

ASTR8150/PHYS8150

Optimization

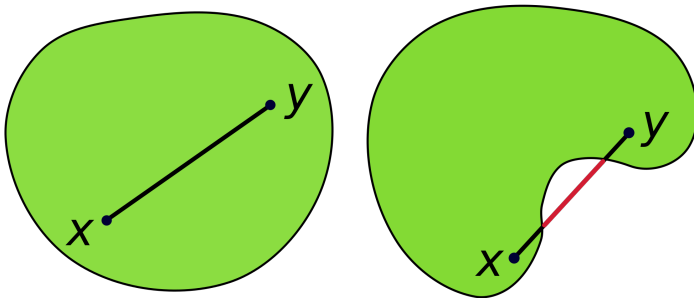
Fabien Baron

Georgia State University

baron@chara.gsu.edu

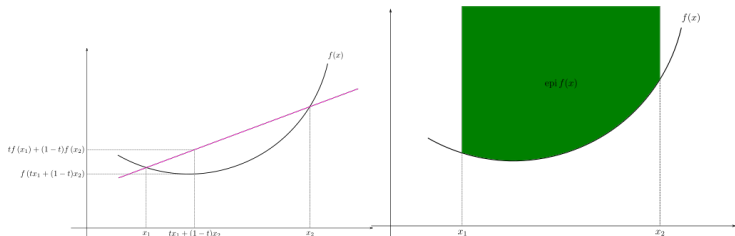
Fall 2021

Convexity of a set



- In a convex set, for every pair of points within the region, every point on the straight line segment that joins the pair of points is also within the region.
- A set which is hollow or has an indent, for example, a crescent shape, is not convex.

Convexity of a function



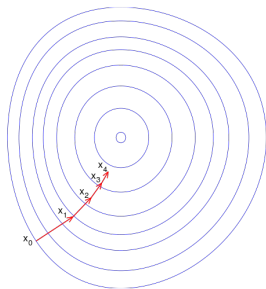
- A real-valued function is called convex if the set of points on or above the graph of the function (epigraph) is a convex set.
- For a twice differentiable function of a single variable, if the second derivative is always greater than or equal to zero for its entire domain then the function is convex. Examples: $f(x) = x^2$ or $f(x) = e^x$
- Jensen's inequality: if X is a convex set and $f : X \rightarrow \mathbb{R}$, f is convex if:

$$\forall x_1, x_2 \in X, \forall t \in [0, 1] : \quad f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2).$$

Smoothness of a function

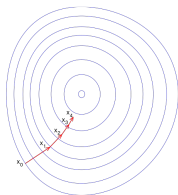
- The smoothness of a function is a property measured by the number of derivatives it has which are continuous. A smooth function is a function that has derivatives of all orders everywhere in its domain.
- The function $f(x) = |x|^k$ is continuous and k times differentiable at all x . But at $x = 0$ it is not $(k + 1)$ times differentiable.
- The norms ℓ_2 , ℓ_1 and pseudo-norm ℓ_0 are used in regularization. ℓ_2 is convex, differentiable and smooth. ℓ_1 is convex, differentiable but nonsmooth. ℓ_0 is non-convex and nonsmooth.

Gradient descent (1)



- Gradient descent is based on the observation that if the multi-variable function $f(\mathbf{x})$ is defined differentiable in a neighborhood of a point \mathbf{x}_0 , then $f(\mathbf{x})$ decreases "fastest" if one goes from \mathbf{x}_0 in the direction of the negative gradient of f at \mathbf{x}_0 , $-\nabla f(\mathbf{x}_0)$.

Gradient descent (2)



- It follows that, if

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \nabla f(\mathbf{x}_n) \quad (1)$$

for α small enough, then $f(\mathbf{x}_n) \geq f(\mathbf{x}_{n+1})$. In other words, the term $\alpha \nabla f(\mathbf{x})$ is subtracted from \mathbf{x} because we want to move against the gradient, namely down toward the minimum.

