Proximal Gradient Algorithms: Applications in Signal Processing

Niccoló Antonello, Lorenzo Stella, Panos Patrinos, Toon van Waterschoot

niccolo.antonello@idiap.ch, stellalo@amazon.com, panos.patrinos@esat.kuleuven.be, toon.vanwaterschoot@esat.kuleuven.be

EUSIPCO 2019

IDIAP, Amazon Research, KU Leuven ESAT-STADIUS
Proximal Gradient Algorithms: Applications in Signal Processing
Part I: Introduction

Toon van Waterschoot
toon.vanwaterschoot@esat.kuleuven.be

EUSIPCO 2019

KU Leuven, ESAT-STADIUS
Inference problems such as ... 

- Signal estimation,
- Parameter estimation,
- Signal detection,
- Data classification

... naturally lead to optimization problems

\[ x_\star = \arg \min_x \text{cost}(x) + \text{prior}(x) \]

Variables \( x \) could be signal samples, model parameters, algorithm tuning parameters, etc.
Modeling and Inverse Problems

Many inference problems lead to optimization problems in which signal models need to be inverted, i.e. inverse problems:

Given set of observations $y$, infer unknown signal or model parameters $x$

- Inverse problems are often ill-conditioned or underdetermined
- Large-scale problems may suffer more easily from ill-conditioning
- Including prior in cost function then becomes crucial (e.g. regularization)
Choice of suitable cost function often depends on adoption of application-specific **signal model**, e.g.

- Dictionary model, e.g. sum of sinusoids $y = Dx$ with DFT matrix $D$
- Filter model, e.g. linear FIR filter $y = Hx$ with convolution matrix $H$
- Black-box model, e.g. neural network $y = f(x)$ with feature transformation function $f$

In this tutorial, we will often represent signal models as **operators**, i.e.

\[
\begin{align*}
  y &= Ax \text{ with } A = \text{linear operator} \\
  y &= A(x) \text{ with } A = \text{nonlinear operator}
\end{align*}
\]
Motivating Examples

Example 1: **Line spectral estimation**

- **DFT dictionary model with selection matrix** $S$ and inverse DFT matrix $F_i$

  $$y = SF_i x$$

- Underdetermined inverse problem: $\dim(y) \ll \dim(x)$

- Spectral sparsity prior for line spectrum

\[
x_\star = \text{argmin}_x \left( \text{DFT model output error}(x) + \text{spectral sparsity}(x) \right)
\]

![Graphical representation of line spectral estimation](image-url)
Motivating Examples

Example 2: **Video background removal**

- Static background + dynamic foreground decomposition model
  \[ Y = L + S \]

- Underdetermined inverse problem: \( \text{dim}(Y) = \frac{1}{2} (\text{dim}(L) + \text{dim}(S)) \)

- Rank-1 prior for static BG + sparse prior for FG changes (robust PCA)

\[ x^* = \underset{x}{\text{argmin}} \left( \text{BG + FG model output error}(x) + \text{BG rank} + \text{FG sparsity}(x) \right) \]
Motivating Examples

Example 3: **Audio de-clipping**

- DCT dictionary model with inverse DCT matrix $F_{i,c}$
  
  $$y = F_{i,c}x$$

- Underdetermined inverse problem: missing data (clipped samples) in $y$

- Spectral sparsity for audio signal + amplitude prior for clipped samples

\[
x_\star = \underset{x}{\text{argmin}} \left( \text{DCT model output error}(x) + \text{spectral sparsity} + \text{clipping}(x) \right)
\]

\[
\text{loss}(x) + \text{prior}(x)
\]
Challenges in Optimization

Linear vs. nonlinear optimization
- Linear: closed-form solution
- Nonlinear: iterative numerical optimization algorithms

Convex vs. nonconvex optimization
- Convex: unique optimal point (global minimum)
- Nonconvex: multiple optimal points (local minima)

Smooth vs. non-smooth optimization
- Smooth: Newton-type methods using first- and second-order derivatives
- Non-smooth: first-order methods using (sub)gradients
Challenges in Optimization

Trends and observations:
- Loss is often linear/convex/smooth but prior is often not
- Even if non-convex problems are hard to solve globally, iterating from good initialization may yield local minimum close enough to global minimum
- Non-smooth problems are typically tackled with first-order methods, showing slower convergence than Newton-type methods

Key message of this tutorial:

Also for non-smooth optimization problems, Newton-type methods showing fast convergence can be derived

- This greatly broadens variety of loss functions and priors that can be used
- Theory, software implementation, and signal processing examples will be presented in next 2.5h
1. Introduction

2. Proximal Gradient (PG) algorithms
   - Proximal mappings and proximal gradient method
   - Dual and accelerated proximal gradient methods
   - Newton-type proximal gradient algorithms

3. Software Toolbox
   - Short introduction to Julia language
   - Structured Optimization package ecosystem

4. Demos and Examples
   - Line spectral estimation
   - Video background removal
   - Audio de-clipping

5. Conclusion
Proximal Gradient Algorithms: Applications in Signal Processing

Part II

Lorenzo Stella
stellalo@amazon.com

EUSIPCO 2019

AWS AI Labs (Amazon Research)
About me

- Applied Scientist at AWS AI Labs (Amazon Research)
- Deep learning, probabilistic time series models
- Time series forecasting, classification, anomaly detection...
- **We’re hiring!**

**Gluon Time Series:** github.com/awslabs/gluon-ts

- Previously: Ph.D. at IMT Lucca and KU Leuven with Panos Patrinos
- The work presented here was done prior to joining Amazon
Outline

1. Preliminary concepts, composite optimization, proximal mappings

2. Proximal gradient method

3. Duality

4. Accelerated proximal gradient

5. Newton-type proximal gradient methods

6. Concluding remarks
Blanket assumptions

In this presentation:

- Underlying space is the Euclidean space $\mathbb{R}^n$ equipped with
  - Inner product $\langle \cdot, \cdot \rangle$, e.g. dot product
  - Induced norm $\| \cdot \| = \sqrt{\langle \cdot, \cdot \rangle}$

- Linear mappings will be identified by their matrices and adjoints will be denoted by transpose $^\top$

- Most algorithms will be matrix-free: can view matrices and their transposes are linear mappings and their adjoints

- All results carry over to general Euclidean spaces, most of them even to Hilbert spaces
The space $\mathbb{R}^n$

- $n$-dimensional column vectors with real components endowed with

$$
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n
\end{bmatrix}
+ 
\begin{bmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_n
\end{bmatrix}
= 
\begin{bmatrix}
  x_1 + y_1 \\
  x_2 + y_2 \\
  \vdots \\
  x_n + y_n
\end{bmatrix},
\quad 
\alpha 
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n
\end{bmatrix}
= 
\begin{bmatrix}
  \alpha x_1 \\
  \alpha x_2 \\
  \vdots \\
  \alpha x_n
\end{bmatrix}
$$

- Standard inner product: $\langle x, y \rangle = x^\top y = \sum_{i=1}^{n} x_i y_i$
- Induced norm: $\|x\| = \|x\|_2 = \sqrt{\langle x, x \rangle} = \sqrt{\sum_{i=1}^{n} x_i^2}$

Alternative inner product and induced norm ($Q \succ 0$ is $n \times n$)

$$
\langle x, y \rangle = \langle x, y \rangle_Q = x^\top Q y
\quad 
\|x\| = \|x\|_Q = \sqrt{x^\top Q x}
$$
The space $\mathbb{R}^{m \times n}$

- $m \times n$ real matrices

$$X = \begin{bmatrix}
  x_{11} & x_{12} & \cdots & x_{1n} \\
  x_{21} & x_{22} & \cdots & x_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{m1} & x_{m2} & \cdots & x_{mn}
\end{bmatrix}$$

- Standard inner product

$$\langle X, Y \rangle = \text{trace}(X^TY) = \sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} Y_{ij}$$

- Induced norm

$$\|X\| = \|X\|_F = \sqrt{\langle X, X \rangle} = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} X_{ij}^2}$$

Frobenius norm
Extended-real-valued functions

- Extended real line
  \[ \overline{\mathbb{R}} = \mathbb{R} \cup \{\infty\} = (-\infty, \infty] \]

- Extended-real-valued functions
  \[ f : \mathbb{R}^n \to \overline{\mathbb{R}} \]

- Effective domain
  \[ \text{dom} \ f = \{x \in \mathbb{R}^n \mid f(x) < \infty\} \]

- \( f \) is called proper if \( f(x) < \infty \) for some \( x \) (\( \text{dom} \ f \) is nonempty)

- Offer a unified view of optimization problems

**Main example**: indicator of set \( C \subseteq \mathbb{R}^n \)

\[ \delta_C(x) = \begin{cases} 0 & \text{if } x \in C \\ +\infty & \text{otherwise} \end{cases} \]
**Epigraph**

**Epigraph:** \( \text{epi} \, f = \{(x, \alpha) \in \mathbb{R}^n \times \mathbb{R} \mid f(x) \leq \alpha\} \)

- \( f \) is **closed** iff \( \text{epi} \, f \) is a closed set.
- \( f \) is **convex** iff \( \text{epi} \, f \) is a convex set.
Subdifferential of a proper, convex function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$:

$$\partial f(x) = \{ v | f(y) \geq f(x) + \langle v, y - x \rangle \ \forall y \in \mathbb{R}^n \}$$

- $\partial f(x)$ is a convex set
- $\partial f(x) = \{ v \}$ iff $f$ is differentiable at $x$ with $\nabla f(x) = v$
- $\bar{x}$ minimizes $f$ iff $0 \in \partial f(\bar{x})$
- Definition above can be extended to nonconvex $f$
Composite optimization problems

\[
\text{minimize } \varphi(x) = f(x) + g(x)
\]

Assumptions

- \( f : \mathbb{R}^n \to \mathbb{R} \) differentiable with \( L \)-Lipschitz gradient (\( L \)-smooth)

\[
\| \nabla f(x) - \nabla f(y) \| \leq L \| x - y \| \quad \forall x, y \in \mathbb{R}^n
\]

- \( g : \mathbb{R}^n \to \overline{\mathbb{R}} \) proper, closed

- Set of optimal solutions \( \text{argmin } f + g \) is nonempty
Proximal mapping (or operator)

