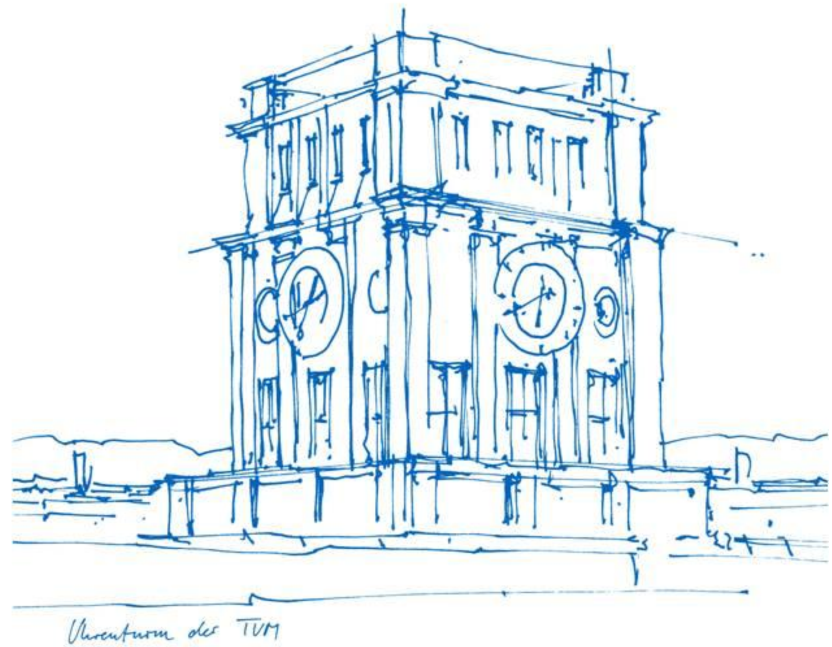
Practical Course | Applied Optimization Methods for Inverse Problems



Student
Kaan Güney Keklikçi

Instructor(s)
PD Dr. Tobias Lasser
David Frank, MSc

Agenda

- Overview
- Fast Proximal Gradient Method (FPGM)
- Barzilai & Borwein (BB)
  - BB1
  - BB2
- Takeaways
- Discussion

Introduction

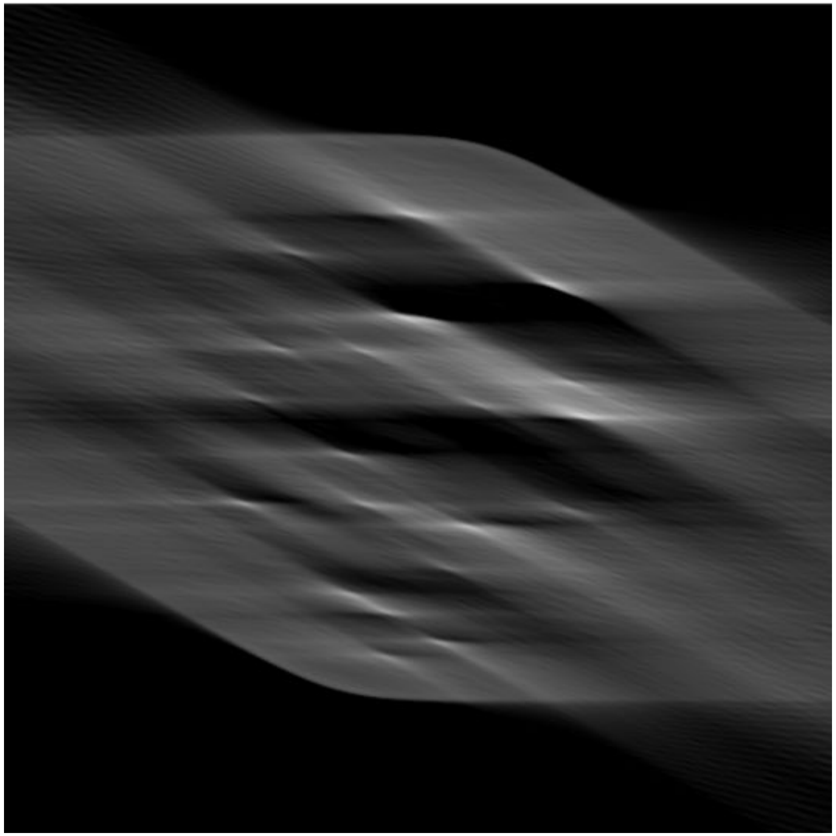
- Quantitative assessment of top 2 optimization methods ranked by performance on the challenge dataset
- Most successful algorithm
  - Fast Proximal Gradient Method (FPGM)
- Identification of most important parameters, i.e., regularization
  - If dynamically computed, not highly important
  - For instance, momentum calculations and use of Lipschitz constant
- Non-negativity constraint
  - Eliminates streaky lines and reduces overall noise

Fast Proximal Gradient Method (FPGM)