- How can we choose α ?
- The Rosenbrock function $f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$. has a narrow curved valley which contains the minimum. The bottom of the valley is very flat. Because of the curved flat valley the optimization is zig-zagging slowly with small stepsizes towards the minimum.

Steepest descent using line search

- **Inexact line search** consists in finding $\alpha_k \simeq \underset{\alpha \in \mathbb{R}_+}{\operatorname{argmin}} f(\mathbf{x}_k + \alpha \mathbf{d}_k)$
- The **line search** method is one of two basic iterative approaches to find a local minimum \mathbf{x}^* of an objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ using gradients. The other approach is **trust region**.

Algorithm 1 Steepest descent with line search

```
1: procedure STEEPEST DESCENT( $f, \mathbf{x}$ )  
2:    $k = 0, \mathbf{x}_0$                                 ▷ Iteration counter + initial parameter guess  
3:   while  $\|\nabla f(\mathbf{x}_k)\| > \epsilon$  do                ▷  $\epsilon =$  tolerance  
4:      $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$                         ▷ Descent direction = Steepest descent  
5:      $\alpha_k \simeq \underset{\alpha \in \mathbb{R}_+}{\operatorname{argmin}} f(\mathbf{x}_k + \alpha \mathbf{d}_k)$     ▷ Line search  
6:      $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$   
7:      $k = k + 1$   
8:   end while  
9: end procedure
```

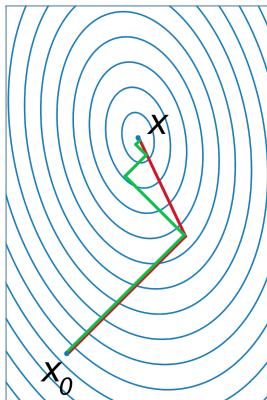
Nonlinear conjugate gradient methods

- Let's pose $\mathbf{g}_k = \nabla f(x_k)$
- The steepest descent direction was $\mathbf{d}_k = -\mathbf{g}_k$
- Conjugate directions differs from the steepest descent by attempting moves based on the history of the previous moves
- The descent direction for nonlinear conjugate gradient methods is

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k, \quad \mathbf{d}_0 = -\mathbf{g}_0 \quad (2)$$

- The variation of the gradient is measured by $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$
- The Conjugate Gradient update parameter β_k can be updated with different formulas

Conjugate gradient: convergence



- A comparison of the linear convergence of simple gradient descent with optimal step size (in green) and the superlinear convergence of conjugate gradient (in red) for minimizing a quadratic function.

Nonlinear conjugate gradient methods

$$\beta_k^{HS} = \frac{\mathbf{g}_{k+1}^\top \mathbf{y}_k}{\mathbf{d}_k^\top \mathbf{y}_k}$$

(1952) in the original (linear) CG paper
of Hestenes and Stiefel [59]

$$\beta_k^{FR} = \frac{\|\mathbf{g}_{k+1}\|^2}{\|\mathbf{g}_k\|^2}$$

(1964) first nonlinear CG method, proposed
by Fletcher and Reeves [45]

$$\beta_k^D = \frac{\mathbf{g}_{k+1}^\top \nabla^2 f(\mathbf{x}_k) \mathbf{d}_k}{\mathbf{d}_k^\top \nabla^2 f(\mathbf{x}_k) \mathbf{d}_k}$$

(1967) proposed by Daniel [39], requires
evaluation of the Hessian $\nabla^2 f(\mathbf{x})$

$$\beta_k^{PRP} = \frac{\mathbf{g}_{k+1}^\top \mathbf{y}_k}{\|\mathbf{g}_k\|^2}$$

(1969) proposed by Polak and Ribière [84]
and by Polyak [85]

$$\beta_k^{CD} = \frac{\|\mathbf{g}_{k+1}\|^2}{-\mathbf{d}_k^\top \mathbf{g}_k}$$

(1987) proposed by Fletcher [44], CD
stands for “Conjugate Descent”

$$\beta_k^{LS} = \frac{\mathbf{g}_{k+1}^\top \mathbf{y}_k}{-\mathbf{d}_k^\top \mathbf{g}_k}$$