Assume $g : \mathbb{R}^n \rightarrow \overline{\mathbb{R}}$ closed, proper

$$\text{prox}_{\gamma g}(x) = \arg\min_{z \in \mathbb{R}^n} \left\{ g(z) + \frac{1}{2\gamma} \| z - x \|^2 \right\}, \quad \gamma > 0$$

If $g$ is convex:

- for all $x \in \mathbb{R}^n$, function $z \mapsto g(z) + \frac{1}{2\gamma} \| z - x \|^2$ is strongly convex
- $\text{prox}_{\gamma g}(x)$ is unique for all $x \in \mathbb{R}^n$, i.e., $\text{prox}_{\gamma g} : \mathbb{R}^n \rightarrow \mathbb{R}^n$

Examples

- $f(x) = 0$: $\text{prox}_{\gamma f}(x) = x$
- $f(x) = \delta_C(x)$: $\text{prox}_{\gamma f}(x) = \Pi_C(x)$

Proximal mapping: generalization of Euclidean projection
Proximal mapping (or operator)

Assume $g : \mathbb{R}^n \to \mathbb{R}$ closed, proper

$$\text{prox}_{\gamma g}(x) = \arg\min_{z \in \mathbb{R}^n} \left\{ g(z) + \frac{1}{2\gamma} \| z - x \|^2 \right\}, \quad \gamma > 0$$

If $g$ is convex:
- for all $x \in \mathbb{R}^n$, function $z \mapsto g(z) + \frac{1}{2\gamma} \| z - x \|^2$ is strongly convex
- $\text{prox}_{\gamma g}(x)$ is unique for all $x \in \mathbb{R}^n$, i.e., $\text{prox}_{\gamma g} : \mathbb{R}^n \to \mathbb{R}^n$

Examples
- $f(x) = 0$: $\text{prox}_{\gamma f}(x) = x$
- $f(x) = \delta_C(x)$: $\text{prox}_{\gamma f}(x) = \Pi_C(x)$

Proximal mapping: generalization of Euclidean projection
Properties

$$\prox_{\gamma g}(x) = \arg\min_{z \in \mathbb{R}^n} \left\{ g(z) + \frac{1}{2\gamma} \|z - x\|^2 \right\}, \quad \gamma > 0$$

- If $g$ is convex, from the optimality conditions:

$$p \in \arg\min_z g(z) + \frac{1}{2\gamma} \|z - x\|^2 \iff -\gamma^{-1}(p - x) \in \partial g(p)$$

$$\iff x \in p + \gamma \partial g(p)$$

- In other words

$$p \in x - \gamma \partial g(p) \quad (\spadesuit)$$

- Equivalent to \textit{implicit subgradient} step
- Analogous to implicit Euler method for ODEs
- From (\spadesuit), any fixed-point $\bar{x} = \prox_{\gamma g}(\bar{x})$ satisfies $0 \in \partial g(\bar{x})$

\textbf{Fixed-points of} $\prox_{\gamma g} \equiv \text{minimizers of } g$
\[ \text{prox}_{\gamma g}(x) = \arg\min_{z \in \mathbb{R}^n} \left\{ g(z) + \frac{1}{2\gamma} \|z - x\|^2 \right\}, \quad \gamma > 0 \]

- For convex \( g \), mapping \( \text{prox}_{\gamma g} : \mathbb{R}^n \to \mathbb{R}^n \) is **firmly nonexpansive (FNE)**

\[ \| \text{prox}_{\gamma g}(x) - \text{prox}_{\gamma g}(y) \|^2 \leq \langle \text{prox}_{\gamma g}(x) - \text{prox}_{\gamma g}(y), x - y \rangle \quad \forall x, y \in \mathbb{R}^n \]
Properties

\[ \text{prox}_{\gamma g}(x) = \arg\min_{z \in \mathbb{R}^n} \left\{ g(z) + \frac{1}{2\gamma} \| z - x \|^2 \right\}, \quad \gamma > 0 \]

- For convex \( g \), mapping \( \text{prox}_{\gamma g} : \mathbb{R}^n \rightarrow \mathbb{R}^n \) is **firmly nonexpansive (FNE)**

\[ \| \text{prox}_{\gamma g}(x) - \text{prox}_{\gamma g}(y) \|^2 \leq \langle \text{prox}_{\gamma g}(x) - \text{prox}_{\gamma g}(y), x - y \rangle \quad \forall x, y \in \mathbb{R}^n \]
Properties

\[
\text{prox}_g(x) = \arg\min_{z \in \mathbb{R}^n} \left\{ g(z) + \frac{1}{2\gamma} \|z - x\|^2 \right\}, \quad \gamma > 0
\]

- For convex \( g \), mapping \( \text{prox}_g : \mathbb{R}^n \to \mathbb{R}^n \) is firmly nonexpansive (FNE)

\[
\| \text{prox}_g(x) - \text{prox}_g(y) \|^2 \leq \langle \text{prox}_g(x) - \text{prox}_g(y), x - y \rangle \quad \forall x, y \in \mathbb{R}^n
\]

- FNE implies \( \text{prox}_g \) nonexpansive (Cauchy-Schwarz)

\[
\| \text{prox}_g(x) - \text{prox}_g(y) \| \leq \|x - y\| \quad \forall x, y \in \mathbb{R}^n
\]
Examples of proximal mappings

- Convex quadratic function
  \[ g(x) = \frac{1}{2} \langle x, Qx \rangle + \langle q, x \rangle \quad \text{prox}_{\gamma g}(x) = (I + \gamma Q)^{-1}(x - \gamma q) \]

- Euclidean norm
  \[ g(x) = \|x\| \quad \text{prox}_{\gamma g}(x) = \begin{cases} (1 - \gamma/\|x\|)x, & \|x\| > \gamma, \\ 0, & \text{otherwise} \end{cases} \]

- $L_1$-norm
  \[ g(x) = \|x\|_1 = \sum_i |x_i| \quad [\text{prox}_{\gamma g}(x)]_i = \begin{cases} x_i + \gamma, & x_i < -\gamma \\ 0, & |x_i| \leq \gamma \\ x_i - \gamma, & x_i > \gamma \end{cases} \]

- Nuclear norm
  \[ g(X) = \sum \text{diag } \Sigma \quad \text{prox}_{\gamma g}(X) = U\hat{\Sigma}V^T \]
  where \( X = U\Sigma V^T \)
  where \( \text{diag } \hat{\Sigma} = \text{prox}_{\gamma \| \cdot \|_1}(\text{diag } \Sigma) \)
Proximal calculus rules

- **Separable sum:** \( f(x_1, x_2) = f_1(x_1) + f_2(x_2) \)

  \[
  \text{prox}_{\gamma f}(x_1, x_2) = (\text{prox}_{\gamma f_1}(x_1), \text{prox}_{\gamma f_2}(x_2))
  \]

- **Scaling and translation:** \( f(x) = \phi(\alpha x + \beta), \alpha \neq 0 \)

  \[
  \text{prox}_{\gamma f}(x) = \frac{1}{\alpha} (\text{prox}_{\alpha^2 \lambda \phi}(\alpha x + \beta) - \beta)
  \]

- **Postcomposition:** \( f(x) = \alpha \phi(x) + \beta, \alpha > 0 \)

  \[
  \text{prox}_{\gamma f}(x) = \text{prox}_{\alpha \gamma \phi}(x)
  \]

- **Orthogonal composition:** \( f(x) = \phi(Qx), \ Q^\top Q = QQ^\top = I \)

  \[
  \text{prox}_{\gamma f}(x) = Q^\top \text{prox}_{\gamma \phi}(Qx)
  \]

  (e.g.: \( Q = \text{DCT}, \text{DFT} \))
Properties

\[ \text{prox}_{\gamma g}(x) = \arg\min_{z \in \mathbb{R}^n} \left\{ g(z) + \frac{1}{2\gamma} \|z - x\|^2 \right\}, \quad \gamma > 0 \]

- If \( g \) is convex \( \text{prox}_{\gamma g} \) is single-valued
- If \( g \) is nonconvex \( \text{prox}_{\gamma g} \) is set-valued in general
  - Can be empty, can be multi-valued
  - If \( g \) is lower bounded then \( \text{prox}_{\gamma g}(x) \) nonempty for all \( x \)
  - Algorithms will work by taking any \( p \in \text{prox}_{\gamma g}(x) \)
Outline

1. Preliminary concepts, composite optimization, proximal mappings

2. Proximal gradient method

3. Duality

4. Accelerated proximal gradient

5. Newton-type proximal gradient methods

6. Concluding remarks
Composite optimality conditions

\[
\text{minimize } \varphi(x) = f(x) + g(x)
\]

- If \( x^* \) is a local minimum of \( \varphi \) then
  \[
  -\nabla f(x^*) \in \partial g(x^*)
  \]
  (1)
- Moreover, we have shown already that for any \( x \in \mathbb{R}^n \)
  \[
  p \in \text{prox}_{\gamma g}(x) \iff x \in p + \gamma \partial g(p)
  \]
  (2)
- We can reformulate (1) as follows, using (2)
  \[
  -\nabla f(x^*) \in \partial g(x^*) \iff x^* - \gamma \nabla f(x^*) \in x^* + \gamma \partial g(x^*)
  \leftarrow\rightarrow x^* = \text{prox}_{\gamma g}(x^* - \gamma \nabla f(x^*))
  \]
- We have shown that \( x^* \) satisfies (1) iff it is a fixed point of mapping
  \[
  T(x) = \text{prox}_{\gamma g}(x - \gamma \nabla f(x))
  \]
Proximal gradient method

To minimize $f + g$ iterate

$$x^{k+1} = \text{prox}_{\gamma g}(x^{k} - \gamma \nabla f(x^{k}))$$

• Reduces to gradient method if $g = 0$

$$x^{k+1} = x^{k} - \gamma \nabla f(x^{k})$$

• Reduces to gradient projection when $g = \delta_{C}$

$$x^{k+1} = \Pi_{C}(x^{k} - \gamma \nabla f(x^{k}))$$

• Reduces to proximal point method when $f = 0$

$$x^{k+1} = \text{prox}_{\gamma g}(x^{k})$$
Interpretations

\[ x^{k+1} = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]