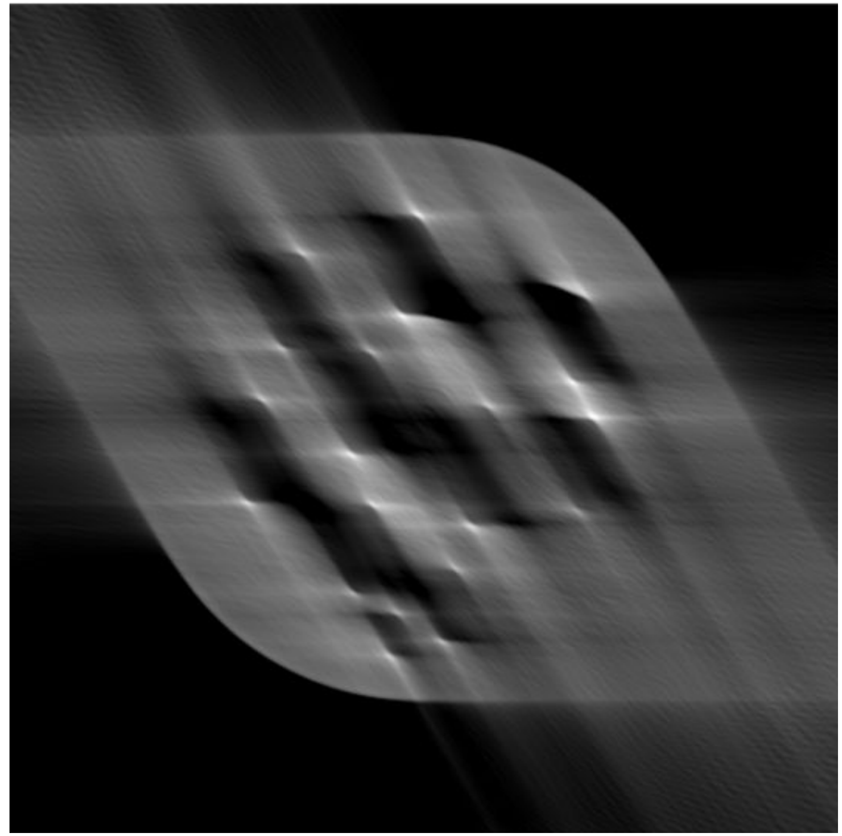
Phantom	Angle	Score*	Regularization (λ)
C	360	0.998532	$\lambda = 1.0$
C	90	0.669394	$\lambda = 1.0$
C	60	0.523257	$\lambda = 1.0$
C	30	0.457177	$\lambda = 1.0$
A	360	0.998875	$\lambda = 1.0$
A	90	0.743667	$\lambda = 1.0$
A	60	0.544239	$\lambda = 1.0$
A	30	0.419584	$\lambda = 1.0$
B	360	0.998647	$\lambda = 1.0$
B	90	0.716096	$\lambda = 1.0$
B	60	0.536802	$\lambda = 1.0$
B	30	0.404619	$\lambda = 1.0$

* reconstruction score on the server

Reconstructions via Challenge Dataset (A)

