L. Rudin and S. Osher, "Total variation based image restoration with free local constraints," *Proc. ICIP*, pp. 31-35, Austin, TX., 1994.
4. S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin, "An iterative regularization method for total variation-based image restoration," *SIAM Journal on Multiscale Modeling and Simulation*, vol. 4, no. 2, pp. 460-489, 2005.
5. Y. Meyer, *Oscillating Patterns in Image Processing and Nonlinear Evolution Equations*, University Lecture Series, vol. 22, American Mathematical Society, Providence, RI., 2001.
6. UCLA Reports: http://www.math.ucla.edu/applied/cam/index.html

1. Elements of Calculus of Variations

1.1 *Functionals*

Roughly speaking, a functional is a "function of function". We all know the definition of function, say $f(x)$ – which is taken to mean a correspondence that assigns a definite (real) number for each variable value x in a certain domain. Similarly, a *functional* is a correspondence that assigns a real number to each *function* (such as a curve, or a surface, etc.) that belongs to some class of functions.

For example, for each curve that connects two fixed points $\boldsymbol{a}$ and $\boldsymbol{b}$, the length of the curve is a functional. As another example, the time required for a mass point traveling down along a smooth and monotonically decreasing plane curve with two fixed ends is a functional. From these examples, we see that a functional is defined over a certain class of functions.

1.2 *Variational Problems: Examples*

Example 1 A smooth plane curve that joins points $\boldsymbol{p}_a = (a, A)$ and $\boldsymbol{p}_b = (b, B)$ can be described by a differentiable function $y = y(x)$ with $A = y(a)$ and $B = y(b)$, and its length is known to be

$$J[y] = \int_a^b \sqrt{1 + y'^2}\, dx \qquad (1)$$

Here $J[y]$ is obviously a functional for the class of smooth functions $y(x)$ that satisfy the boundary conditions $A = y(a)$ and $B = y(b)$. A simple variational problem associated with the functional in (1) is to find the shortest curve joining points $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$.

Typically, for the one-variable case a functional of interest assumes the form

$$J[y] = \int_a^b F(x, y, y')\, dx \tag{2}$$

and for the two-variable case

$$J[u] = \iint_R F(x, y, u, u_x, u_y)\, dxdy \tag{3}$$

where $u = u(x, y)$, $u_x = \partial u / \partial x$ and $u_y = \partial u / \partial y$.

Example 2 Let $u(x, y)$ denote a two-variable smooth function that models the light intensity of an (analog) image over region R. The total variation (TV) of the image is defined as

$$J[u] = \iint_R \sqrt{u_x^2 + u_y^2}\, dx\, dy \tag{4}$$

which is obviously a functional of type (3) for the class of smooth functions defined over region R. It has been argued that the noise removal problem for images can be cast as minimizing the TV subject to some statistical constraints.

1.3 *Several Lemmas* [1]

The key analytical tool for investigating various image processing problems in a variational optimization setting is the Euler-Lagrange (EL) equation. The lemmas presented below are instrumental in deriving the EL equation.

In what follows the set of continuous functions over interval $[a, b]$ is denoted by $C(a, b)$ and the set of functions that are continuous and have up to
p-th order continuous derivatives on $[a, b]$ is denoted by $D_p(a, b)$.

Lemma 1 If $\alpha(x)$ is continuous on $[a, b]$, and if

$$\int_a^b \alpha(x) h(x) dx = 0$$

for every function $h(x) \in C(a, b)$ such that $h(a) = h(b) = 0$, then $\alpha(x) = 0$ for all x in $[a, b]$.

Proof. The lemma will be proved by the method of contradiction. Suppose the function $\alpha(x)$ is nonzero, say positive, at some point in $[a, b]$. Then $\alpha(x)$ is also positive in some interval $[x_1, x_2]$ contained in $[a, b]$. If we set

$$h(x) = (x - x_1)(x_2 - x)$$

for x in $[x_1, x_2]$ and $h(x) = 0$ otherwise, then $h(x)$ obviously satisfies the condition of the lemma. However,

$$\int_a^b \alpha(x)h(x)\,dx = \int_{x_1}^{x_2} \alpha(x)(x-x_1)(x_2-x)dx > 0$$

because the integrand is positive (except at x_1 and x_2). This contradiction proves the lemma. ■

Lemma 2 If $\alpha(x)$ is continuous in [a, b], and if

$$\int_a^b \alpha(x)h'(x)dx = 0$$

for every $h(x) \in D_1(a,b)$ such that $h(a) = h(b) = 0$, then $\alpha(x) = c$ for all x in [a, b], where c is a constant.

Proof. Let c be a constant given by

$$c = \frac{1}{b-a}\int_a^b \alpha(x)dx$$

which implies that

$$\int_a^b [\alpha(x) - c]dx = 0\,,$$

and define function $h(x)$ as

$$h(x) = \int_a^x [\alpha(\xi) - c]d\xi\,.$$

Evidently, $h(x)$ belongs to $D_1(a, b)$ and satisfies $h(a) = 0$ and $h(b) = 0$. Then on the one hand,

$$\int_a^b [\alpha(x) - c]h'(x)dx = \int_a^b \alpha(x)h'(x)dx - c[h(b) - h(a)] = 0\,.$$

while on the other hand,

$$\int_a^b [\alpha(x) - c]h'(x)dx = \int_a^b [\alpha(x) - c]^2\,dx\,.$$

Therefore, $\alpha(x) - c = 0$ i.e. $\alpha(x) = c$ for all x in [a, b]. ■

Lemma 3 If $\alpha(x)$ is continuous in [a, b], and if

$$\int_a^b \alpha(x)h''(x)dx = 0$$

for every $h(x) \in D_2(a,b)$ such that $h(a) = h(b) = 0$ and $h'(a) = h'(b) = 0$, then $\alpha(x) = c_0 + c_1 x$ for all x in $[a, b]$, where c_0 and c_1 are constants.