(1991) proposed by Liu and Storey [67]

$$\beta_k^{DY} = \frac{\|\mathbf{g}_{k+1}\|^2}{\mathbf{d}_k^\top \mathbf{y}_k}$$

(1999) proposed by Dai and Yuan [27]

$$\beta_k^N = \left(\mathbf{y}_k - 2\mathbf{d}_k \frac{\|\mathbf{y}_k\|^2}{\mathbf{d}_k^\top \mathbf{y}_k} \right)^\top \frac{\mathbf{g}_{k+1}}{\mathbf{d}_k^\top \mathbf{y}_k}$$

(2005) proposed by Hager and Zhang [53]

Newton optimization method

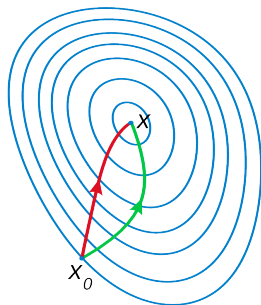


Figure: A comparison of gradient descent (green) and Newton's method (red) for minimizing a function (with small step sizes). Newton's method uses curvature information to take a more direct route.

- Hessian is used to exploit the curvature information

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha [\mathbf{H}f(\mathbf{x}_n)]^{-1} \nabla f(\mathbf{x}_n) \quad (3)$$

- $\alpha \in (0, 1)$, with $\alpha = 1$ the exact form.

Newton-Raphson root-finding and eccentric anomaly

- Newton optimization method and Newton-Raphson's root finding methods are based on similar principles
- Newton-Raphson: $x_{n+1} = x_n - f(x_n)/f'(x_n)$
- Example: the mean anomaly is proportional to time it is an easily measured quantity for an orbiting body. Given the mean anomaly M , find the eccentric anomaly E and the orbital eccentricity e with Kepler's Equation:

$$M = E - e \sin E \quad (4)$$

Better than Newton: quasi-Newton methods

- Also known as variable metric methods, they avoid computing the Hessian then its inverse.
- The Broyden-Fletcher-Goldfarb-Shanno (BFGS) or Davidon-Fletcher-Powell (DFP) algorithms build iteratively approximations of $[\mathbf{H}f(\mathbf{x}_n)]^{-1}$.
- The most successful and well-known quasi-Newton method is the **Limited-memory BFGS (L-BFGS)** that approximates $[\mathbf{H}f(\mathbf{x}_n)]^{-1}\nabla f(\mathbf{x}_n)$ directly and thus can work on large scale problems (millions of variables).
- New gradient descent variants attempt to deal with non-smooth functions (subgradient and bundle method).
- There are variants that deal with constrained minimization (i.e. bounds on variables or linearly tied variables) such as L-BFGS-B. Further refinements led to the VMLM algorithm in OptimPack.

Trust-region method and Levenberg-Marquardt

- Consider the quadratic approximation of function f around x_0 :

$$q(\epsilon) \simeq f(x_0) + \nabla f(x_0)\epsilon + \frac{1}{2}\epsilon^T \nabla^2 f(x_0)\epsilon \quad (5)$$

- $q(\epsilon)$ has a close-form minimum.
- $q(\epsilon)$ remains a good approximation within a given radius, $\|e\|_2 < r^2$ defines the **trust region** radius r .
- The quadratic approximation predicts a certain reduction in the cost function, Δf_{pred} , which is compared to the true reduction $\Delta f_{\text{actual}} = f(x) - f(x + \epsilon)$. By looking at the ratio $\Delta f_{\text{pred}}/\Delta f_{\text{actual}}$ we can estimate the trust-region size at each iteration, jump to the closed-form minimum within the trust region, and iterate.
- The **Levenberg-Marquardt** algorithm (first published in 1944 by Kenneth Levenberg, rediscovered in 1963 by Donald Marquardt) uses the trust-region approach with conjugate-gradients and Gauss-Newton (Newton optimized for non-linear χ^2). Like conjugate gradient and Newton, these local optimization.