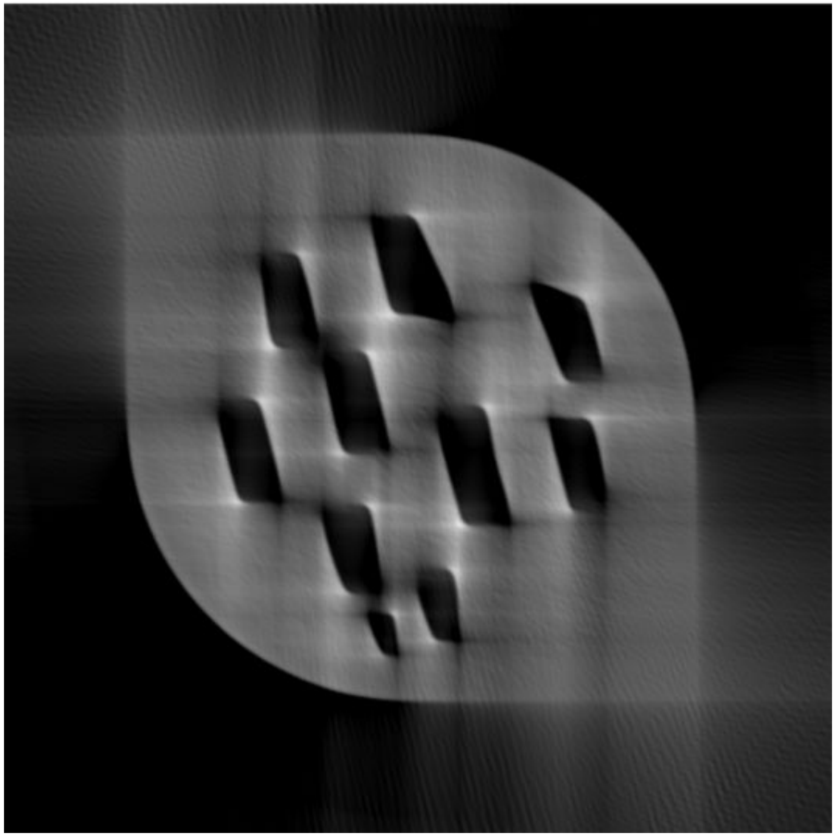


Score: 0.419584 | λ = 1.0 | θ = 30

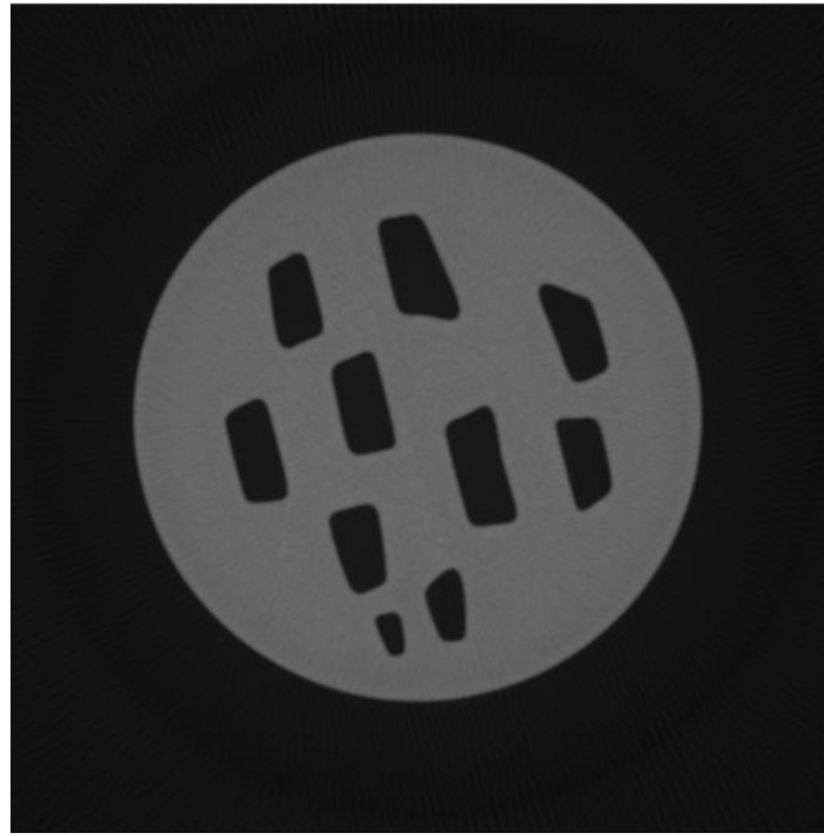


Score: 0.544239 | λ = 1.0 | θ = 60

Reconstructions via Challenge Dataset (A)