- Proximal gradient step can be expressed as linearized (in \( f \)) sub-problem

\[ x^{k+1} = \arg\min_u \left\{ f(x^k) + \langle \nabla f(x^k), u - x^k \rangle + g(u) + \frac{1}{2\gamma} \| u - x^k \|^2 \right\}_{l_f(u; x^k)} \]

- Since \( \nabla f \) is Lipschitz, for \( \gamma \leq 1/L \):

\[ f(u) \leq l_f(u; x^k) + \frac{1}{2\gamma} \| u - x^k \|^2 \quad \text{for all } u \in \mathbb{R}^n \]

- Thus \( l_f(u; x^k) + g(u) + \frac{1}{2\gamma} \| u - x^k \|^2 \) majorizes \( \varphi(u) \)

- Proximal gradient as a majorization minimization algorithm
$$x^{k+1} = \arg\min_u \{ f(x^k) + \langle \nabla f(x^k), u - x^k \rangle + g(u) + \frac{1}{2\gamma} \| u - x^k \|^2 \}$$

\[
\phi = f + g
\]
\[ x^{k+1} = \arg\min_u \left\{ f(x^k) + \langle \nabla f(x^k), u - x^k \rangle + g(u) + \frac{1}{2\gamma} \| u - x^k \|^2 \right\} \]

\( l_f(u; x^k) \)

\[ \varphi(x^0) \]

\[ \varphi = f + g \]
\[ x^{k+1} = \operatorname{argmin}_u \{ f(x^k) + \langle \nabla f(x^k), u - x^k \rangle + g(u) + \frac{1}{2\gamma} \| u - x^k \|^2 \} \]

\[ \ell_f(u; x^k) \]

\[ \varphi(x^0) \]

\[ \varphi = f + g \quad \ell_f(u; x^0) + g(u) + \frac{1}{2\gamma} \| u - x^0 \|^2 \]
\[
x^{k+1} = \underset{u}{\text{argmin}} \left\{ f(x^k) + \langle \nabla f(x^k), u - x^k \rangle + g(u) + \frac{1}{2\gamma} \| u - x^k \|^2 \right\}
\]

\[
\ell_f(u; x^k)
\]

\[
\varphi(x^0) \quad \varphi(x^1)
\]

\[
\varphi = f + g \quad \ell_f(u; x^0) + g(u) + \frac{1}{2\gamma} \| u - x^0 \|^2
\]
$x^{k+1} = \arg\min_u \left\{ f(x^k) + \langle \nabla f(x^k), u - x^k \rangle + g(u) + \frac{1}{2\gamma} \| u - x^k \|^2 \right\}$

\[ \ell_f(u; x^k) \]

\[ \varphi(x^0) \]

\[ \varphi(x^1) \]

\[ \varphi(x^2) \]

\[ x^0 \]

\[ x^1 \]

\[ x^2 \]

\[ \varphi = f + g \]

\[ \ell_f(u; x^1) + g(u) + \frac{1}{2\gamma} \| u - x^1 \|^2 \]
Interpretations

$$x^{k+1} = \arg\min_u \left\{ f(x^k) + \langle \nabla f(x^k), u - x^k \rangle + g(u) + \frac{1}{2\gamma} \| u - x^k \|^2 \right\}$$

\[ \ell_f(u; x^k) \]
Convergence rate (convex case)

Theorem (Convergence rate – convex case)

The iterates of proximal gradient method with $\gamma \in (0, 1/L]$ satisfy

$$\varphi(x^k) - \varphi(x_*) \leq \frac{\|x_0 - x_*\|^2}{2\gamma k}$$

• **Conclusion**: to reach $\varphi(x^k) - \varphi(x_*) \leq \epsilon$, proximal gradient needs

$$k = \lceil \frac{\|x_0 - x_*\|^2}{2\gamma \epsilon} \rceil \quad \text{iterations}$$
Convergence rate (strongly convex case)

\[ \|x^{k+1} - x_*\|^2 \leq (1 - \gamma \mu) \|x^k - x_*\|^2 \]  

1. if \( f \) strongly convex (\( \mu > 0 \)), then linear convergence

\[ \|x^k - x_*\|^2 \leq c^k \|x^0 - x_*\|^2 \quad c = 1 - \gamma \mu \]

2. for \( \gamma = \frac{1}{L} \) contraction factor is \( c = 1 - \frac{\mu}{L} \)
3. for small \( \frac{\mu}{L} \) convergence is slow
4. \( \mu = 0 \): (♠) shows that distance from solution set is nonincreasing

\[ \|x^{k+1} - x_*\| \leq \|x^k - x_*\| \]

5. sequences with this property are called Fejér monotone
6. Fejér monotonicity: convergence of the sequence of iterates to some \( x_* \)
Convergence (nonconvex case)

- If \( f \) is **nonconvex** and \( \gamma \leq 1/L \)
  \[
  \lim_{k \to \infty} \| R(x_k) \| = 0 \quad \text{\( R(x) = x - \text{prox}_{\gamma g}(x - \gamma \nabla f(x)) \)}
  \]
- \( R : \mathbb{R}^n \to \mathbb{R}^n \) is sufficiently regular, e.g. it’s Lipschitz if \( g \) is convex
- This implies that every cluster point \( \bar{x} \) of \( (x_k)_{k \in \mathbb{N}} \) satisfies
  \[
  R(\bar{x}) = 0 \iff -\nabla f(\bar{x}) \in \partial g(\bar{x})
  \]
- Convergence of the sequence using **Kurdyka-Lojasiewicz** assumption
Proximal gradient with line search

- In practice Lipschitz constant $L$ is not known, how to select $\gamma$?
- Can do backtracking: start with $\gamma_0$ large and at every iteration run

**Algorithm 1:** Line search to determine $\gamma$

**Input:** $x^k$, $\gamma_{k-1}$ and $\beta \in (0, 1)$

$\gamma \leftarrow \gamma_{k-1}$

while $f(z) > f(x^k) + \langle \nabla f(x^k), z - x^k \rangle + \frac{1}{2\gamma} \|z - x^k\|^2$ do

| $z \leftarrow \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k))$
| $\gamma \leftarrow \beta \gamma$

end

- Requires one evaluation of $\text{prox}_{\gamma g}$ and $f$ per line search iteration
- Only a finite number of backtracking will be necessary
- Preserves convergence properties of the algorithms
Outline

1. Preliminary concepts, composite optimization, proximal mappings

2. Proximal gradient method

3. Duality

4. Accelerated proximal gradient

5. Newton-type proximal gradient methods

6. Concluding remarks
**Duality**

\[
\text{minimize } f(x) + g(Ax)
\]

- \(f\) and \(g\) are proper, closed, convex
- \(A\) matrix (e.g. data) or linear operator (e.g. finite differencing)

**Note:** computing \(\text{prox}_{\gamma(g \circ A)}\) is much more complex than \(\text{prox}_{\gamma g}\)
Duality

\[ \text{minimize } f(x) + g(Ax) \]

- \( f \) and \( g \) are proper, closed, convex
- \( A \) matrix (e.g. data) or linear operator (e.g. finite differencing)

**Note:** computing \( \text{prox}_{\gamma(g \circ A)} \) is much more complex than \( \text{prox}_{\gamma g} \)

Example: simple **bound** constraints become **polyhedral** constraints

\[
g = \delta_{\{z: z \leq b\}} \quad \Rightarrow \quad \text{prox}_{\gamma g} = \prod_{\{z: z \leq b\}} = \min(\cdot, b)
\]

\[
(g \circ A) = \delta_{\{x: Ax \leq b\}} \quad \Rightarrow \quad \text{prox}_{\gamma(g \circ A)} = \prod_{\{x: Ax \leq b\}} = ?
\]
Duality

\[ \text{minimize} \quad f(x) + g(Ax) \]

- \( f \) and \( g \) are proper, closed, convex
- \( A \) matrix (e.g. data) or linear operator (e.g. finite differencing)

**Note:** computing \( \text{prox}_{\gamma(g \circ A)} \) is much more complex than \( \text{prox}_{\gamma g} \)

Reformulate problem in **separable form** and solve the dual:

\[
\min_{x,z} f(x) + g(z)
\]

subject to \( Ax = z \)
Duality

**Primal**

\[
\begin{align*}
\text{minimize } & \quad f(x) + g(z) \\
\text{subject to } & \quad Ax = z
\end{align*}
\]

**Dual**

\[
\begin{align*}
\text{minimize } & \quad f^*(-A^T y) + g^*(y)
\end{align*}
\]

- Functions \( f^* \) and \( g^* \) are the **Fenchel conjugates** of \( f \) and \( g \)
  \[
  f^*(u) = \sup_x \{ \langle u, x \rangle - f(x) \} \quad \text{(similarly for } g) \]

- If \( f \) is \( \mu \)-strongly convex then \( f^* \) has \( \mu^{-1} \)-Lipschitz gradient
  \[
  \nabla f^*(u) = \arg\max_x \{ \langle u, x \rangle - f(x) \}
  \]

- **Moreau identity:** \( y = \prox_{\gamma g}(y) + \gamma \prox_{\gamma^{-1} g^*}(\gamma^{-1} y) \)

We can apply (accelerated) proximal gradient method to the dual
### Duality

#### Primal

\[
\begin{align*}
\text{minimize} & \quad f(x) + g(z) \\
\text{subject to} & \quad Ax = z 
\end{align*}
\]

#### Dual

\[
\begin{align*}
\text{minimize} & \quad f^*(-A^Ty) + g^*(y) 
\end{align*}
\]

- Functions \( f^* \) and \( g^* \) are the **Fenchel conjugates** of \( f \) and \( g \)
  \[
  f^*(u) = \sup_x \{ \langle u, x \rangle - f(x) \} \quad \text{(similarly for} \ g) 
  \]

- If \( f \) is \( \mu \)-strongly convex then \( f^* \) has \( \mu^{-1} \)-Lipschitz gradient
  \[
  \nabla f^*(u) = \arg\max_x \{ \langle u, x \rangle - f(x) \} 
  \]

- **Moreau identity:** \( y = \text{prox}_{\gamma g}(y) + \gamma \text{prox}_{\gamma^{-1}g^*}(\gamma^{-1}y) \)