Derivative-free optimization methods

- **Derivative-free** optimization methods simply do not require gradient information
- Among the most popular local optimizer is **Nelder–Mead** method (aka downhill simplex method or amoeba method), which moves points of a polytope of $n + 1$ vertices in n -parameter dimensions via reflection, contraction, expansion steps
- Most MCMC optimization methods.
- The NLOpt library provides mostly derivative-free algorithms, some of them for global optimization.

Constrained minimization: the Lagrangian method

- **Constrained** minimization is minimization under equality or inequality constraints. **Bounded** optimization is a special case of constrained optimization where bounds are imposed on parameters (e.g. positivity, or variable within a range). In Bayesian terms, we're imposing a prior.
- A classic example is:

$$(\tilde{x}, \tilde{y}) = \underset{(x,y) \in \mathbb{R}^2}{\operatorname{argmin}} (x + y) \quad \text{s. t. } x^2 + y^2 = 1$$

- We pose $g(x, y) = x^2 + y^2 - 1$ and the Lagrangian is:

$$\begin{aligned} \mathcal{L}(x, y, \lambda) &= f(x, y) + \lambda \cdot g(x, y) \\ &= x + y + \lambda(x^2 + y^2 - 1). \end{aligned}$$

where λ is a Lagrange multiplier.

Constrained minimization: the Lagrangian method

- The gradient with respect to variables x, y and λ

$$\begin{aligned}\nabla_{x,y,\lambda}\mathcal{L}(x,y,\lambda) &= \left(\frac{\partial\mathcal{L}}{\partial x}, \frac{\partial\mathcal{L}}{\partial y}, \frac{\partial\mathcal{L}}{\partial\lambda}\right) \\ &= (1 + 2\lambda x, 1 + 2\lambda y, x^2 + y^2 - 1)\end{aligned}$$

and therefore:

$$\nabla_{x,y,\lambda}\mathcal{L}(x,y,\lambda) = 0 \quad \Leftrightarrow \quad \begin{cases} 1 + 2\lambda x = 0 \\ 1 + 2\lambda y = 0 \\ x^2 + y^2 - 1 = 0 \end{cases}$$

- Solution $x = y = -\frac{1}{2\lambda}$, $\lambda \neq 0$. Substituting into the last equation we get $\lambda = \pm \frac{1}{\sqrt{2}}$ which implies that the stationary points of \mathcal{L} are $\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, -\frac{1}{\sqrt{2}}\right)$, $\left(-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, \frac{1}{\sqrt{2}}\right)$. And since $f\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right) = \sqrt{2}$ and $f\left(-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right) = -\sqrt{2}$, the solution is found.

Constrained minimization: Half-quadratic splitting (1)

- Let's say we want to minimize:

$$\tilde{\mathbf{x}} = \operatorname{argmin}_{\mathbf{x}} \frac{1}{2} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \mu\Phi(\mathbf{x})$$

where $\Phi(x)$ is a function such that we wouldn't be able to solve this via Tikhonov. We saw a good example is $\Phi(\mathbf{x}) = \ell_1(\mathbf{x})$.

- Splitting methods are methods that split the unconstrained problem into a constrained problem, using two different variables to represent the same one in different functions:

$$\begin{aligned}\tilde{\mathbf{x}} &= \operatorname{argmin}_{\mathbf{x}} \frac{1}{2} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \mu\Phi(\mathbf{z}) \quad \text{s.t.} \quad \mathbf{z} = \mathbf{x} \\ &= \operatorname{argmin}_{\mathbf{x}, \mathbf{z}} \frac{1}{2} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \mu\Phi(\mathbf{z}) + \frac{\rho}{2} \|\mathbf{z} - \mathbf{x}\|_2^2\end{aligned}$$

Constrained minimization: the two subproblems

- So we now want to minimize:

$$\mathcal{L}(\mathbf{x}, \mathbf{z}) = \frac{1}{2} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \mu\Phi(\mathbf{z}) + \frac{\rho}{2} \|\mathbf{z} - \mathbf{x}\|_2^2$$