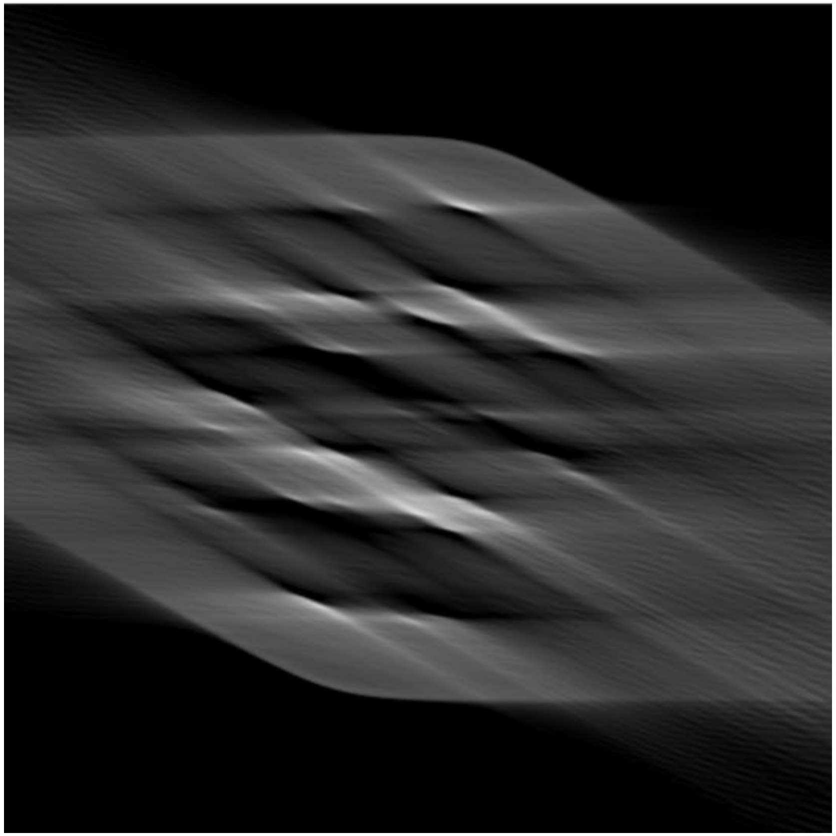


Score: 0.743667 | λ = 1.0 | θ = 90

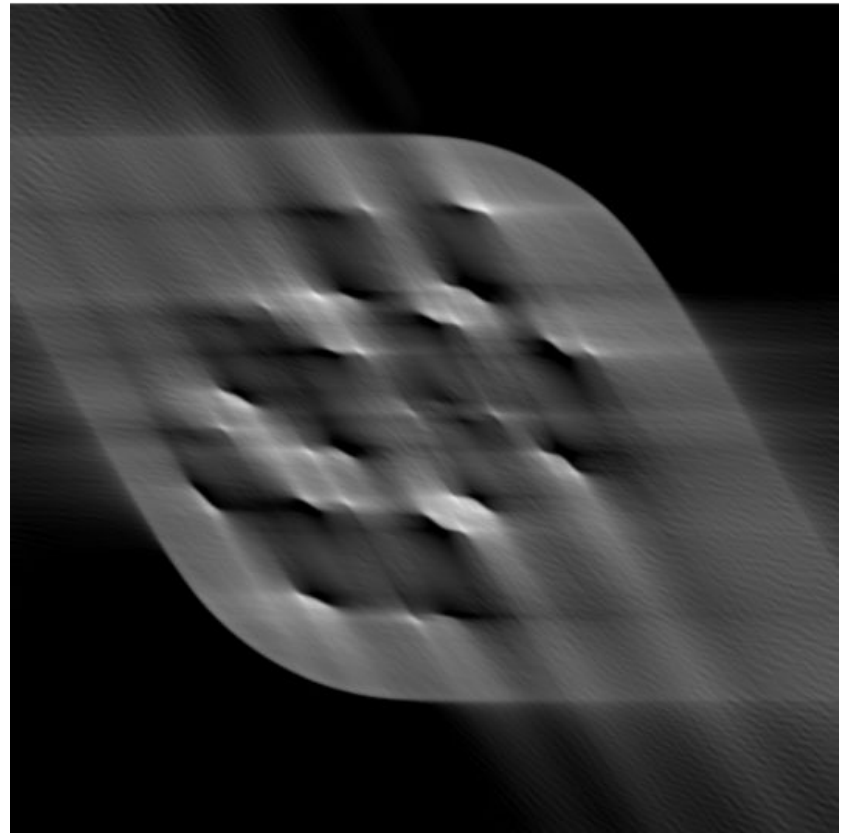


Score: 0.998875 | λ = 1.0 | θ = 360

Reconstructions via Challenge Dataset (B)