We can apply (accelerated) proximal gradient method to the dual
Outline

1. Preliminary concepts, composite optimization, proximal mappings

2. Proximal gradient method

3. Duality

4. Accelerated proximal gradient

5. Newton-type proximal gradient methods

6. Concluding remarks
Accelerated proximal gradient (APG)

\[ \text{minimize } f(x) + g(x) \]

- When \( f \) and \( g \) are convex, convergence rate of proximal gradient is \( O(1/k) \)
- Proximal gradient reduces to gradient method whenever \( g \equiv 0 \)
- Gradient method **not optimal** for smooth convex problems
- Optimal convergence rate is \( O(1/k^2) \)
- **Nesterov (1983)** suggested simple modification that attains optimal rate
- Beck & Teboulle (2009) extended the method to composite problems
Accelerated proximal gradient (APG)

Start with $x^{-1} = x^0$, repeat

$$
\beta_k = \begin{cases} 
0 & \text{if } k = 0, \\
\frac{k-1}{k+2} & \text{if } k = 1, 2, \ldots 
\end{cases}
$$

$$
y^k = x^k + \beta_k(x^k - x^{k-1}) \quad \text{extrapolation step}
$$

$$
x^{k+1} = \text{prox}_{\gamma g}(y^k - \gamma \nabla f(y^k)) \quad \text{proximal gradient step}
$$

Theorem (Convergence rate of APG – convex case)

The iterates of APG with $\gamma \in (0, 1/L]$ satisfy

$$
\varphi(x^{k+1}) - \varphi_* \leq \frac{2L}{(k+2)^2} \|x^0 - x_*\|^2
$$

- APG faster than PG in theory (and practice!)
- Convergent extensions to nonconvex problems exist
Accelerated proximal gradient (APG)

Start with $x^{-1} = x^0$, repeat

$$\beta_k = \begin{cases} 
0 & \text{if } k = 0, \\
\frac{k-1}{k+2} & \text{if } k = 1, 2, \ldots 
\end{cases}$$

$$y^k = x^k + \beta_k(x^k - x^{k-1})$$

extrapolation step

$$x^{k+1} = \text{prox}_{\gamma g}(y^k - \gamma \nabla f(y^k))$$

proximal gradient step

Theorem (Convergence rate of APG – convex case)

The iterates of APG with $\gamma \in (0, 1/L]$ satisfy

$$\varphi(x^{k+1}) - \varphi_* \leq \frac{2L}{(k + 2)^2} \|x^0 - x_*\|^2$$

- APG faster than PG in theory (and practice!)
- Convergent extensions to nonconvex problems exist
Example: lasso (or basis pursuit)

\[
\text{minimize } \frac{1}{2}\|y - Ax\|^2 + \lambda \|x\|_1 \quad A \in \mathbb{R}^{1000 \times 2500}
\]

- Original signal \( \hat{x} \) is sparse with 100 nonzeros
- Output \( y = A\hat{x} + \mathcal{N}(0, \sigma) \) (SNR = 10)
- \( f(x) = \frac{1}{2}\|y - Ax\|^2, \quad g(x) = \lambda \|x\|_1 \)
- Lipschitz constant of \( \nabla f \) is \( \|A^\top A\| \)
Example: lasso (or basis pursuit)

\[
\text{minimize } \frac{1}{2} ||y - Ax||^2 + \lambda ||x||_1 \quad A \in \mathbb{R}^{1000 \times 2500}
\]

\[
\max_{\text{nnz}(x)} = 90
\]

\[
\lambda = 0.05 \lambda_{\text{max}}
\]
Example: lasso (or basis pursuit)

\[
\min_{x} \frac{1}{2} \| y - Ax \|^2 + \lambda \|x\|_1 \quad A \in \mathbb{R}^{1000 \times 2500}
\]

\[\lambda = 0.02 \lambda_{\text{max}} \]

\[\text{nnz}(x_*) = 187\]
Example: lasso (or basis pursuit)

\[
\begin{align*}
\text{minimize} \ & \frac{1}{2} \| y - Ax \|^2 + \lambda \| x \|_1 \\
A & \in \mathbb{R}^{1000 \times 2500}
\end{align*}
\]

\[
\lambda = 0.01 \lambda_{\text{max}}
\]

\[
nnz(x_*) = 404
\]
Outline

1. Preliminary concepts, composite optimization, proximal mappings

2. Proximal gradient method

3. Duality

4. Accelerated proximal gradient

5. Newton-type proximal gradient methods

6. Concluding remarks
Newton-type methods (smooth case)

1. Solve \textbf{minimize} $f(x)$ using

$$
x^{k+1} = \arg\min_x f(x^k) + \langle \nabla f(x^k), x - x^k \rangle + \frac{1}{2} \| x - x^k \|_{H_k}^2 \quad H_k \succ 0
$$

2. Solve $\nabla f(x) = 0$

$$
x^{k+1} = x^k - H_k^{-1} \nabla f(x^k) \quad H_k \text{ nonsingular}
$$

- \textbf{Equivalent approaches}
- Choose $H_k \approx \nabla^2 f(x^k) \equiv J\nabla f(x^k)$
- Gradient method corresponds to $H_k = I$
- \textbf{Damp} iterations using line search to guarantee convergence

Can we extend this to composite problems $f + g$?
Newton-type methods (smooth case)

1. Solve \textbf{minimize} \( f(x) \) using

\[
x^{k+1} = \arg\min_x f(x^k) + \langle \nabla f(x^k), x - x^k \rangle + \frac{1}{2} \| x - x^k \|^2_{H_k} \quad H_k > 0
\]

2. Solve \( \nabla f(x) = 0 \)

\[
x^{k+1} = x^k - H_k^{-1} \nabla f(x^k) \quad H_k \text{ nonsingular}
\]

- **Equivalent approaches**
- Choose \( H_k \approx \nabla^2 f(x^k) \equiv J \nabla f(x^k) \)
- Gradient method corresponds to \( H_k = I \)
- **Damp** iterations using line search to guarantee convergence

Can we extend this to composite problems \( f + g \)?
Variable metric proximal gradient

$$\text{minimize } f(x) + g(x)$$

$$d^k = \arg\min_d f(x^k) + \langle \nabla f(x^k), d \rangle + \frac{1}{2} \|d\|_{H_k}^2 + g(x^k + d) \quad H_k > 0$$

$$x^{k+1} = x^k + \tau_k d^k$$

where $H_k \approx \nabla^2 f(x^k)$. Define the \textit{scaled proximal mapping}

$$\text{prox}^{H_k}_g(x) = \arg\min_z \left\{ g(z) + \frac{1}{2} \|z - x\|_{H}^2 \right\}, \quad H \succ 0$$

Then the above is equivalent to

$$d^k = \text{prox}^{H_k}_g(x^k - H_k^{-1} \nabla f(x^k)) - x^k$$

$$x^{k+1} = x^k + \tau_k d^k$$

$\tau_k > 0$
Variable metric proximal gradient

- Becker, Fadili, 2012: \( f, g \) both convex, uses a modified SR1 method to approximate \( H_k \approx \nabla^2 f \)
- Lee et al., 2014: \( f, g \) both convex, show superlinear convergence when \( H_k \) is computed using quasi-Newton formulas, and solving subproblem inexactly
- Chouzenoux et al., 2014: \( f \) can be nonconvex, analyzes convergence of an inexact method under KL assumption
- Frankel et al., 2015: \( f, g \) can both be nonconvex
- ....
Variable metric proximal gradient

\[ \text{prox}_g^H(x) = \underset{z}{\text{argmin}} \left\{ g(z) + \frac{1}{2}\|z - x\|_H^2 \right\}, \quad H \succ 0 \]

Major **practical** drawback:

- No closed-form for computing \( \text{prox}_g^H \) in general, even for very simple \( g \)
- Closed form for \( \text{prox}_g^H \) if:
  - if \( g = \| \cdot \|_1 \) then \( H \) must be **diagonal**
  - if \( g = \| \cdot \|_2 \) then \( H \) must have **constant diagonal**
- Otherwise, need **inner iterative procedure** to compute \( \text{prox}_g^H \)
- Change in oracle
  - before: \( \nabla f \) and \( \text{prox}_\gamma g \) (often very simple to compute)
  - after: \( \nabla f \) and \( \text{prox}_g^H \) (much harder in general)
- Not as easily implementable as original proximal gradient method
Newton-type method for optimality conditions

\[ R(x) = x - \text{prox}_{\gamma g}(x - \gamma \nabla f(x)) = 0 \]  

- Any local minimum \( x_* \) satisfies \( R(x_*) = 0 \)
- System of nonlinear, nonsmooth equations
- **Idea:** apply Newton-type method to solve (♠)

\[ z^k = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]
\[ d^k = B_k(z^k - x^k) \]
\[ x^{k+1} = x^k + d^k \]

- Choose \( B_k \) (approximately) as \( JR(x^k)^{-1} \)
- Need to **damp** the last step to enforce convergence
- **Key:** penalty function for line search
Newton-type method for optimality conditions

\[ R(x) = x - \text{prox}_{\gamma g}(x - \gamma \nabla f(x)) = 0 \] (♠)

- Any local minimum \( x_* \) satisfies \( R(x_*) = 0 \)
- System of nonlinear, nonsmooth equations
- **Idea:** apply Newton-type method to solve (♠)

\[
\begin{align*}
  z^k &= \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \\
  d^k &= B_k(z^k - x^k) \\
  x^{k+1} &= x^k + d^k
\end{align*}
\]

- Choose \( B_k \) (approximately) as \( JR(x^k)^{-1} \)
- Need to **damp** the last step to enforce convergence
- **Key:** penalty function for line search
Newton-type method for optimality conditions

\[ R(x) = x - \text{prox}_{\gamma g}(x - \gamma \nabla f(x)) = 0 \]  

- Any local minimum \( x_\star \) satisfies \( R(x_\star) = 0 \)
- System of nonlinear, nonsmooth equations
- **Idea:** apply Newton-type method to solve (♠)

\[ z^k = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]
\[ d^k = B_k(z^k - x^k) \]
\[ B_k \in \mathbb{R}^{n \times n} \text{ nonsingular} \]
\[ x^{k+1} = x^k + d^k \]