- $\frac{\rho}{2} \|\mathbf{z} - \mathbf{x}\|_2^2$ is called an augmented term, and ρ the augmented penalty or hyperparameter.
- The **half-quadratic splitting method** solves iteratively (iteration variable = k) the problem with respect to \mathbf{x} , then \mathbf{z} :

$$\tilde{\mathbf{x}}^{k+1} = \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \frac{\rho}{2} \|\tilde{\mathbf{z}}^k - \mathbf{x}\|_2^2 \quad \mathbf{x} \text{ sub-problem}$$

$$\tilde{\mathbf{z}}^{k+1} = \underset{\mathbf{z}}{\operatorname{argmin}} \frac{\rho}{2} \|\mathbf{z} - \tilde{\mathbf{x}}^{k+1}\|_2^2 + \mu\Phi(\mathbf{z}) \quad \mathbf{z} \text{ sub-problem}$$

and then increases ρ from initially low values to higher and higher ones.

Constrained minimization: analytical solutions exist

- Why is this easier than the original problem ?

$$\tilde{\mathbf{x}}^{k+1} = \operatorname{argmin}_{\mathbf{x}} \frac{1}{2} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \frac{\rho}{2} \|\tilde{\mathbf{z}}^k - \mathbf{x}\|_2^2 \quad \mathbf{x} \text{ sub-problem}$$

$$\tilde{\mathbf{z}}^{k+1} = \operatorname{argmin}_{\mathbf{z}} \frac{\rho}{2} \|\mathbf{z} - \tilde{\mathbf{x}}^{k+1}\|_2^2 + \mu \Phi(\mathbf{z}) \quad \mathbf{z} \text{ sub-problem}$$

- The \mathbf{x} sub-problem can be solved by Tikhonov.
- The \mathbf{z} sub-problem can be solved analytically for some functions, for which we know the solution of the problem:

$$\operatorname{prox}_f(\mathbf{z}) = \operatorname{argmin}_{\mathbf{w}} \left(\frac{1}{2} \|\mathbf{z} - \mathbf{w}\|_2^2 + f(\mathbf{w}) \right)$$

- This solution $\operatorname{prox}_f(\mathbf{z})$ is the **proximal operator** for the function f . At each point \mathbf{z} it finds a close-by local minimum of f .

Proximal operator for the ℓ_1 norm and positivity

- One can demonstrate that the proximal operator for ℓ_1 norm is:

$$\text{prox}_{\alpha\ell_1}(\mathbf{z}) = \underset{\mathbf{w}}{\operatorname{argmin}} \left(\frac{1}{2} \|\mathbf{z} - \mathbf{w}\|_2^2 + \alpha\ell_1(\mathbf{w}) \right) = \text{sign}(\mathbf{z}) \cdot \max(|\mathbf{z}| - \alpha, 0)$$

where \cdot is the Hadamard product.

- The proximal operator for positivity is the projection onto the positive set:

$$\text{prox}_{I_{\mathbb{R}^+}}(\mathbf{z}) = \underset{\mathbf{w} \in \mathbb{R}^+}{\operatorname{argmin}} \left(\frac{1}{2} \|\mathbf{z} - \mathbf{w}\|_2^2 \right) = \max(\mathbf{z}, 0)$$

Half-quadratic splitting: beyond the 1:1 change of variable

- Half-quadratic splitting only involves analytical steps: the \mathbf{x} sub-problem is solved via Tikhonov and the \mathbf{z} sub-problem via proximal operators (provided it is known). Generalizing beyond $\mathbf{z} = \mathbf{x}$, we can also have more complex linear constraints under the form $\mathbf{Ax} + \mathbf{Bz} + \mathbf{c} = 0$.
- How should we solve the total variation problem, i.e. minimize $\frac{1}{2} \|\mathbf{Hx} - \mathbf{y}\|_2^2 + \mu \ell_1(\nabla \mathbf{x})$?
- We only know the proximal operator for $\ell_1(\mathbf{z})$ and not for $\ell_1(\nabla \mathbf{z})$. So we should not pose $\mathbf{z} = \mathbf{x}$, since we wouldn't know how to solve the \mathbf{z} sub-problem. However we can pose $\mathbf{z} = \nabla \mathbf{x}$, leading to:

$$\mathcal{L}(\mathbf{x}, \mathbf{z}) = \frac{1}{2} \|\mathbf{Hx} - \mathbf{y}\|_2^2 + \mu \ell_1(\mathbf{z}) + \frac{\rho}{2} \|\mathbf{z} - \nabla \mathbf{x}\|_2^2$$