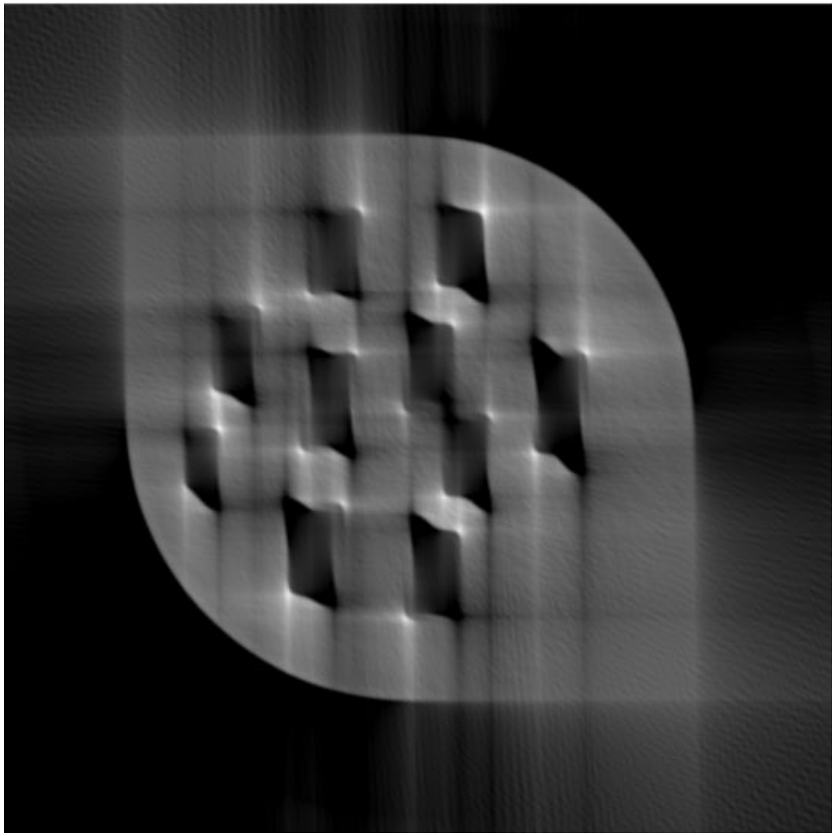


Score: 0.404619 | λ = 1.0 | θ = 30

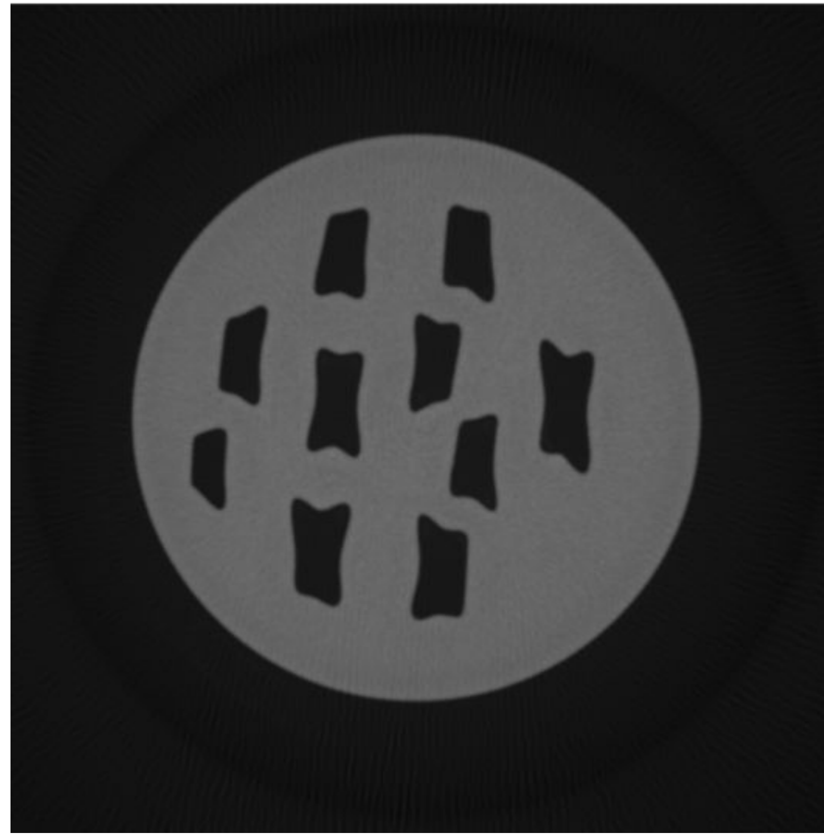


Score: 0.536802 | λ = 1.0 | θ = 60

Reconstructions via Challenge Dataset (B)