Proof. Let c_0 and c_1 be constants defined by the conditions

$$\int_a^b [\alpha(x) - c_0 - c_1 x]\,dx = 0$$

$$\int_a^b dx \int_a^x [\alpha(\xi) - c_0 - c_1 \xi]\,d\xi = 0$$

and let

$$h(x) = \int_a^x d\xi \int_a^\xi [\alpha(t) - c_0 - c_1 t]\,dt$$

It is easy to verify that $h(x) \in D_2(a,b)$ and satisfies $h(a) = h(b) = 0$ and $h'(a) = h'(b) = 0$. Then on the one end,

$$\int_a^b [\alpha(x) - c_0 - c_1 x] h''(x)\,dx$$

$$= \int_a^b \alpha(x) h''(x)\,dx - c_0[h'(b) - h'(a)] - c_1 \int_a^b x h''(x)\,dx$$

$$= -c_1[bh'(b) - ah'(a)] - c_1[h(b) - h(a)] = 0$$

while on the other hand,

$$\int_a^b [\alpha(x) - c_0 - c_1 x] h''(x)\,dx = \int_a^b [\alpha(x) - c_0 - c_1 x]^2\,dx = 0$$

It follows that $\alpha(x) - c_0 - c_1 x = 0$, hence $\alpha(x) = c_0 + c_1 x$ for all x in $[a, b]$. ■

Lemma 4 If $\alpha(x)$ and $\beta(x)$ are continuous in $[a, b]$, and if

$$\int_a^b [\alpha(x) h(x) + \beta(x) h'(x)]\,dx = 0 \qquad (5)$$

for every function $h(x) \in D_1(a,b)$ such that $h(a) = h(b) = 0$, then $\beta(x)$ is differentiable, and $\beta'(x) = \alpha(x)$ for all x in $[a, b]$.

Proof. Let $A(x)$ be the function defined by

$$A(x) = \int_a^x \alpha(\xi)\, d\xi$$

Using integration by parts, we can verify that

$$\int_a^b \alpha(x)h(x)\, dx = -\int_a^b A(x)h'(x)\, dx$$

hence (5) can be written as

$$\int_a^b [-A(x) + \beta(x)]h'(x)\, dx = 0 \qquad (6)$$

By Lemma 2, (6) implies that $\beta(x) - A(x) = \text{const}$, i.e., $\beta(x) = A(x) + \text{const}$, thus function $\beta(x)$ is differentiable and by definition of $A(x)$, $\beta'(x) = \alpha(x)$. ■

1.4 *Variation of a Functional*

Let $J[y]$ be a functional defined on some normed linear space and let

$$\Delta J[h] = J[y + h] - J[y]$$

be its increment corresponding to the increment $h = h(x)$ of the "independent variable" $y = y(x)$. If y is fixed, then $\Delta J[h]$ is a functional of h, in general a nonlinear functional. Suppose that

$$\Delta J[h] = \varphi[h] + \varepsilon \|h\|$$

where $\varphi[h]$ is a linear functional and $\varepsilon \to 0$ as $\|h\| \to 0$. Then the functional $J[y]$ is said to be *differentiable*, and the principal linear part of the increment $\Delta J[h]$, i.e., the linear functional $\varphi[h]$ which differs from $\Delta J[h]$ by an infinitesimal of order higher than 1 relative to $\|h\|$, is called the variation of $J[y]$ and is denoted by $\delta J[h]$. It can be shown that the variation of a differentiable functional is unique [1].

We now use the concept of the variation of a functional to establish a necessary condition for a functional to have an extremum.

Similar to the concept of the extremum of a function from analysis, we say that the functional $J[y]$ has a relative extremum for $y = \hat{y}$ if $J[y] - J[\hat{y}]$ does not change its sign in some neighborhood of the curve $y = \hat{y}(x)$.

At this point, it is important that the meaning of the term "neighborhood" is explicitly clarified. To this end we need to define two norms, one for space $C(a, b)$ and the other for $D_1(a, b)$:

♦ For a function $y(x)$ in space $C(a, b)$, we define

$$\|y\|_0 = \max_{a \le x \le b} |\, y(x) \,|$$

♦ For a function $y(x)$ in space $D_1(a, b)$, we define

$$\|y\|_1 = \max_{a \le x \le b} |\, y(x) \,| + \max_{a \le x \le b} |\, y'(x) \,|$$

Depending on which of these two norms (topology) is employed, a *neighborhood* of the curve $y = \hat{y}(x)$ may be referred to those y such that $\|y - \hat{y}\|_0 < \varepsilon$, or to those y such that $\|y - \hat{y}\|_1 < \varepsilon$. Accordingly, we shall say that the functional $J[y]$ has a weak extremum for $y = \hat{y}$, if there exist an $\varepsilon > 0,$ such that $J[y] - J[\hat{y}]$ has the same sign for all y in the neighborhood of $\hat{y}(x)$ characterized by $\|y - \hat{y}\|_1 < \varepsilon$. On the other hand, we shall say that the functional $J[y]$ has a strong extremum for $y = \hat{y}$, if there exist an $\varepsilon > 0,$ such that $J[y] - J[\hat{y}]$ has the same sign for all y in the neighborhood of $\hat{y}(x)$ characterized by $\|y - \hat{y}\|_0 < \varepsilon$.

Theorem 1 [1] A necessary condition for the differentiable functional $J[y]$ to have an extremum for $y = \hat{y}$ is that its variation vanish for $y = \hat{y}$, namely,

$$\delta J[h] = 0$$

for $y = \hat{y}$ and all admissible h.