Total variation solved via Half-quadratic splitting

$$\mathcal{L}(\mathbf{x}, \mathbf{z}) = \frac{1}{2} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \mu \ell_1(\mathbf{z}) + \frac{\rho}{2} \|\mathbf{z} - \nabla \mathbf{x}\|_2^2$$

- The \mathbf{x} sub-problem is:

$$\tilde{\mathbf{x}}^{k+1} = \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \frac{\rho}{2} \|\tilde{\mathbf{z}}^k - \nabla \mathbf{x}\|_2^2 \quad \mathbf{x} \text{ sub-problem}$$

$$\implies \mathbf{H}^\top (\mathbf{H}\mathbf{x} - \mathbf{y}) - \rho \nabla^\top (\tilde{\mathbf{z}}^k - \nabla \mathbf{x}) = 0$$

$$\implies \tilde{\mathbf{x}}^{k+1} = (\mathbf{H}^\top \mathbf{H} + \rho \nabla^\top \nabla)^{-1} (\mathbf{H}^\top \mathbf{y} + \rho \nabla^\top \tilde{\mathbf{z}}^k)$$

- The \mathbf{z} sub-problem is:

$$\tilde{\mathbf{z}}^{k+1} = \underset{\mathbf{z}}{\operatorname{argmin}} \mu \ell_1(\mathbf{z}) + \frac{\rho}{2} \|\mathbf{z} - \nabla \tilde{\mathbf{x}}^{k+1}\|_2^2 \quad \mathbf{z} \text{ sub-problem}$$

$$= \operatorname{prox}_{\frac{\mu}{\rho} \ell_1}(\nabla \tilde{\mathbf{x}}^{k+1}) = \operatorname{sign}(\nabla \tilde{\mathbf{x}}^{k+1}) \cdot \max(|\nabla \tilde{\mathbf{x}}^{k+1}| - \frac{\mu}{\rho}, 0)$$

Making the \mathbf{x} step faster: some tricks

- The \mathbf{x} sub-problem is the slowest one since it involves:

$$\tilde{\mathbf{x}}^{k+1} = (\mathbf{H}^\top \mathbf{H} + \rho \nabla^\top \nabla)^{-1} (\mathbf{H}^\top \mathbf{y} + \rho \nabla^\top \tilde{\mathbf{z}}^k)$$

but matrix inversion is $\sim \mathcal{O}(N^3)$ process for a N matrix, and thus costly both in memory space and computing time. To work with larger images, tricks to speed up the inversion are used in practice.

- One is to use **sparse arrays** that only store the non-zero elements of \mathbf{H} and ∇ . The inversion and subsequent multiplication are then faster.
- Using an **orthogonal wavelet basis**, \mathbf{W} , as a sparsity basis instead of the spatial gradient ∇ since in this case $\mathbf{W}^\top \mathbf{W} = \alpha \mathbf{I}$.
- Another is to employ the backslash operator in Julia (or Matlab). We have $\mathbf{X}^{-1} \mathbf{Y} = \mathbf{X} \backslash \mathbf{Y}$, but the latter operation doesn't store the inverted matrix \mathbf{X}^{-1} ; instead it just temporarily stores the parts useful for its multiplication by \mathbf{Y} .

$$\tilde{\mathbf{x}}^{k+1} = (\mathbf{H}^\top \mathbf{H} + \rho \nabla^\top \nabla) \backslash (\mathbf{H}^\top \mathbf{y} + \rho \nabla^\top \tilde{\mathbf{z}}^k)$$