- Choose \( B_k \) (approximately) as \( JR(x^k)^{-1} \)
- Need to **damp** the last step to enforce convergence
- **Key:** penalty function for line search
Forward-backward envelope

\[ \varphi_\gamma(x) = \min_z \left\{ Q_\gamma(z; x) = f(x) + \langle z - x, \nabla f(x) \rangle + g(z) + \frac{1}{2\gamma} ||z - x||^2 \right\} \]
Forward-backward envelope

\[ \varphi_{\gamma}(x) = \min_z \left\{ Q_{\gamma}(z; x) = f(x) + \langle z - x, \nabla f(x) \rangle + g(z) + \frac{1}{2\gamma} \| z - x \|^2 \right\} \]
Forward-backward envelope

\[ \varphi_\gamma(x) = \min_z \left\{ Q_\gamma(z; x) = f(x) + \langle z - x, \nabla f(x) \rangle + g(z) + \frac{1}{2\gamma} \|z - x\|^2 \right\} \]
Forward-backward envelope

\[ \varphi_\gamma(x) = \min_z \left\{ f(x) + \langle z - x, \nabla f(x) \rangle + g(z) + \frac{1}{2\gamma} \|z - x\|^2 \right\} \]

**Theorem**

If \( g \) is convex (results can be extended to \( g \) being nonconvex)

1. \( \varphi_\gamma \) is strictly continuous
2. \( \varphi_\gamma \leq \varphi \) for any \( \gamma > 0 \)
3. \( \varphi(z) \leq \varphi_\gamma(x) - \frac{1-\gamma L}{2\gamma} \|x - z\|^2 \) where \( z = \text{prox}_{\gamma g}(x - \gamma \nabla f(x)) \)
4. \( \varphi_\gamma(x) = \varphi(x) \) for any stationary point \( x \)
5. \( \inf \varphi_\gamma = \inf \varphi \) and \( \text{argmin} \varphi_\gamma = \text{argmin} \varphi \) for \( \gamma \in (0, L^{-1}) \)

- 1. implies that \( \varphi_\gamma \) is everywhere finite
- if \( \gamma < L^{-1} \), 2. and 3. imply that \( z \) (strictly) decreases \( \varphi_\gamma \)
- if \( \gamma < L^{-1} \), 5. implies minimizing \( \varphi \) equivalent to minimizing \( \varphi_\gamma \)
Forward-backward envelope

\[ \varphi_\gamma(x) = \min_z \left\{ f(x) + \langle z - x, \nabla f(x) \rangle + g(z) + \frac{1}{2\gamma} \| z - x \|^2 \right\} \]

**Theorem**

If \( g \) is convex (results can be extended to \( g \) being nonconvex)

1. \( \varphi_\gamma \) is strictly continuous
2. \( \varphi_\gamma \leq \varphi \) for any \( \gamma > 0 \)
3. \( \varphi(z) \leq \varphi_\gamma(x) - \frac{1-\gamma L}{2\gamma} \| x - z \|^2 \) where \( z = \text{prox}_{\gamma g}(x - \gamma \nabla f(x)) \)
4. \( \varphi_\gamma(x) = \varphi(x) \) for any stationary point \( x \)
5. \( \inf \varphi_\gamma = \inf \varphi \) and \( \text{argmin} \varphi_\gamma = \text{argmin} \varphi \) for \( \gamma \in (0, L^{-1}) \)

- 1. implies that \( \varphi_\gamma \) is everywhere finite
- if \( \gamma < L^{-1} \), 2. and 3. imply that \( z \) (strictly) decreases \( \varphi_\gamma \)
- if \( \gamma < L^{-1} \), 5. implies minimizing \( \varphi \) equivalent to minimizing \( \varphi_\gamma \)
Proximal averaged Newton-type method (PANOC)

\[ z^k = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]
\[ d^k = B_k(z^k - x^k) \quad B_k \in \mathbb{R}^{n \times n} \text{ nonsingular} \]
\[ x^{k+1} = (1 - \tau_k)z^k + \tau_k(x^k + d^k) \quad \tau_k \in (0, 1] \]

From the theorem: \( \varphi_\gamma \) continuous and

\[ \varphi_\gamma(z^k) \leq \varphi_\gamma(x^k) - \frac{1 - \gamma L}{2\gamma} \| x^k - z^k \|^2 \]

Therefore: \( \tau_k \in (0, 1] \) exists such that

\[ \varphi_\gamma(x^{k+1}) \leq \varphi_\gamma(x^k) - \alpha \frac{1 - \gamma L}{2\gamma} \| x^k - z^k \|^2 \quad \alpha \in (0, 1) \]
Proximal averaged Newton-type method (PANOC)

\[ z^k = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]
\[ d^k = B_k(z^k - x^k) \]
\[ x^{k+1} = (1 - \tau_k)z^k + \tau_k(x^k + d^k) \]
\[ B_k \in \mathbb{R}^{n \times n} \text{ nonsingular} \]
\[ \tau_k \in (0, 1] \]

minimize \( f(x) + g(x) \)

\[ f = \frac{1}{2} \text{dist}^2(\cdot, \ell) \]
\[ g = \delta_C \]
Proximal averaged Newton-type method (PANOC)

\[ z^k = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]
\[ d^k = B_k(z^k - x^k) \]
\[ x^{k+1} = (1 - \tau_k)z^k + \tau_k(x^k + d^k) \]

\[ B_k \in \mathbb{R}^{n \times n} \text{ nonsingular} \]
\[ \tau_k \in (0, 1] \]

minimize \( f(x) + g(x) \)

\[ f = \frac{1}{2} \text{dist}^2(\cdot, \ell) \]
\[ g = \delta_C \]
Proximal averaged Newton-type method (PANOC)

\[ z^k = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]
\[ d^k = B_k(z^k - x^k) \]
\[ x^{k+1} = (1 - \tau_k)z^k + \tau_k(x^k + d^k) \]

\[ B_k \in \mathbb{R}^{n \times n} \text{ nonsingular} \]
\[ \tau_k \in (0, 1] \]

\text{minimize} \quad f(x) + g(x)

\[ f = \frac{1}{2} \text{dist}^2(\cdot, \ell) \]
\[ g = \delta_C \]
Proximal averaged Newton-type method (PANOC)

\[ z^k = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]
\[ d^k = B_k(z^k - x^k) \]
\[ x^{k+1} = (1 - \tau_k)z^k + \tau_k(x^k + d^k) \]

\[ B_k \in \mathbb{R}^{n \times n} \text{ nonsingular} \]
\[ \tau_k \in (0, 1] \]

minimize \( f(x) + g(x) \)

\[ f = \frac{1}{2} \text{dist}^2(\cdot, \ell) \]
\[ g = \delta_C \]
Proximal averaged Newton-type method (PANOC)

\[ z^k = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]
\[ d^k = B_k(z^k - x^k) \]
\[ x^{k+1} = (1 - \tau_k)z^k + \tau_k(x^k + d^k) \]

\[ B_k \in \mathbb{R}^{n \times n} \text{ nonsingular} \]
\[ \tau_k \in (0, 1] \]

**minimize** \( f(x) + g(x) \)

\[ f = \frac{1}{2} \text{dist}^2(\cdot, \ell) \]
\[ g = \delta_C \]
Proximal averaged Newton-type method (PANOC)

\[ z^k = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]
\[ d^k = B_k(z^k - x^k) \]
\[ x^{k+1} = (1 - \tau_k)z^k + \tau_k(x^k + d^k) \]

\[ B_k \in \mathbb{R}^{n \times n} \text{ nonsingular} \]
\[ \tau_k \in (0, 1] \]

minimize \( f(x) + g(x) \)

\[ f = \frac{1}{2} \text{dist}^2(\cdot, \ell) \]
\[ g = \delta_C \]
Proximal averaged Newton-type method (PANOC)

\[ z^k = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)) \]
\[ d^k = B_k(z^k - x^k) \]
\[ x^{k+1} = (1 - \tau_k)z^k + \tau_k(x^k + d^k) \]
\[ B_k \in \mathbb{R}^{n \times n} \text{ nonsingular} \]
\[ \tau_k \in (0, 1] \]

How to choose \( B_k \)? Quasi-Newton: start with nonsingular \( B_0 \), update it s.t.

\[ B_k y^k = s^k \quad \text{(inverse secant condition)} \]
\[ \begin{cases} 
  s^k = x^k - x^{k-1} \\
  y^k = R(x^k) - R(x^{k-1}) 
\end{cases} \]

- (Modified) Broyden method yields superlinear convergence
- Limited-memory BFGS: works well in practice, no need to store \( B^k \)
- Products with \( B^k \) computed in \( O(n) \) using inner products only
Example: lasso (or basis pursuit)

$$\min_{x} \frac{1}{2} \| y - Ax \|^2 + \lambda \| x \|_1 \quad A \in \mathbb{R}^{1000 \times 2500}$$

$$\lambda = 0.05 \lambda_{\text{max}}$$

$$\text{nnz}(x_*) = 90$$
Example: lasso (or basis pursuit)

\[
\minimize_x \frac{1}{2} \| y - Ax \|^2 + \lambda \| x \|_1 \quad A \in \mathbb{R}^{1000 \times 2500}
\]

\[
\lambda = 0.02\lambda_{max}
\]

\[
nnz(x_*) = 187
\]
Example: lasso (or basis pursuit)

\[
\text{minimize} \quad \frac{1}{2} \|y - Ax\|^2 + \lambda \|x\|_1 \quad A \in \mathbb{R}^{1000 \times 2500}
\]

\[
\lambda = 0.01 \lambda_{\text{max}}
\]

\[
nnz(x_*) = 404
\]
Outline

1. Preliminary concepts, composite optimization, proximal mappings

2. Proximal gradient method

3. Duality

4. Accelerated proximal gradient

5. Newton-type proximal gradient methods

6. Concluding remarks
Concluding remarks

Proximal gradient (PG) method:
- Extends classical gradient descent to composite problems
- Convergence rate guarantees in the convex case
- Convergence to local minima in the nonconvex case (under assumptions)
- Accelerated variants greatly improve convergence (and makes it practical)