Proof. To be explicit, suppose $J[y]$ has a minimum for $y = \hat{y}$. By the definition of the variation $\delta J[h]$, we have

$$\Delta J[h] = \varphi[h] + \varepsilon \|h\| \qquad (7)$$

where $\varepsilon \to 0$ as $\|h\| \to 0$. Hence, for sufficiently small $\|h\|$, the sign of $\Delta J[h]$ is the same as the sign of $\delta J[h]$. Now, suppose that $\delta J[h_0] \neq 0$ for some admissible h_0. Then for any $\alpha > 0$, no matter how small, we have

$$\delta J[-\alpha h_0] = -\delta J[\alpha h_0]$$

because $\delta J[h]$ is a linear functional of h. Hence (7) can be made to have either sign for arbitrary small $\|h\|$, which is not possible according to the hypothesis $J[y]$ has a minimum for $y = \hat{y}$. This contradiction proves the theorem. ■

1.5 *Euler-Lagrange Equation: The One-Variable Case*

We now study the simplest variational problem with one variable: *Let F(x, y, z) be a function with continuous first and second (partial) derivatives with respect to all its variables. Then, among all functions in $D_1(a, b)$ that satisfy the boundary conditions*

$$y(a) = A, \qquad y(b) = B, \tag{8}$$

find the function for which the functional

$$J[y] = \int_a^b F(x, y, y')\,dx \tag{9}$$

has a weak extremum.

Suppose we give $y(x)$ an increment $h(x)$. In order for the function $y(x) + h(x)$ to continue to satisfy the boundary condition (8), we must impose

$$h(a) = h(b) = 0 \tag{10}$$

The corresponding increment of the functional (9) is given by

$$\Delta J = J[y+h] - J[y] = \int_a^b F(x, y+h, y'+h')dx - \int_a^b F(x, y, y')dx$$

$$= \int_a^b \left[F(x, y+h, y'+h') - F(x, y, y')\right]dx$$

Using Taylor's theorem, it follows that

$$\Delta J = \int_a^b [F_y(x, y, y')h + F_{y'}(x, y, y')h']dx + \cdots\cdots \tag{11}$$

where the subscripts denote partial derivatives with respect to the corresponding arguments, and the dots denote terms of order higher than 1 relative to h and h'. The integral in the right-hand side of (11) represents the principal linear part of the increment ΔJ, hence the variation of $J[y]$ is given by

$$\delta J = \int_a^b \left[F_y(x, y, y')h + F_{y'}(x, y, y')h'\right]dx$$

By Theorem 1, a necessary condition for $J[y]$ to have an extreme for $y = y(x)$ is that

$$\delta J = \int_a^b \left(F_y h + F_{y'}h'\right)dx = 0 \tag{12}$$

for all admissible h. According to Lemma 2, (12) implies that

$$F_y - \frac{d}{dx} F_{y'} = 0 \qquad (13)$$

The equation in (13) is known as the *Euler-Lagrange equation*. Thus we have proved

Theorem 2 Let $J[y]$ be a functional of the form

$$J[y] = \int_a^b F(x, y, y')\, dx$$

defined on the set of functions which have continuous first derivatives in $[a, b]$ and satisfy the boundary conditions $y(a) = A$, $y(b) = B$. Then a necessary condition for $J[y]$ to have an extremum for a given $y(x)$ is that $y(x)$ satisfies the Euler-Lagrange equation

$$F_y - \frac{d}{dx} F_{y'} = 0$$

1.6 *Euler-Lagrange Equation: The Two-Variable Case*

In this section we consider functionals depending on functions of *two variables* that assume the form of

$$J[u] = \iint_R F(x, y, u, u_x, u_y)\, dx\, dy \qquad (14)$$

where R is some closed region and u_x, u_y are partial derivatives of $u = u(x, y)$.

Similar to Lemma 1 (for the one-variable case), we have

Lemma 5 If $\alpha(x, y)$ is a fixed function which is continuous in a closed region R, and if the integral

$$\iint_R \alpha(x, y) h(x, y)\, dx\, dy$$

vanishes for every function $h(x, y)$ which has continuous first and second derivatives in R and equals zero on the boundary Γ of region R, then $\alpha(x, y) = 0$ every in R.

The proof of the above lemma is similar in spirit to that of Lemma 1 and the interested reader is referred to [1].

In order to apply the necessary condition for an extremum of the functional (14), i.e., $\delta J = 0$, we must first calculate the variation δJ. Let $h(x, y)$ be an arbitrary function which has continuous first and second derivative and vanishes in the boundary Γ of R. Then if $u(x, y)$ belongs to the domain of definition, i.e., $u(x, y)$ and its first and second derivatives are continuous in R and $u(x, y)$ takes given values on boundary Γ, so does $u(x, y) + h(x, y)$. It follows that

$$\Delta J = J[u+h] - J[u]$$

$$= \iint_R \Big[F(x, y, u+h, u_x + h_x, u_y + h_y) - F(x, y, u, u_x, u_y) \Big] dx\, dy$$

$$= \iint_R \Big[F_u h + F_{u_x} h_x + F_{u_y} h_y \Big] dx\, dy + \cdots\cdots$$