Making the x step faster: circulant matrices (1)

- It turns out $\nabla^\top \nabla$ and $\mathbf{H}^\top \mathbf{H}$ can be simplified if ∇ and \mathbf{H} are circulant matrices, or concatenation of circulant matrices. This is the case when ∇ is the spatial gradient, and when \mathbf{H} is a convolution (= if modeling the imaging done by an optical system).
- A circulant matrix \mathbf{C} takes the form:

$$\mathbf{C} = \begin{bmatrix} c_1 & c_n & \dots & c_3 & c_2 \\ c_2 & c_1 & c_n & & c_3 \\ \vdots & c_2 & c_1 & \ddots & \vdots \\ c_{n-1} & \vdots & \ddots & \ddots & c_n \\ c_n & c_{n-1} & \dots & c_2 & c_1 \end{bmatrix}.$$

- All circulant matrixes can be written as $\mathbf{C} = \mathbf{F} \mathbf{D} \mathbf{F}^\top$ where \mathbf{F} is the Fourier transform ($n^2 \times n^2$ if implemented via matrix operations) and $\mathbf{D} = \text{diag}(\mathbf{F}([c_1 \dots c_n]))$ is diagonal.

Making the x step faster: circulant matrices (2)

- We remind that $\mathbf{F}^\top \mathbf{F} = \mathbf{I}$ since $\mathbf{F}^\top = \mathbf{F}^{-1}$ and $(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top$
- Since $\mathbf{C} = \mathbf{FDF}^\top$, $\mathbf{C}^\top = ((\mathbf{FD})\mathbf{F}^\top)^\top = \mathbf{F}(\mathbf{FD})^\top = \mathbf{FD}^\top \mathbf{F}^\top$
- Consequently $\mathbf{C}^\top \mathbf{C} = \mathbf{FDF}^\top \mathbf{FD}^\top \mathbf{F}^\top = \mathbf{FD}^2 \mathbf{F}^\top$ since \mathbf{D} is diagonal.
- The application relies on the fact that applying the Fourier transforms \mathbf{F} or \mathbf{F}^{-1} can be done with very fast FFT algorithms:

$$\begin{aligned}(\mathbf{H}^\top \mathbf{H} + \rho \nabla^\top \nabla) \tilde{\mathbf{x}}^{k+1} &= \mathbf{H}^\top \mathbf{y} + \rho \nabla^\top \tilde{\mathbf{z}}^k \\(\mathbf{FDH}^2 \mathbf{F}^\top + \rho \mathbf{FD} \nabla^2 \mathbf{F}^\top) \tilde{\mathbf{x}}^{k+1} &= \mathbf{H}^\top \mathbf{y} + \rho \nabla^\top \tilde{\mathbf{z}}^k \\ \mathbf{F}(\mathbf{D_H}^2 + \rho \mathbf{D_\nabla}^2) \mathbf{F}^\top \tilde{\mathbf{x}}^{k+1} &= \mathbf{H}^\top \mathbf{y} + \rho \nabla^\top \tilde{\mathbf{z}}^k \\ (\mathbf{D_H}^2 + \rho \mathbf{D_\nabla}^2) \mathbf{F}^\top \tilde{\mathbf{x}}^{k+1} &= \mathbf{F}^{-1}(\mathbf{H}^\top \mathbf{y} + \rho \nabla^\top \tilde{\mathbf{z}}^k) \\ \mathbf{F}^\top \tilde{\mathbf{x}}^{k+1} &= \frac{\mathbf{F}^{-1}(\mathbf{H}^\top \mathbf{y} + \rho \nabla^\top \tilde{\mathbf{z}}^k)}{\mathbf{D_H}^2 + \rho \mathbf{D_\nabla}^2} \\ \tilde{\mathbf{x}}^{k+1} &= \mathbf{F} \left(\frac{\mathbf{F}^{-1}(\mathbf{H}^\top \mathbf{y} + \rho \nabla^\top \tilde{\mathbf{z}}^k)}{\mathbf{D_H}^2 + \rho \mathbf{D_\nabla}^2} \right)\end{aligned}$$

Augmented Lagrangian methods

- Let's say we want to minimize $f(\mathbf{x})$ under a sets of constraints on \mathbf{x} . We saw we could express each of these constrains as $g_i(\mathbf{x}) = 0$. Then we just need to minimize the Lagrangian:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_i \lambda_i \cdot g_i(\mathbf{x})$$

where $\boldsymbol{\lambda}$ is a vector this time.