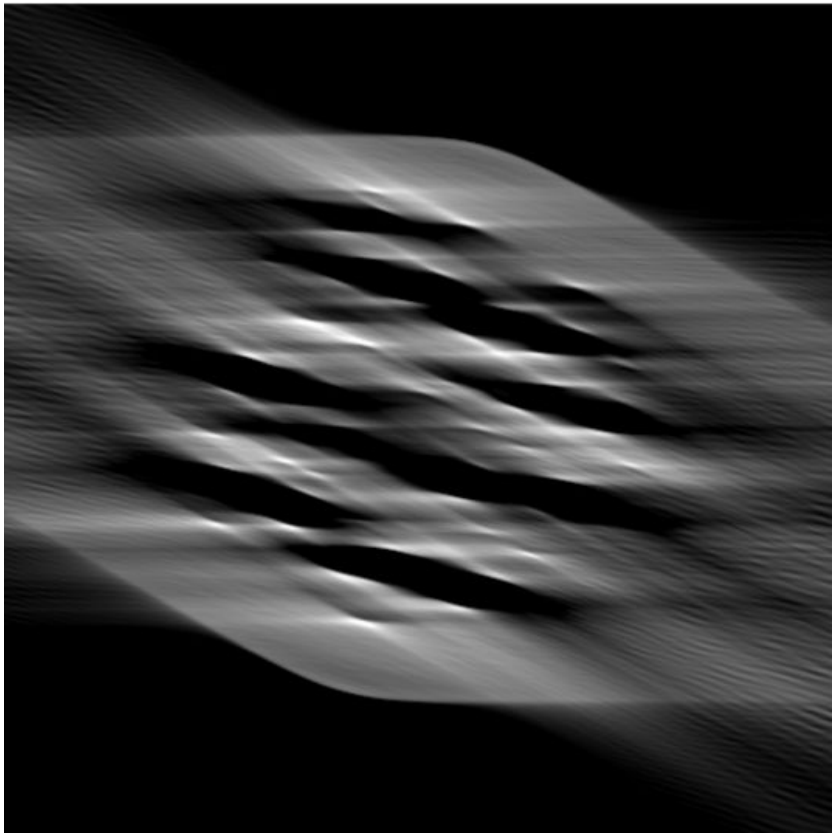


Score: 0.716096 | λ = 1.0 | θ = 90

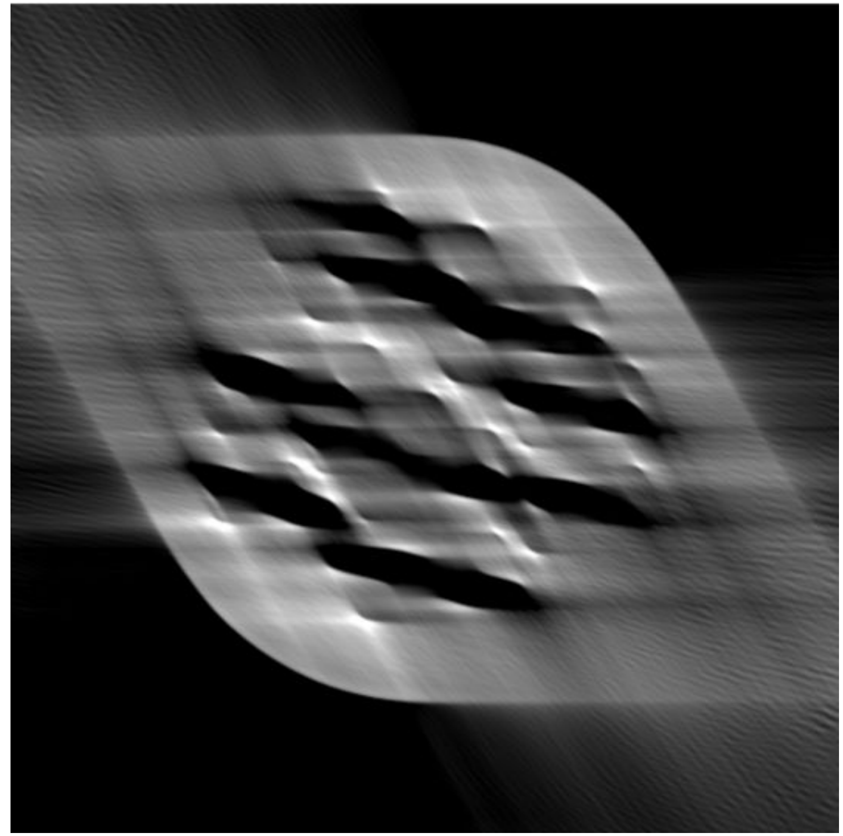


Score: 0.998647 | λ = 1.0 | θ = 360

Reconstructions via Challenge Dataset (C)