The integral in the last equality represents the principal linear part of the increment ΔJ, hence the variation of $J[u]$ is given by

$$\delta J = \iint_R \Big[F_u h + F_{u_x} h_x + F_{u_y} h_y \Big] dx\, dy .$$

By using Green's theorem

$$\iint_R \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dx\, dy = \int_\Gamma \big(P\, dx + Q\, dy \big)$$

we can write

$$\iint_R \Big(F_{u_x} h_x + F_{u_y} h_y \Big) dx\, dy$$

$$= \iint_R \left[\frac{\partial}{\partial x} \Big(F_{u_x} h \Big) + \frac{\partial}{\partial y} (F_{u_y} h) \right] dx\, dy - \iint_R \left(\frac{\partial}{\partial x} F_{u_x} + \frac{\partial}{\partial y} F_{u_y} \right) h\, dx\, dy$$

$$= \int_\Gamma \Big(F_{u_x} h\, dy - F_{u_y} h\, dx \Big) - \iint_R \left(\frac{\partial}{\partial x} F_{u_x} + \frac{\partial}{\partial y} F_{u_y} \right) h\, dx\, dy$$

$$= - \iint_R \left(\frac{\partial}{\partial x} F_{u_x} + \frac{\partial}{\partial y} F_{u_y} \right) h\, dx\, dy$$

where the last equality is obtained by the fact that $h(x, y)$ assumes zero value on boundary Γ. Consequently, the variation of the functional $J[u]$ becomes

$$\delta J = \iint_R \left(F_u - \frac{\partial}{\partial x} F_{u_x} - \frac{\partial}{\partial y} F_{u_y} \right) h(x, y)\, dx\, dy \qquad (15)$$

Thus the condition $\delta J = 0$ implies that the integral in (15) vanishes for any $h(x, y)$ satisfying the zero boundary condition. By Lemma 5, this leads to the second-order partial differential equation, again known as *the Euler-Lagrange equation*:

$$F_u - \frac{\partial}{\partial x} F_{u_x} - \frac{\partial}{\partial y} F_{u_y} = 0 \qquad (16)$$

2. Functions of Bounded Variation

2.1 *Bounded Variation Functions of One Variable*

Let $u(x)$ be a function defined over $[a, b]$ on which it always assumes finite values. For a set grid points $a = x_0 < x_1 < \ldots < x_n = b$, we construct the sum

$$V_u(x_0, \cdots, x_n) = \sum_{i=1}^{n} | u(x_i) - u(x_{i-1}) |$$

which is called *the variation of u(x) with respect to grid set* $(x_0, \ldots, x_n)$. If there a constant M such that

$$\sup_{x_0, \ldots, x_n} V_u(x_0, \cdots, x_n) \le M < \infty$$

regardless of the number of grid points, then $u(x)$ is called a function of *bounded variation* (BV), and

$$\bigvee_a^b (u) = \sup_{x_0, \ldots, x_n} V_u(x_0, \cdots, x_n)$$

is called the total variation (TV) of $u(x)$ on $[a, b]$. In what follows, "$u(x)$ is BV" is taken to mean that $u(x)$ is a function of BV.

Example 1 Let $u(x)$ be a function defined on $[a, b]$ that satisfies the Lipschitz condition, namely, there exists a constant M such that, for any x and x' on $[a, b]$, $| u(x) - u(x') | \le M | x - x' |$ holds. It follows that

$$V_u(x_0, \cdots, x_n) = \sum_{i=1}^{n} | u(x_i) - u(x_{i-1}) | \le \sum_{i=1}^{n} M | x_i - x_{i-1} | = M(b - a)$$

which implies that $\bigvee_a^b (u) \le M(b - a)$, hence $u(x)$ is BV. In addition, if we use Lagrange's formula $u(x_1) - u(x_2) = u'(\xi) \cdot (x_1 - x_2)$ for some $\xi \in [x_1, x_2]$, then we conclude that if $u(x)$ has bounded first-order derivative on $[a, b]$, then $u(x)$ is BV. ■

Example 2 A continuous function is not necessarily BV. For example, function $u(x)$ defined by

$$u(x) = \begin{cases} x \sin \dfrac{1}{x} & \text{for } 0 < x \le 1 \\ 0 & \text{for } x = 0 \end{cases}$$

is continuous on [0, 1]. If we take

$$x_n = \frac{1}{n\pi + \dfrac{\pi}{2}}, \qquad n = 0, 1, \ldots, m$$

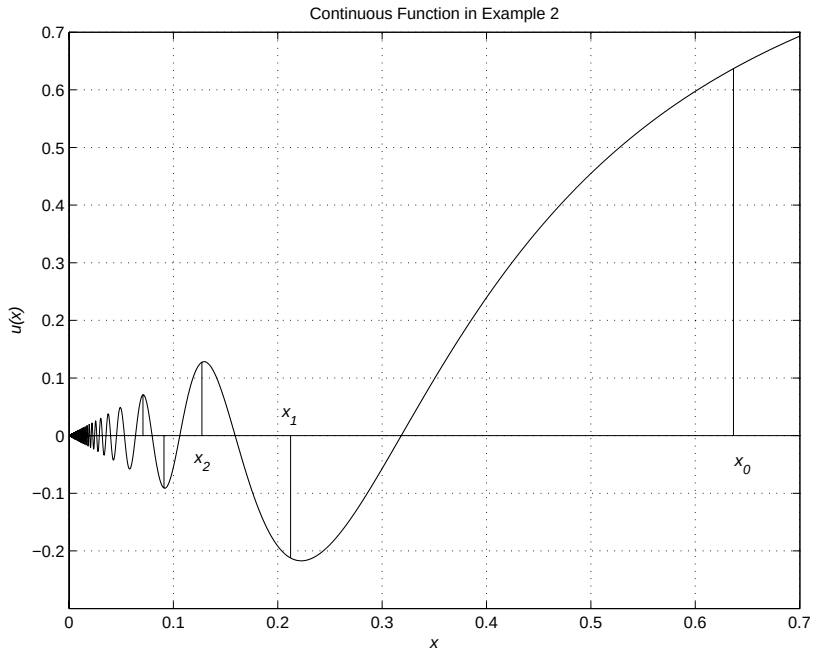
together with points 0 and 1 as the grid points, then the associated variation (see the figure below) can be estimated as

$$V_u(0, x_m, \cdots, x_0, 1) \ge \sum_{i-1}^{m} \left| x_i \sin \frac{1}{x_i} - x_{i-1} \sin \frac{1}{x_{i-1}} \right| \ge \frac{1}{\pi} \sum_{i=1}^{m} \frac{1}{i}$$

Hence

$$\sup_m V_u(0, x_m, \cdots, x_0, 1) \ge \frac{1}{\pi} \sum_{i=1}^{m} \frac{1}{i} \to \infty$$

and $u(x)$ is not BV.