- The idea is to use an **augmented Lagrangian**:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_i \lambda_i \cdot g_i(\mathbf{x}) + \frac{\rho}{2} \sum_i |g_i(\mathbf{x})|^2$$

where the augmentation term $\sum_i |g_i(\mathbf{x})|^2$ is multiplied by penalty terms ρ .

- The **method of multipliers** solves this by iterating:

$$\mathbf{x}^{k+1} = \underset{\mathbf{x}}{\operatorname{argmin}} \mathcal{L}(\mathbf{x}, \lambda_i^k)$$

$$\lambda_i^{k+1} = \lambda_i^k + \rho^k g_i(\mathbf{x}^k) \quad \forall i$$

Example: Method of multipliers for linear constraints

- Let's say we want to minimize $f(\mathbf{x})$ under a sets of linear constraints on \mathbf{x} , we can express them as $\mathbf{Ax} = \mathbf{b}$.
- The augmented Lagrangian is:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^\top (\mathbf{Ax} - \mathbf{b}) + \frac{\rho}{2} \|\mathbf{Ax} - \mathbf{b}\|_2^2$$

- The method of multipliers solves this by iterating:

$$\mathbf{x}^{k+1} = \underset{\mathbf{x}}{\operatorname{argmin}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}^k)$$

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \rho(\mathbf{Ax}^{k+1} - \mathbf{b})$$

Alternating Direction Method of Multipliers (ADMM)

- Putting all our previous knowledge together, using variable splitting, Lagrangian multipliers, and augmentation terms, we come with ADMM, a method to solve:

$$\operatorname{argmin}_{\mathbf{x}, \mathbf{z}} f(\mathbf{x}) + g(\mathbf{z}) \quad \text{s.t. } \mathbf{Ax} + \mathbf{Bz} + \mathbf{c} = \mathbf{0}$$

- ADMM solves this by posing:

$$\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\lambda}) = f(\mathbf{x}) + g(\mathbf{z}) + \boldsymbol{\lambda}^\top (\mathbf{Ax} + \mathbf{Bz} + \mathbf{c}) + \frac{\rho}{2} \|\mathbf{Ax} + \mathbf{Bz} + \mathbf{c}\|_2^2$$

- And then we iterate:

$$\mathbf{x}^{k+1} = \operatorname{argmin}_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \mathbf{z}^k, \boldsymbol{\lambda}^k)$$

$$\mathbf{z}^{k+1} = \operatorname{argmin}_{\mathbf{z}} \mathcal{L}(\mathbf{x}^{k+1}, \mathbf{z}, \boldsymbol{\lambda}^k)$$

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \rho(\mathbf{Ax}^{k+1} + \mathbf{Bz}^{k+1} + \mathbf{c})$$

Why use ADMM ?

- Since ADMM is solving:

$$\operatorname{argmin}_{\mathbf{x}, \mathbf{z}} f(\mathbf{x}) + g(\mathbf{z}) \quad \text{s.t. } \mathbf{Ax} + \mathbf{Bz} + \mathbf{c} = 0,$$

we use it for the solution of Bayesian problems where f is the log-likelihood and g a regularization function or prior (e.g. positivity).

- More regularization terms can be added $f(\mathbf{x}) + g(\mathbf{z}) + h(\mathbf{w}) + \dots$
- If the log-likelihood can be split (by wavelengths, times, etc.) such as is often the case with χ^2 then ADMM allows for efficient parallelization of large-scale problems. In this case we go from $f(\mathbf{x})$ to $f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2) + \dots$ so that $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots\}$ and f_1, f_2, \dots can be minimized in parallel.