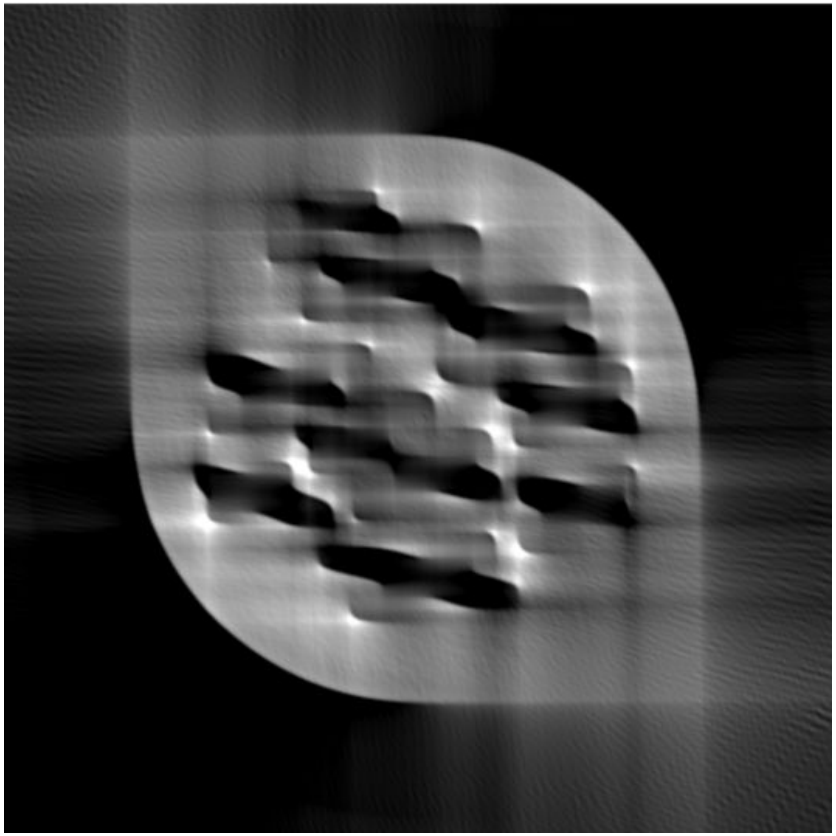


Score: 0.457177 | λ = 1.0 | θ = 30

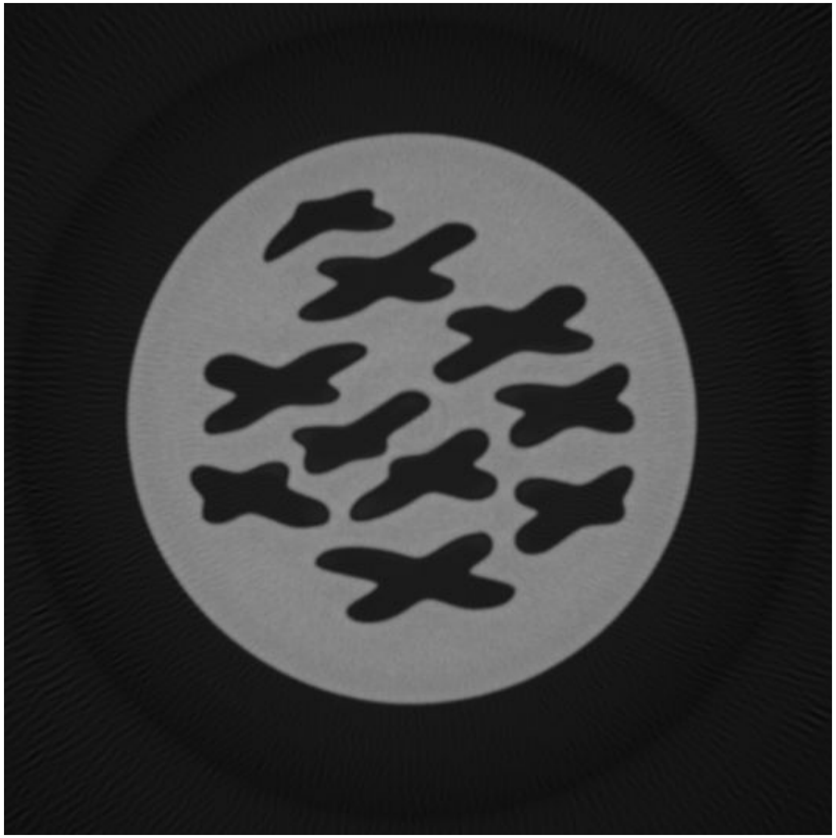


Score: 0.523257 | λ = 1.0 | θ = 60

Reconstructions via Challenge Dataset (C)



Score: 0.669394 | λ = 1.0 | θ = 90



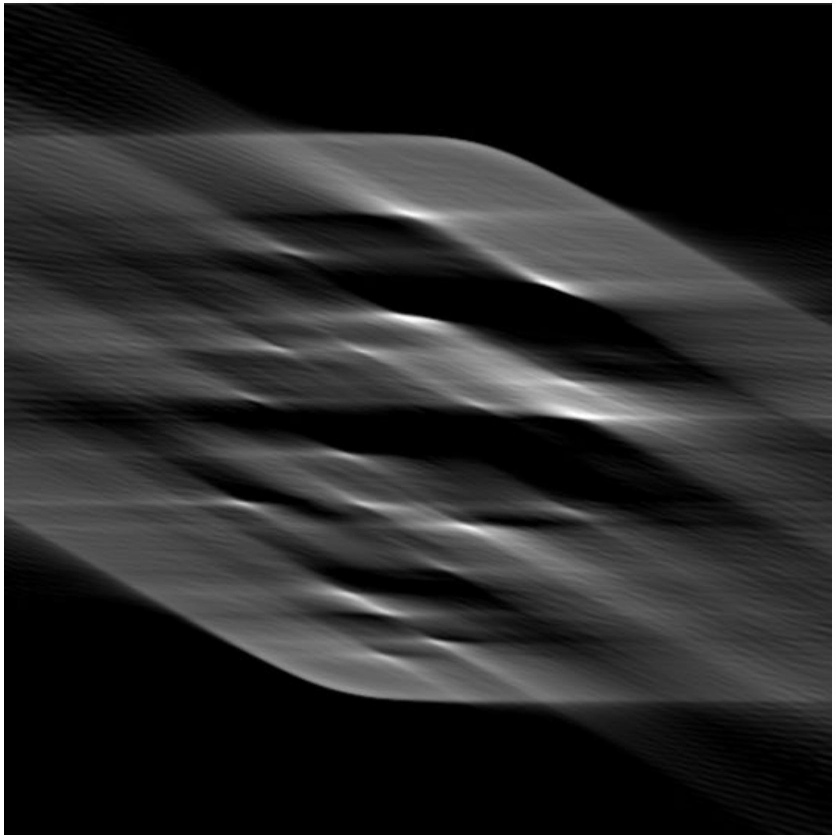
Score: 0.998532 | λ = 1.0 | θ = 360

Barzilai & Borwein

Phantom	Angle	Score*	Regularization (λ)
C	360	0.994723	λ = 0.0001
C	90	0.688492	λ = 0.0001
C	60	0.436286	λ = 0.0001
C	30	0.492479	λ = 0.0001
A	360	0.998964	λ = 0.0001
A	90	0.775538	λ = 0.0001
A	60	0.541325	λ = 0.0001
A	30	0.378975	λ = 0.0001
B	360	0.998853	λ = 0.0001
B	90	0.762010	λ = 0.0001
B	60	0.535358	λ = 0.0001
B	30	0.341525	λ = 0.0001