■

Some properties and characterization of BV functions are listed below without proof.

♦ If $u(x)$ is monotonic on $[a, b]$, then $u(x)$ is BV and $\bigvee_a^b(u) = |u(a) - u(b)|$.

♦ If $u(x)$ is BV, then it is a bounded function, i.e., there exists a constant M such that $|u(x)| \le M$ for $x \in [a, b]$.

♦ If $u(x)$ is BV and $\bigvee_a^b(u) = 0$, then $u(x)$ on $[a, b]$ is a constant.

♦ If $u(x)$ and $v(x)$ are BV on $[a, b]$ and α and β are constants, then $\alpha u(x) + \beta v(x)$ is BV.
♦ If $u(x)$ and $v(x)$ are BV on $[a, b]$, then $u(x) \cdot v(x)$ is BV.

♦ If $u(x)$ is BV on $[a, b]$ and c is any real value with $a < c < b$, then

$$\bigvee_a^b(u) = \bigvee_a^c(u) + \bigvee_c^b(u)$$

Conversely, if $u(x)$ is BV on $[a, c]$ and $[c, b]$, then $u(x)$ is also BV on $[a, b]$.

♦ If $u(x)$ is BV on $[a, b]$, then $\bigvee_a^x(u)$ is a monotonically increasing function of x on $[a, b]$.

♦ **Jordan's Decomposition Theorem** If $u(x)$ is defined on $[a, b]$ and is BV, then $u(x)$ can be expressed is the difference of two monotonically increasing functions, e.g.,

$$u(x) = \varphi(x) - \psi(x) \ \textit{with} \ \varphi(x) = \frac{1}{2}\left[\bigvee_a^x(u) + u(x)\right], \psi(x) = \frac{1}{2}\left[\bigvee_a^x(u) - u(x)\right]$$

♦ If $u(x)$ is BV on $[a, b]$, then it has at most a countable infinite number of points on $[a, b]$, where u(x) is discontinuous and all discontinuous points are of first kind; $u(x)$ is Riemann-integralable; $u(x)$ has finite derivative almost everywhere on $[a, b]$; and its derivative $u'(x)$ is Lesbeque-integralable on $[a, b]$.

♦ If $u(x)$ is BV on $[a, b]$, then

$$\frac{d}{dx}\bigvee_a^x(u) = |u'(x)| \qquad (17)$$

holds almost everywhere on $[a, b]$.

2.2 *Bounded Variation Functions of Two Variables*

From (17), we can write $d\overset{x}{\underset{a}{\vee}}(u) = |u'(x)|dx$, hence the total variation (TV) of $u(x)$ can be computed as

$$\overset{b}{\underset{a}{\vee}}(u) = \int_a^b d\overset{x}{\underset{a}{\vee}}(u) = \int_a^b |u'(x)|dx \tag{18}$$

The expression of TV in (18) provides a natural way to extend the concept of BV from the one-variable case to the two-variable case.

Definition A function $u(x, y)$ defined in a compact region R is said to be a function of bounded variation (BV) if it has a finite total variation (TV) which is defined by

$$\vee_R(u) = \iint_R \sqrt{u_x^2 + u_y^2}\, dx\, dy \tag{19}$$

where $u_x = \partial u(x,y)/\partial x, u_y = \partial u(x,y)/\partial y$. Note that the integrand in (19), i.e. $\sqrt{u_x^2 + u_y^2}$ is just the Euclid norm of the gradient of $u(x, y)$, the TV can also be written as

$$\vee_R(u) = \iint_R \left\|\nabla u(x,y)\right\|_2 dx\, dy \tag{20}$$

3. Image De-Noising by TV Minimization

3.1 *Osher's Model for Images*

Given a noisy image which is represented by its light intensity $u_0(x, y)$ over a bounded open set R in the two-dimensional vector space, we want to obtain a decomposition

$$u_0(x, y) = u(x, y) + n(x, y) \tag{21}$$

where $u(x, y)$ represents the true image and $n(x, y)$ is the noise with zero mean and standard deviation σ^2. The de-noising problem at hand is to reconstruct $u(x, y)$ from $u_0(x, y)$. The model in (21) has been known as the *Osher model* [5] since the variational optimization method for image de-noising [2].

In the case of image de-blurring, model (21) is modified to

$$u_0(x, y) = (\mathcal{A}u)(x, y) + n(x, y) \tag{22}$$

where $\mathcal{A}$ is a compact operator on $L^2(R)$ to model the de-blurring process [3].

The main difference between the methods proposed in [2][3] with many well known earlier attempts is that here images are regarded as two-variable functions of bounded variation and the

de-noising and de-blurring problems are treated by minimizing the total variation of the image subject to some variational constraints derived from the model used and the statistics of the noise component. It is this BV framework that allows analysis and processing of piecewise smooth functions with jumps as seen in virtually all natural and synthetic images and is proven proper for many basic image processing tasks.