Newton-type PG:
- Variable metric PG exist, which require solving inner subproblem in general
- PANOC: Newton-type method for the composite optimality conditions
- Same oracle as PG: $\nabla f$ and $\text{prox}_{\gamma g}$
- Same global convergence as PG
- Much faster local convergence using e.g. L-BFGS directions
References

Theory books:
- Bertsekas, "Convex Optimization Theory", 2009
- Rockafellar, Wets, "Variational Analysis", 2009
- Bauschke, Combettes, "Convex Analysis and Monotone Operator Theory in Hilbert Spaces", 2017

Algorithms books:
- Beck, "First-Order Methods in Optimization", 2017

(Accelerated) proximal gradient method:
- Nesterov, "Gradient methods for minimizing composite functions", 2013

Newton-type proximal gradient methods:
- Lee et al, "Proximal Newton-Type Methods for Minimizing Composite Functions", 2014
- Chouzenoux et al, "Variable Metric Forward-Backward Algorithm for Minimizing the Sum of a Differentiable Function and a Convex Function", 2014
- Frankel et al, "Splitting Methods with Variable Metric for Kurdyka-Lojasiewicz Functions and General Convergence Rates", 2015
- Themelis et al, "Forward-backward envelope for the sum of two nonconvex functions: ...", 2016
- Stella et al, "Newton-type Alternating Minimization Algorithm for Convex Optimization", 2018
Proximal Gradient Algorithms: Applications in Signal Processing
Part III

Niccolò Antonello
niccolo.antonello@idiap.ch

EUSIPCO 2019

IDIAP Research Institute
StructuredOptimization.jl

Niccolò Antonello (Idiap Research Institute)
Outline

- Introduction to Julia
- StructuredOptimization.jl
  - AbstractOperators.jl
  - ProximalOperators.jl
  - ProximalAlgorithms.jl
- Demos
Introduction to Julia
The Julia language

- General-purpose programming language
- Designed specifically for scientific computing
- Young language
  - Born in 2012
  - 1.0 stable release Summer 2018
The Julia language features

- Dynamic language (Like Python, MATLAB)
- Interoperability
  - Easily call other languages (C, Fortran)
- Designed to be fast
  - Approaches C, faster than Python
- Open Source
• Syntax very close to MATLAB

In [118]:
# solving a random linear system of equations (y = A*x)
A = randn(3,3)
x = randn(3)    # just a vector
y = A\x

Out[118]:
3-element Array{Float64,1}:
  2.0664785413005244
 -3.0015294370755936
 -1.5437559125502291
• ...but often very close to Python

In [119]:
a, b = (1, 2)  # Tuples

Out[119]:
(1, 2)

In [120]:
[i + j for i = 1:3, j = 1:3]  # Comprehensions

Out[120]:
3×3 Array{Int64,2}:
2 3 4
3 4 5
4 5 6
• Ahead-of-time compilation

In [121]:

    function foo(x)
        return sum(x)
    end

x = randn(1000)
# first time you run a function code is compiled
@time foo(x)
# second time code is re-used
@time foo(x);

    0.024094 seconds (3.84 k allocations: 170.800 KiB)
    0.000005 seconds (5 allocations: 176 bytes)
Learning Julia

- Full documentation [docs.julialang.org](https://docs.julialang.org/en/v1/)
- Responsive and helpful community in [Discourse](https://discourse.julialang.org)
- Help funtion
cos # Interactive help through Read–Eval–Print Loop (REPL)

Compute cosine of \( x \), where \( x \) is in radians.

\[
\cos(A::AbstractMatrix)
\]

Compute the matrix cosine of a square matrix \( A \).

If \( A \) is symmetric or Hermitian, its eigendecomposition \( \text{eigen}(A) \) is used to compute the cosine. Otherwise, the cosine is determined by calling \( \text{exp} \).

### Examples

```julia
jldoctest
julia> cos(fill(1.0, (2,2)))
2×2 Array{Float64,2}:
   0.291927  -0.708073
  -0.708073   0.291927
```

\[
\cos(x::AbstractExpression)
\]

Cosine function:
\cos(x)

See documentation of AbstractOperator.Cos.
Integrated development environment (IDE)

- Juno ([http://junolab.org](http://junolab.org)) (Atom extension) ← user friendly IDE
- Jupyter notebooks ([https://jupyter.org](https://jupyter.org)) (such as this one) available also online ([juliabox (https://juliabox.com)](https://juliabox.com))
- Other editors such as Vim, Spacemacs have dedicated extensions
Package manager

- Julia has built-in package manager

- Installing packages
  
  julia> ] add AbstractOperators
Optimization in Julia

- JuMP.jl (https://github.com/JuliaOpt/JuMP.jl)
  - LP, MIP, SOCP, NLP
  - Convex Optimization (like MATLAB's CVX)
- Optim.jl (https://github.com/JuliaNLSolvers/Optim.jl)
  - Smooth Nonlinear Programming
StructuredOptimization.jl

- **Large scale** and **nonsmooth** problems
- Convex & Nonconvex
- PG algorithms
- Modeling language with mathematical formulation
StructuredOptimization.jl: Package ecosystem

Joins 3 independent packages:

- ProximalOperators.jl
- AbstractOperators.jl
- ProximalAlgorithms.jl
ProximalOperators.jl
Proximal Operators

\[ y = \text{prox}_{\gamma g}(x) = \arg \min_z \left\{ g(z) + \frac{1}{2\gamma} \| z - x \|^2 \right\}, \]

where \( g : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}, \gamma > 0. \)

- Generalization of projection
- Often has efficient closed form
Library of functions with efficient prox

- Indicators of sets
  - Norm balls e.g. $S = \{ x : \sum_i |x_i| \leq r \}$
- Norm and regularization functions
  - Norms e.g. $\|x\|_1, \|x\|_\infty$
- Penalties and other functions
  - Least squares, Huber loss, Logistic loss
Example: $\mathcal{L}_1$-norm

**In [126]:**
```
using ProximalOperators # load a package
lambda = 3.5    # regularization parameter
f = NormL1(lambda) # one can create the L1-norm as follows
```

**Out[126]:**
```
description : weighted L1 norm
domain      : AbstractArray{Real}, AbstractArray{Complex}
expression  : $x \mapsto \lambda \|x\|_1$
parameters  : $\lambda = 3.5$
```
In [128]:
    # `prox` evaluates proximal operator at `x`  
    # optional positive stepsize `gamma`  
    gamma = 0.5  
    x = randn(10)  
    y, fy = prox(f, x, gamma)  
    # returning proximal point y and the value of the f(y)

Out[128]: ([0.0, 0.0, 0.0, -0.00310438, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.010865314173881369)

In [129]:
    # `prox!` evaluates the proximal operator in-place  
    # (Note: by convention func. with ! are in-place)  
    y = similar(x);  # pre-allocate y  
    fy = prox!(y, f, x, gamma)

Out[129]: 0.010865314173881369
ProximalOperators.jl calculus rules

Modify & combine functions

- Convex conjugate
- Functions combinations (Separable sum)
- Function regularization (Moreau Envelope)
- Pre and post composition
Example: Precomposition

Construct a least squares function with diagonal matrix:

\[ f(x) = \|Dx - y\|^2 \]

In [130]: `d, y = randn(10), randn(10);`
In [131]:
f_ls = SqrNormL2() # smooth function

Out[131]:

description : weighted squared Euclidean norm
domain : AbstractArray{Real}, AbstractArray{Complex}
expression : \( x \mapsto (\lambda/2)||x||^2 \)
parameters : \( \lambda = 1.0 \)

In [132]:
f = PrecomposeDiagonal(f_ls, d, y)

Out[132]:

description : Precomposition by affine diagonal mapping of weighted squared Euclidean norm
domain : AbstractArray{Real}, AbstractArray{Complex}
expression : \( x \mapsto f(diag(a)*x + b) \)
parameters : \( f(x) = x \mapsto (\lambda/2)||x||^2, a = Array\{Float64,1\}, b = Array\{Float64,1\} \)
In [133]:
x = randn(10)
y, fy = prox(f, x)

Out[133]:
([1.73048, -0.384114, 0.298391, 0.885389, -0.567672, -0.166953, -0.368458, -0.240603, 0.396272], 2.1379379614662475)

In [134]:
gradfx, fx = gradient(f, x)

Out[134]:
([-0.0572666, 2.11036, 2.10945, 0.358283, -1.87252, 0.0232842, 0.177813, -1.00873, 1.43095, -0.21392], 5.933046424575226)
AbstractOperators
AbstractOperators.jl extends syntax typically used for matrices to mappings.

In [135]:
using AbstractOperators
A = DCT(3,4) # create a 2-D Discrete Cosine Transform operator

Out[135]: 𝒇∶ℝ^(3, 4) → ℝ^(3, 4)

In [136]:
x = randn(3,4) # notice that x is not restricted to a vector!
y = A*x # apply the linear operator

Out[136]:
3×4 Array{Float64,2}:
  0.383866  -1.40719  -0.727502  -0.452948
-0.468083   0.740363  -0.237086  -0.49221
  0.492823  -1.47733   0.313025  -2.24597
Fast (Matrix free) operators library

- Basic operators (Eye, DiagOp)
- DSP
  - Transformations (e.g. DFT, DCT)
  - Filtering (e.g. Conv, Xcorr)
- Nonlinear functions (Cos, Sin)
Matrix free?

Use fast operators, avoid building matrices.