3.2 *Problem Formulation*

The TV minimization problem for image de-noising can ce formulated as

$$\begin{aligned} &\text{minimize} \quad \vee_R(u) = \iint_R \sqrt{u_x^2 + u_y^2}\, dx\, dy \\ &\text{subject to:} \quad \text{constraints involving } u_0(x,y) \end{aligned} \tag{23}$$

It follows from model (21) that

$$\iint_R u(x,y)\, dx\, dy = \iint_R u_0(x,y)\, dx\, dy \tag{24}$$

and

$$\frac{1}{2}\iint_R [u(x,y) - u_0(x,y)]^2\, dx\, dy = \sigma^2 \tag{25}$$

3.3 *A Solution Procedure*

By defining the *Lagragian* of the problem (23), where the constraints are given in (24) and (25), as the functional

$$L(u, \lambda_1, \lambda_2) = \iint_R \left\{ \sqrt{u_x^2 + u_y^2} + \lambda_1 (u - u_0) + \lambda_2 \left[\frac{1}{2}(u - u_0)^2 - \sigma^2 \right] \right\} dx\, dy \tag{26}$$

and applying the Euler-Lagrange equation (16) to functional (26), it can be concluded that the solution of problem (23) – (26) must satisfy the second-order partial differential equation

$$\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2 + u_y^2}} \right) + \frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2 + u_y^2}} \right) - \lambda_1 - \lambda_2 (u - u_0) = 0 \tag{27a}$$

subject to boundary condition

$$\frac{\partial u(x,y)}{\partial n} = 0 \ \text{ on the boundary } \Gamma = \partial R \tag{27b}$$

where *n* denotes the normal along the boundary of region *R*, which is a reasonable requirement for images defined on a rectangular region and undergone a symmetric extension along the boundary of *R*. Another advantage of imposing (27b) is that the Lagrange multiplier λ_1 can now be dropped from the equation because a solution of

$$\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)-\lambda(u-u_0)=0 \qquad (28a)$$

$$\frac{\partial u(x,y)}{\partial n}=0 \quad \text{on } \Gamma=\partial R \qquad (28b)$$

automatically satisfies $\iint_R (u-u_0)\,dx\,dy=0$, i.e., constraint (24).

In [2], the solution procedure for problem (28) uses a nonlinear *parabolic* equation with time as an evolution parameter. Numerically this is equivalent to the gradient descent method. In doing so, we need to solve

$$\frac{\partial u}{\partial t}=\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)-\lambda(u-u_0) \quad \text{for } t>0,(x,y)\in R \qquad (29a)$$

$$u(x, y, 0) \text{ given} \qquad (29b)$$

$$\frac{\partial u(x,y)}{\partial n}=0 \quad \text{on } \Gamma=\partial R \qquad (29c)$$

$$\frac{1}{2}\iint_R [u(x,y,t)-u_0(x,y)]^2\,dx\,dy\equiv\sigma^2 \qquad (29d)$$

where the initial value in (29b) is typically taken to be the original noisy image $u_0(x, y)$. As time *t* increases, the solution of the (29) converges to its "steady state" i.e. becomes time invariant thus the limiting solution of (29) satisfies (28), a de-noised version of the image.

Notice that the Lagrange multiplier λ in (29a) is *time-dependent* and must be evaluated in order to keep the time-evolution of the solution $u(x, y, t)$ going forward. By multiplying (29a) by $(u - u_0)$, using (29d), integrating by parts over region *R* and using (29c) , and assuming that the steady state has been reached, we obtain a formula for updating the Lagrange multiplier $\lambda(t)$ as

$$\lambda(t)=-\frac{1}{2\sigma^2}\iint_R\left[\sqrt{u_x^2+u_y^2}-\left(\frac{(u_0)_x\cdot u_x}{\sqrt{u_x^2+u_y^2}}+\frac{(u_0)_y\cdot u_y}{\sqrt{u_x^2+u_y^2}}\right)\right]dx\,dy \qquad (30)$$

3.4 *A Difference Scheme for Solving (29), (30)*

Given below is a difference scheme proposed in [2] for computing a numerical solution of problem (28) by solving (30) on an evolving time grids. For the sake of notation simplicity, in the difference scheme a *square* region $R = [0, 1] \times [0, 1]$ has been assumed. The scheme can be applied to non-square images after some straightforward modifications.

$$x_i = ih,\ y_j = jh,\ \text{for } i, j = 0,1,...,N, \text{ with } Nh = 1$$

$$t_n = n\Delta t, \qquad \text{for } n = 0,1,.....$$

$$u_{i,j}^n = u(x_i, y_j, t_n)$$

$$u_{i,j}^0 = u_0(x_i, y_j)$$

For $i, j = 0,1,..., N-1$, do

$$u_{i,j}^{n+1} = u_{i,j}^n$$

$$+\frac{\Delta t}{h}\left[\Delta_-^x\left(\frac{\Delta_+^x u_{i,j}^n}{\left[(\Delta_+^x u_{i,j}^n)^2 + (m(\Delta_+^y u_{i,j}^n, \Delta_-^y u_{i,j}^n))^2\right]^{1/2}}\right)\right]$$

$$+\frac{\Delta t}{h}\left[\Delta_-^y\left(\frac{\Delta_+^y u_{i,j}^n}{\left[(\Delta_+^y u_{i,j}^n)^2 + (m(\Delta_+^x u_{i,j}^n, \Delta_-^x u_{i,j}^n))^2\right]^{1/2}}\right)\right]$$

$$-\Delta t \cdot \lambda^n \left(u_{i,j}^n - u_0(ih, jh)\right)$$

with

$$u_{0,j}^n = u_{1,j}^n, u_{N,j}^n = u_{N-1,j}^n, u_{i,0}^n = u_{i,1}^n, \text{ and } u_{i,N}^n = u_{i,N-1}^n.$$

$$\Delta_\mp^x u_{i,j} = \mp(u_{i\mp1,j} - u_{i,j})$$

$$\Delta_\mp^y u_{i,j} = \mp(u_{i,j\mp1} - u_{i,j})$$

$$m(a,b) = \min \text{mod}(a,b) = \left(\frac{\text{sgn}\, a + \text{sgn}\, b}{2}\right)\min(|\, a\,|, |\, b\,|)$$

where λ^n is computed based on (30) as

$$\lambda^n = -\frac{h}{2\sigma^2}\left[\sum_i\sum_j \sqrt{\left(\Delta_+^x u_{i,j}^n\right)^2 + \left(\Delta_+^y u_{i,j}^n\right)^2} - \frac{(\Delta_+^x u_{i,j}^0)(\Delta_+^x u_{i,j}^n)}{\sqrt{\left(\Delta_+^x u_{i,j}^n\right)^2 + \left(\Delta_+^y u_{i,j}^n\right)^2}} - \frac{\left(\Delta_+^y u_{i,j}^0\right)\left(\Delta_+^y u_{i,j}^n\right)}{\sqrt{\left(\Delta_+^x u_{i,j}^n\right)^2 + \left(\Delta_+^y u_{i,j}^n\right)^2}}\right]$$

A step size restriction is imposed for numerical stability of the scheme:

$$\frac{\Delta t}{h^2} \le c \quad \text{for some constant } c \qquad (31)$$

3.5 *MATLAB Code and Results*

Listed below are several MATLAB functions written by Guy Gilboa of UCLA that can be used to demonstrate the Rudin-Osher-Fatemi algorithm [2]. The main routine named demo_tv.m is given in the first m-file which calls for the second and third MATLAB routines.

1. File Name: demo_tv.m

```
%%% TV demo %%
%%% By Guy Gilboa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Based on: [ROF92] L. Rudin, S. Osher, E. Fatemi,
% "Nonlinear Total Variation based noise removal algorithms",
% Physica D 60 259-268,1992.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Simple TV denoising  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
I = imread('camera256.tif'); % load image
I = double(I(11:138,11:138)); % cut a piece, convert to double

% params
iter = 80;
% dt=0.2; eps=1;

%%% Add noise
std_n = 20; % Gaussian noise standard deviation
In = randn(size(I))*std_n; % White Gaussian noise
I0 = I + In;  % noisy input image
% show original and noisy images
close all
figure(1); imshow(uint8(I)); title('Original')
figure(2); imshow(uint8(I0)); title('Noisy image')
% denoise image by using tv for some iterations
J = tv(I0,iter);
figure(3); imshow(uint8(J)); title('Denoised image')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  TV denoising with (scalar) data fidelity term  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Here we assume the noise vaiance is known.
%% Therefore we can compute lambda (instead of fixing
%% iter). TV therfore can run automatically quite well.
%% The process is slower.
J = I0;
% params
ep_J = 0.01; % minimum mean change in image J
lam = 0; J_old = 0;
iter = 10; dt = 0.2; eps = 1;
```

```
var_n = std_n^2; % noise variance
i = 0;
while (mean(mean(abs(J - J_old))) > ep_J),  % iterate until convergence
  J_old = J;
  J = tv(J,iter,dt,eps,lam,I0);     % scalar lam
  lam = calc_lam(J,I0,var_n,eps); % update lambda (fidelity term)
end % for i
figure(4); imshow(uint8(J)); title('Denoised image with lambda')
```

2. File Name: tv.m

```
function J = tv(I,iter,dt,ep,lam,I0,C)

%% Private function: tv (by Guy Gilboa).
%% Total Variation denoising.
%% Example: J = tv(I,iter,dt,ep,lam,I0)
%% Input: I    - image (double array gray level 1-256),
%%        iter - num of iterations,
%%        dt   - time step [0.2],
%%        ep   - epsilon (of gradient regularization) [1],
%%        lam  - fidelity term lambda [0],
%%        I0   - input (noisy) image [I0=I]
%%       (default values are in [])
%% Output: evolved image

if ~exist('ep')
  ep = 1;
end
if ~exist('dt')
  dt = ep/5;  % dt below the CFL bound
end
if ~exist('lam')
  lam = 0;
end
if ~exist('I0')
        I0 = I;
end
if ~exist('C')
        C = 0;
end
[ny,nx] = size(I);
ep2 = ep^2;

for i=1:iter,  %% do iterations
  % estimate derivatives
   I_x = (I(:,[2:nx nx])-I(:,[1 1:nx-1]))/2;
   I_y = (I([2:ny ny],:)-I([1 1:ny-1],:))/2;
   I_xx = I(:,[2:nx nx])+I(:,[1 1:nx-1])-2*I;
   I_yy = I([2:ny ny],:)+I([1 1:ny-1],:)-2*I;
   Dp = I([2:ny ny],[2:nx nx])+I([1 1:ny-1],[1 1:nx-1]);
        Dm = I([1 1:ny-1],[2:nx nx])+I([2:ny ny],[1 1:nx-1]);
```

```
        I_xy = (Dp-Dm)/4;
  % compute flow
    Num = I_xx.*(ep2+I_y.^2)-2*I_x.*I_y.*I_xy+I_yy.*(ep2+I_x.^2);
    Den = (ep2+I_x.^2+I_y.^2).^(3/2);
    I_t = Num./Den + lam.*(I0-I+C);
    I = I+dt*I_t;  %% evolve image by dt
end % for i
%% return image
%J = I*Imean/mean(mean(I)); % normalize to original mean
J = I;
```

3. File Name: calc_lam.m

```
function lam = calc_lam(I,I0,var_n,ep)

%% private function: calc_lam (by Guy Gilboa).
%% calculate scalar fidelity term of the Total-Variation
%% input: current image, original noisy image, noise variance, epsilon
%% output: lambda
%% example: lam=calc_lam(I,I0,var_n,ep)

if ~exist('ep') % good for 256 gray-level
   ep = 1;
end

[ny,nx] = size(I); ep2 = ep^2;

% estimate derivatives
  I_x = (I(:,[2:nx nx])-I(:,[1 1:nx-1]))/2;
  I_y = (I([2:ny ny],:)-I([1 1:ny-1],:))/2;
  I_xx = I(:,[2:nx nx])+I(:,[1 1:nx-1])-2*I;
  I_yy = I([2:ny ny],:)+I([1 1:ny-1],:)-2*I;
          Dp = I([2:ny ny],[2:nx nx])+I([1 1:ny-1],[1 1:nx-1]);
          Dm = I([1 1:ny-1],[2:nx nx])+I([2:ny ny],[1 1:nx-1]);
          I_xy = (Dp-Dm)/4;
   % compute numerator and denomenator
   Num = I_xx.*(ep2+I_y.^2)-2*I_x.*I_y.*I_xy+I_yy.*(ep2+I_x.^2);
   Den = (ep2+I_x.^2+I_y.^2).^(3/2);
   Div = Num./Den;
   %%% fidelity term
lam = mean(mean(Div.*(I-I0)))./var_n;
```