In [138]:  # Fourier transform
N = 2^9
x = randn(Complex{Float64},N)
A = [exp(-im*2*pi*k*n/(N)) for k =0:N-1, n=0:N-1];  #Fourier Matrix

In [139]: A_mf = DFT(Complex{Float64},(2^9,))  # (matrix free)

Out[139]: ℱ ℂ^512 → ℂ^512
In [140]:
    # not good for memory
    println("Size Fourier Matrix: ", sizeof(A))
    println("Size Abstract Operator: ", sizeof(A_mf))

    Size Fourier Matrix:  4194304
    Size Abstract Operator: 24

In [141]:
    # ...and neither for speed!
    print("Fourier Matrix:")
    @time A*x
    print("Abstract Operators:")
    @time A_mf*x;

    Fourier Matrix:  0.000896 seconds (5 allocations: 8.281 KiB)
    Abstract Operators:  0.000017 seconds (5 allocations: 8.281 KiB)
AbstractOperators.jl calculus rules

- Concatenation HCAT, VCAT, DCAT
- Composition
  - Linear and Nonlinear
- Transformations
  - Scale, Affine addition
  - Adjoint and Jacobian
Automatic differentiation

\[ f(x) = \tilde{f}(ABx), \]

where \( A \) and \( B \) linear operators

\[ \nabla f(x) = B^* A^* \nabla \tilde{f}(ABx) \]

\( A^* \) and \( B^* \) adjoint operators with fast transformation
In [143]:

# define operators
B = IDCT(5)  # inverse DCT transform
A = FiniteDiff((5,))  # finite difference operator
B, A

Out[143]:

(ℱc⁻¹  R^5 -> R^5 , δx  R^5 -> R^4 )

In [144]:

C = A*B  # can combine operators

Out[144]:

δx*ℱc⁻¹  R^5 -> R^4
In [145]:

```plaintext
x = randn(5)  # random point
r = C*x  # r = A*B*x (Forward pass)
f_t = SqrNormL2()  # least squares cost function
f = f_t(r)  # evaluate f(x) = g(A*B*x)
```

Out[145]: 4.5266875469669605
\[ \nabla f(x) = B^* A^* \nabla \tilde{f}(ABx) \]

In [146]:
\[ \nabla f_t, f_{tx} = \text{gradient}(f_t, r) \]
\[ \nabla \tilde{f} = C' \nabla f_t; \quad \# \text{get gradient: adjoint operator } C' \ (\text{Backpropagation}) \]
# gradient using finite differences

```julia
using LinearAlgebra

x_eps = zero(x)
∇f_FD = zero(x)

for i = 1:length(x_eps)
    x_eps .= 0
    x_eps[i] = sqrt(eps())
    ∇f_FD[i] = (f_t(C*(x.+x_eps)-f)./sqrt(eps()))

end	norm( ∇f_FD - ∇f ) # testing gradient using
```

Out[147]: 2.06775621675667e-7
ProximalAlgorithms.jl
Proximal algorithms for nonsmooth optimization in Julia.

- (Accelerated) **Proximal Gradient** (aka Forward-backward)
- **PANOC**
Many others:

- Asymmetric forward-backward-adjoint algorithm (AFBA)
- Chambolle-Pock primal dual algorithm
- Davis-Yin splitting algorithm
- Douglas-Rachford splitting algorithm
- Vū-Condat primal-dual algorithm
PANOC, ZeroFPR, ForwardBackward

Solve problem:

$$\arg\min_x f(Ax) + g(x)$$

- $f$ smooth function
- $A$ linear operator
- $g$ nonsmooth function
Example: sparse deconvolution

\[ x^* = \arg\min_x \frac{1}{2} \| h * x - y \|^2 + \lambda \| x \|_1 \]

- \( h \) impulse response (FIR)
- \( y \) noisy measurement
- \( x \) unknown source clean signal (sparse)
In [148]:
using DelimitedFiles, Plots
h = readdlm("data/h.txt")[:,]  # load impulse response
plot(h; xlabel="Samples", ylabel="Amplitude")
```python
In [149]:
```
```
using SparseArrays, Random;
Random.seed!(123)

x_gt = Array(sprand(2000, 0.05))  # random sparse vector
plot(x_gt; xlabel="Samples", ylabel="Amplitude", label="Ground truth")
```
```
```
Out[149]:
```
In [150]:

```python
using DSP
y = conv(h, x_gt) + 1e-3 .* randn(length(h)+length(x_gt)-1)
plot(y; xlabel="Samples", ylabel="Amplitude", label="Noisy Measurement")
```

Out[150]:
Construct \( f(x) = \frac{1}{2} \| h * x - y \|^2 \) and \( g(x) = \lambda \| x \|_1 \)

In [151]:
```latex
\textbf{using} AbstractOperators
linop = Conv(size(x_gt), h) \# convolution operator
```

Out[151]:
\[ \mathbb{R}^{2000} \rightarrow \mathbb{R}^{2999} \]

In [152]:
```latex
\textbf{using} ProximalOperators
smooth = PrecomposeDiagonal( SqrNormL2(), 1.0, -y )
nonsmooth = NormL1(1e-3);
```
Create PANOC solver

In [153]:
```python
using ProximalAlgorithms: PANOC
solver = PANOC(verbosetrue); # set options
```

In [154]:
```python
x0 = zeros(length(x_gt)) # initial estimate
println(" it | γ | res | τ ")
x0, its = solver(x0; f=smooth, A=linop, g=nonsmooth);
```

<table>
<thead>
<tr>
<th>it</th>
<th>γ</th>
<th>res</th>
<th>τ</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.200e+01</td>
<td>7.836e-04</td>
<td>1.000e+00</td>
</tr>
<tr>
<td>20</td>
<td>2.200e+01</td>
<td>1.418e-04</td>
<td>1.000e+00</td>
</tr>
<tr>
<td>30</td>
<td>2.200e+01</td>
<td>1.363e-06</td>
<td>1.000e+00</td>
</tr>
<tr>
<td>40</td>
<td>2.200e+01</td>
<td>8.510e-08</td>
<td>1.000e+00</td>
</tr>
<tr>
<td>47</td>
<td>2.200e+01</td>
<td>6.195e-09</td>
<td>1.000e+00</td>
</tr>
</tbody>
</table>
In [155]:
plot(x_gt; xlabel="Samples", ylabel="Amplitude", label="Ground truth")
plot!(x0, label="Estimate")

Out[155]:

![Plot showing ground truth and estimate with corresponding labels and axis titles](image-url)
StructuredOptimization.jl
Structured optimization problem:

$$\minimize_{x} f_1(A_1x) + f_2(A_2x) + \cdots + f_N(A_Nx)$$

- Cost function composed of different terms
- $f_i$ loss functions
- $A_i$ linear operators
- Constraints: indicator functions
Structured optimization problem:

\[
\min_x f_1(A_1x) + f_2(A_2x) + \cdots + f_N(A_Nx)
\]

StructuredOptimization.jl converts it to the PG general problem:

\[
\min_x f(Ax) + g(x)
\]
• $f$ smooth (differentiable)
  - automatic differentiation → AbstractOperators.jl
• $g$ nonsmooth (including constraints)
  - efficient proximal mappings → ProximalOperators.jl
Example: Sparse Deconvolution

\[ x^* = \arg\min_x \frac{1}{2} \| h \ast x - y \|^2 + \lambda \| x \|_1 \]

In [156]:
```
using StructuredOptimization
x = Variable(length(x_gt))  # define optimization variable
```

Out[156]:
```
Variable(Float64, (2000,))
```

In [157]:
```
# (ls short hand for 0.5*norm(...)^2 )
@minimize ls( conv(x,h) - y ) + 1e-3*norm(x, 1);  # solve problem
```
In [158]:
plot(x_gt; xlabel="Samples", ylabel="Amplitude", label="Ground truth")
plot!(~x, label="Sparse Estimate")
# ~x to access the solution

Out[158]:

![Graph showing the comparison between Ground truth and Sparse Estimate]
Matrix free optimization

In [159]:
~x .== 0
_, its = @time @minimize ls( conv(x,h) - y ) + 1e-3*norm(x, 1)
x_mf = copy(~x);

0.185090 seconds (17.13 k allocations: 6.785 MiB, 6.34% gc time)

In [160]:
~x .== 0; Nx = length(x_gt)
H = hcat([zeros(i);h;zeros(Nx-1-i)] for i = 0:Nx-1...)
_, its_mf = @time @minimize ls( H*x - y ) + 1e-3*norm(x, 1);
its_mf == its

0.342836 seconds (17.44 k allocations: 6.720 MiB)

Out[160]: true
Example: constraint optimization

Refine the LASSO solution using:

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \| (Ez) \ast h - y \|^2 \\
\text{subject to} & \quad z \geq 0
\end{align*}
\]

- \(z\) is a vector of length \(\|x\|_0\)
- \(E\) is a matrix that expands \(z\) to the support of \(x^*\)
In [162]:

```plaintext
using LinearAlgebra
idx = findall(!((~x) .≈ 0.0 ))  # indices nonzero elements
E = Diagonal(ones(length(~x)))[:,idx]  # create expansion matrix

z = Variable((~x)[idx])  # initialize var. with nonzero elements
@minimize ls( conv(E*z,h) - y ) st z >= 0.0;  # solve problem
```
In [163]:
plot(x_gt, xlabel="Samples", ylabel="Amplitude", label="Ground truth")
plot!(~x, label="Sparse Estimate")
scatter!(idx, ~z, label="Refined solution")

Out[163]:

![Graph showing samples and amplitude with annotations for ground truth, sparse estimate, and refined solution.]
Limitations

- Only PG algorithms supported
- $g$ must be efficiently computable proximal mappings
Nonsmooth function $g(Bx)$ must satisfy:

1. $B$ is a **tight frame**
   - $BB^* = \mu I$, where $\mu \geq 0$

2. $g$ is a separable sum: $g(Bx) = \sum_j h_j(C_jx_j)$
   - $x_j$ non-overlapping slices of $x$
   - $C_j$ tight frames
In [164]:
    \[
    n = 10 \\
    A, b = \text{randn}(2*n,n), \text{randn}(2*n) \\
    x = \text{Variable}(n) \\
    \text{@minimize } \text{ls}(A*x-b)+\text{norm}(\text{dct}(x),1); \\
    \# \text{nonsmooth fun composed with orthogonal operator (rule 1)}
    \]

In [165]:
    \text{@minimize } \text{ls}(A*x-b)+\text{norm}(A*x,1) \\
    \# \text{rule 1 not satisfied!}

In [166]:
    \text{@minimize } \text{ls} ( A*x - b ) + \text{maximum}(x) \text{ st } x >= 2.0 \\
    \# \text{rule 2 not satisfied!}

In [167]:
    \text{@minimize } \text{ls} ( A*x - b ) + \text{maximum}(x[1:5]) \text{ st } x[6:10] >= 2.0; \\
    \# \text{accepted: optimization variables partitioned in nonoverlapping groups}
Demos

- Line Spectral estimation
- Video background removal
- Audio Declipping
Line Spectral Estimation
Goal:

- recover frequencies & amplitudes of signal $y$

Assumption:

- $y$ sparse mixture of $N$ sinusoids.
Simple solutions:

- DFT
- zero-padded DFT of $y$ with $s$ super-resolution factor.
In [18]:
plot(f,  abs.(fft(y) ./ Nt )[1:div(Nt,2)+1]; label = "dft")
plot!(f_s, abs.(xzp ./ Nt )[1:div(s*Nt,2)+1]; label = "dft zero pad.", ylim=[0.2;1], xlim=[0;4e3], xlabel="Frequency", ylabel="Amplitude")
scatter!(fk, abs.(A) ./2; label = "ground truth")

Out[18]:
Spectral leakage:

- frequencies merge
- amplitude not estimated correctly
Lasso formulation:

\[
x_1^* = \arg\min_{x} \frac{1}{2} \| SF^{-1} x - y \|^2 + \lambda \| x \|_1,
\]

- \( F^{-1} \): Inverse Fourier transform
- \( S \): selection mapping takes first \( N \) samples
In [19]: using StructuredOptimization

    x = Variable(Complex{Float64}, s*Nt) # define complex-valued variable
    lambda = 1e-3*norm(xzp./(s*Nt), Inf) # set lambda

    @minimize ls(ifft(x)[1:Nt]-complex(y))+lambda*norm(x,1) with PANOC(tol = 1e-8)
    x1 = copy(~x); # copy solution
In [20]:
scatter(f_s, abs(x1[1:div(s*Nt,2)+1]./(s*Nt) ); label = "LASSO", m=:square)
plot!(f_s, abs(xzp./Nt )[1:div(s*Nt,2)+1]; label = "dft zero pad.", ylim=[0.2;1]
, xlim=[0;4e3], xlabel="Frequency", ylabel="Amplitude")
scatter!(fk, abs(A) ./2; label = "ground truth", ylim=[0.2;1], xlim=[0;4e3])

Out[20]:

![Graph showing frequency versus amplitude with LASSO, dft zero pad, and ground truth markers.](image-url)
**Lasso results**

- $\mathbf{x}_1^\ast$ estimates improve!
- Amplitude usually underestimated

**Non-convex problem**

$$
\mathbf{x}_0^\ast = \arg\min_{\mathbf{x}} \frac{1}{2} \| S F^{-1} \mathbf{x} - \mathbf{y} \|^2 \text{ s.t. } \| \mathbf{x} \|_0 \leq 2N.
$$
In [22]:
# notice that following problem is warm-started by previous solution
@minimize ls(ifft(x)[1:Nt]-complex(y)) st norm(x,0) <= 2*N with PANOC(tol = 1e-8);
x0 = copy(~x);
In [23]: scatter!(f_s, abs.(x0[1:div(s*Nt,2)+1]./(s*Nt))); label = "non-cvx", m=:star)

Out[23]:

![Graph showing frequency vs. amplitude with labels LASSO, dft zero pad, ground truth, and non-cvx.](image)
Demo: Video Background removal
Video

- Static background
- Moving foreground

Goal

- Separate foreground from static background
In [5]:
using Images
include("utils/load_video.jl")
n, m, l = size(Y)
Gray.([Y[:,1] Y[:,2] Y[:,3]])

Out[5]:
Low rank approximation

\[
\min_{L, S} \frac{1}{2} \|L + S - Y\|^2 + \lambda \|\text{vec}(S)\|_1 \\
\text{subject to } \text{rank}(L) \leq 1
\]

- **Y**: \(l\)-th column has \(l\)-th frame
- **L**: background (low-rank)
- **S**: foreground (sparse)
\[
\text{minimize} \quad \frac{1}{2} \| L + S - Y \|^2 + \lambda \| \text{vec}(S) \|_1 \\
\text{subject to} \quad \text{rank}(L) \leq 1
\]

In [6]:

```
using StructuredOptimization

Y = reshape(Y,n*m,l)  # reshape video
L = Variable(n*m,l)   # define variables
S = Variable(n*m,l)

@minimize ls(L+S-Y) + 3e-2*norm(S,1) st rank(L) <= 1 with PANOC(tol = 1e-4);
```
In [7]:

```
L, S = ~L, ~S  # extract vectors from variables
S[ S .!= 0 ] .= S[ S .!= 0 ] .+L[ S .!= 0 ]
# add background to foreground changes in nonzero elements
S[S.== 0 ] .= 1.0
# put white in null pixels
Y, S, L = reshape(Y,n,m,l), reshape(S,n,m,l), reshape(L,n,m,l);
```
In [8]:
idx = [1;3]
img = Gray.(vcat([ [Y,:,:], [S,:,:], [L,:,:]] for i in idx))

Out[8]:

![Images](image.png)
Demo: Audio de-clipping
Audio recording of loud source can saturate
In [1]:
using WAV, Plots
# load wav file
yt, Fs = wavread("data/clipped.wav"); yt = yt[:,1][:]
C = maximum(abs.(yt))    # clipping level
# plotting a frame of the audio signal
idxs = 2^11+1:2^12;
In [2]:
plot(yt[idxs]; label = "clipped signal", xlabel="Samples", ylabel="Amplitude", ylim=[-0.4; 0.4])
plot!([1; length(idxs)], [C.*ones(2), -C.*ones(2)]; color=[:red :red], label = ["s

Out[2]:

![Graph showing clipped signal and saturation]
minimize \quad \frac{1}{2} \| F_{i,c} x - y \|^2, \\
subject \quad to \quad \| My - M\tilde{y} \| \leq \epsilon \\
\quad M_+ y \geq C \\
\quad M_- y \leq -C \\
\quad \| x \|_0 \leq N
Input:

- \( \tilde{y} \) frame of clipped signal

Optimization variables:

- \( x \) DCT transform declipped frame (\( F_{i,c} \) brings to time domain)
- \( y \) time domain declipped frame
Constraints on $\mathbf{y}$:

- $\mathbf{M}$ selection matrix of uncorrupted samples
- $\mathbf{M}_{\pm}$ selection matrix of saturated samples

Constraints on $\mathbf{x}$:

- $\mathbf{x}$ is sparse $\ell_0$-ball constraint (sparsity DCT domain) (nonconvex)
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \| F_{i,c} x - y \|^2, \\
\text{subject to} & \quad \| M y - M \tilde{y} \| \leq \epsilon \\
& \quad M_+ y \geq C \\
& \quad M_- y \leq -C \\
& \quad \| x \|_0 \leq N
\end{align*}

**Nonconvex problem**: refine solution by increasing $N$
```python
using StructuredOptimization, DSP

Nl = 2^10  # time window length
Nt = length(yt)  # signal length
yd = zeros(Nt)  # allocate declipped output

x, y = Variable(Nl), Variable(Nl)  # optimization variables
f = lse(idct(x) - y)  # cost function
yw = zeros(Nl)  # allocate weighted clipped frame

# weight window options
win = sqrt.(hanning(Nl+1)[1:Nl])
overlap = div(Nl,2);
```
In [ ]:

```python
z, ε = 0, sqrt(1e-5)  #weighted Overlap-Add
while z+Nl < Nt
    fill!(~x,0.); fill!(~y,0.)  # initialize variables
    Ip = sort(findall(  yt[z+1:z+Nl] .>=  C  ))  #pos clip idxs
    In = sort(findall(  yt[z+1:z+Nl] .<=  -C   ))  #neg clip idxs
    I  = sort(findall(abs.(yt[z+1:z+Nl]).< C))  #uncor idxs

    yw .= yt[z+1:z+Nl].*win  # weighted frame
    for N = 30:30:30*div(Nl,30)  # increase active components DCT
        cstr = (norm(x,0) <= N,
                norm(y[I]-yw[I]) <= ε,
                y[Ip]  >=  C.*win[Ip],
                y[In]  <=  -C.*win[In]  )
        @minimize f st cstr with PANOC(tol = 1e-4, verbose = false)
        if norm(idct(~x) - ~y) <= ε break end
    end
    yd[z+1:z+Nl] .+= (~y).*win  # store declipped signal
for N = 30:30:30*div(Nl,30)  # increase active components DCT
        cstr = (norm(x,0) <= N,
                norm(y[I]-yw[I]) <= ε,
                y[Ip]  >=  C.*win[Ip],
                y[In]  <=  -C.*win[In]  )
        @minimize f st cstr with PANOC(tol = 1e-4, verbose = false)
        if norm(idct(~x) - ~y) <= ε break end
    end
```

```
end
```
In [ ]:

```python
plot(yd[idxs], label = "declipped signal", xlabel="Time (samples)", ylabel="Amplitude", ylim=[-0.4; 0.4])
plot!(yt[idxs]; label = "clipped signal")
plot![1:length(idxs)], [C.*ones(2), -C.*ones(2)]; color=[:red :red], label = ["saturation" ""]
```
```julia
using LinearAlgebra
wavwrite(0.9 .* normalize(yd[:,], Inf), "data/declipped.wav"; Fs = Fs, nbits = 16, compression=WAVE_FORMAT_PCM) # save wav file
```

Clipped audio:

Declipped audio:
Conclusions

- Proximal gradient (PG) methods apply to a wide variety of signal processing tasks.
- PG framework applies to large-scale inverse problems with non-smooth terms.
- PG framework applies to both convex and nonconvex problems.
- Accelerated and Newton-type extensions of PG enjoy much faster convergence.
- Julia software toolbox offers a modeling language with mathematical notation.
- More signal processing demos & examples available @ https://github.com/kul-forbes/StructuredOptimization.jl.
Conclusions

Additional resources


- Software packages:
  - https://github.com/kul-forbes/ProximalOperators.jl
  - https://github.com/kul-forbes/AbstractOperators.jl
  - https://github.com/kul-forbes/ProximalAlgorithms.jl
  - https://github.com/kul-forbes/StructuredOptimization.jl

Acknowledgements

- Thanks to Andreas Themelis (KU Leuven) and Pontus Giselsson (Lund University) who kindly provided some of the figures used.

- Thanks to AWS AI Labs, IDIAP, KU Leuven, FWO and ERC for funding.