The above MATLAB code was tested by applying it to a small part of a test image named cameraman. The image has a size of 128 by 128 and was taken from the tif file camera256(11:138,11:138) and was contaminated by a Gaissian white noise with zero mean and standard deviation = 20.

The results in term of the original, noisy, and de-noised images are depicted in the following plots. The peak-signal-to-noise ratio (PSNR) was improved from 22.0647 dB to 29.5268 dB.

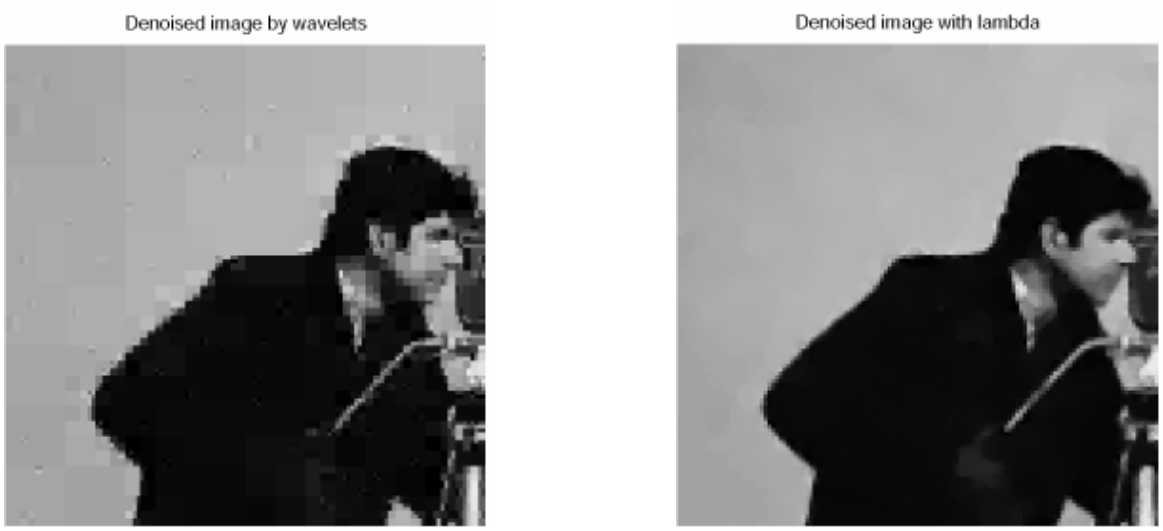
Original

Noisy image

Denoised image with lambda

For comparison purposes, Donoho's well-known wavelet de-noising algorithm using Daubechies wavelets of length 2, 4, and 6 were also applied to the same test image. The best result was achieved by the D2 (Haar) wavelet that yielded PSNR = 27.0670dB.

The plot below displays the de-noised image by wavelet (on the left) and de-noised image by TV (on the right) minimization. It is observed that the TV-minimization based algorithm yields improved de-noising results in terms of PSNR as well as visual examination.



4. Image De-Blurring by TV Minimization

In [3], the image de-blurring problem is investigated in a similar variational optimization framework. Here the image model is given by (22), i.e.,

$$u_0(x, y) = (\mathcal{A}u)(x, y) + n(x, y) \quad \text{for } (x, y) \text{ in } R \tag{32}$$

where $\mathcal{A}$ is typically a convolutional type integral operator that can be used for modeling several common blurring processes such as averaging, Gaussian low-pass, Laplacian of Gaussian, and motion blurs.

The variational optimization problem in this case can be formulated as

$$\begin{aligned} &\text{minimize} \quad \vee_R(u) = \iint_R \sqrt{u_x^2 + u_y^2}\, dx\, dy \\ &\text{subject to:} \quad \iint_R u(x, y)\, dx\, dy = \iint_R u_0(x, y)\, dx\, dy \\ &\qquad\qquad \iint_R [Au(x, y) - u_0(x, y)]^2\, dx\, dy = \sigma^2 \end{aligned} \tag{33}$$

The Euler-Lagrange equation corresponding to problem (33) is given by

$$\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)-\lambda A^*(Au-u_0)=0 \qquad (34a)$$

$$\frac{\partial u(x,y)}{\partial n}=0 \quad \text{on } \Gamma=\partial R \qquad (34b)$$

where $\mathcal{A}^*$ denotes the *adjoint operator* for operator $\mathcal{A}$. In [3], it is proposed that (34) be solved by calculating the steady-state solution of the parabolic partial differential equation

$$\frac{\partial u}{\partial t}=\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)-\lambda A^*(Au-u_0) \quad \text{for } t>0,(x,y)\in R \qquad (35a)$$

$$\frac{\partial u(x,y)}{\partial n}=0 \quad \text{on } \Gamma=\partial R \qquad (35b)$$

where, using a technique similar to that for the de-noising case, the time-varying Lagrange multiplier $\lambda=\lambda(t)$ is found to be

$$\lambda(t)=\frac{\iint_R\left[\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)\right]\cdot A^*\left(Au-u_0\right)dx\,dy}{\iint_R\left[A^*\left(Au-u_0\right)\right]^2 dxdy} \qquad (36)$$