* reconstruction score on the server

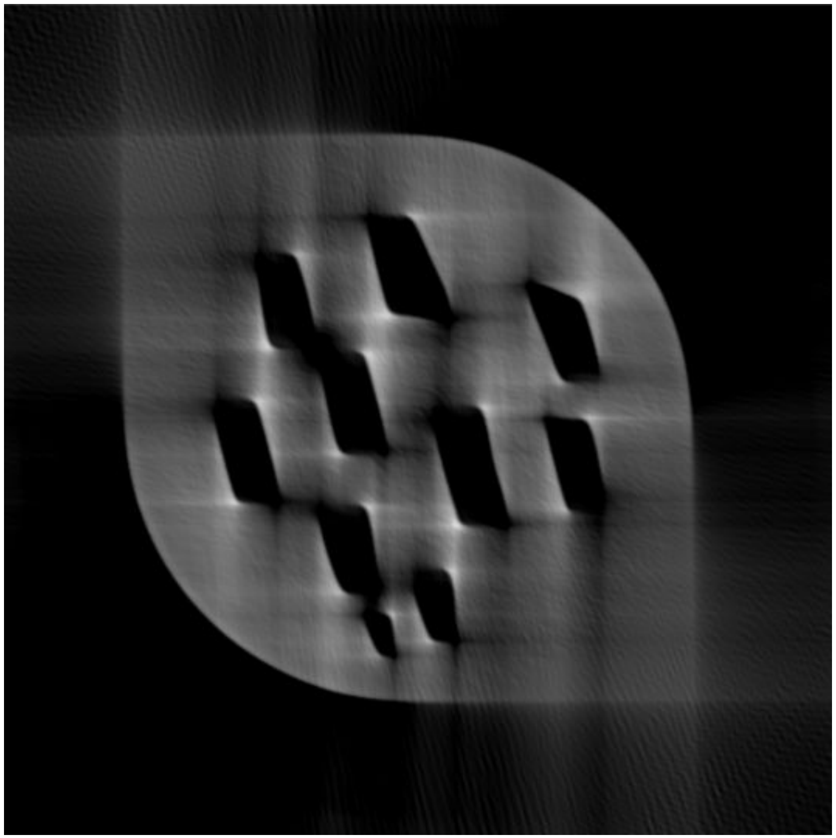
Reconstructions via Challenge Dataset (A)



Score: 0.378975 | λ = 1.0 | θ = 30

Score: 0.541325 | λ = 1.0 | θ = 60

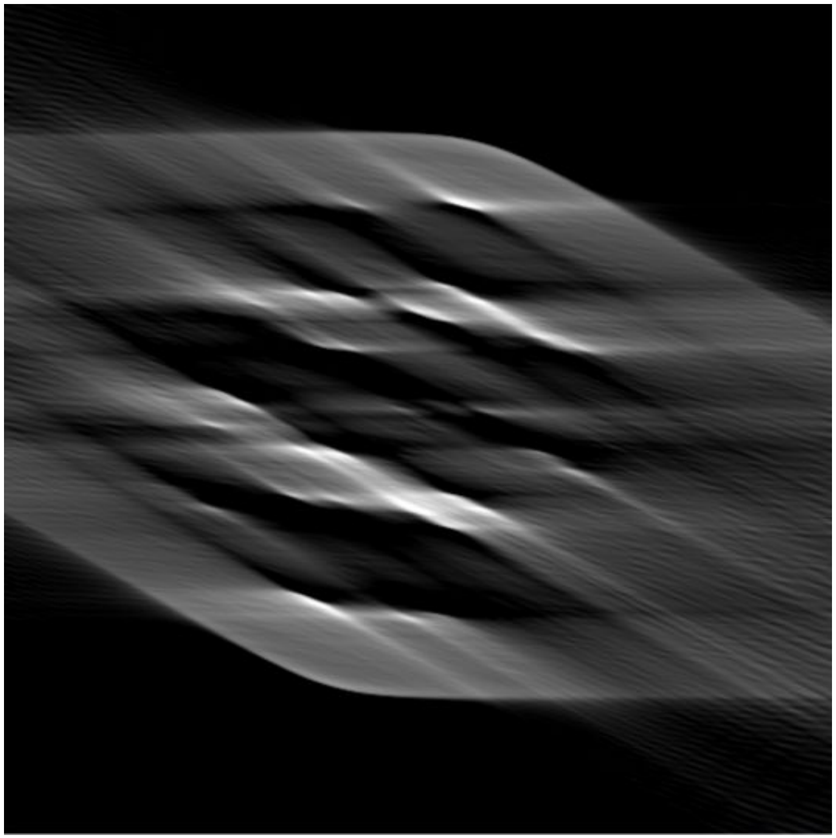
Reconstructions via Challenge Dataset (A)



Score: 0.775538 | λ = 1.0 | θ = 90

Score: 0.998964 | λ = 1.0 | θ = 360

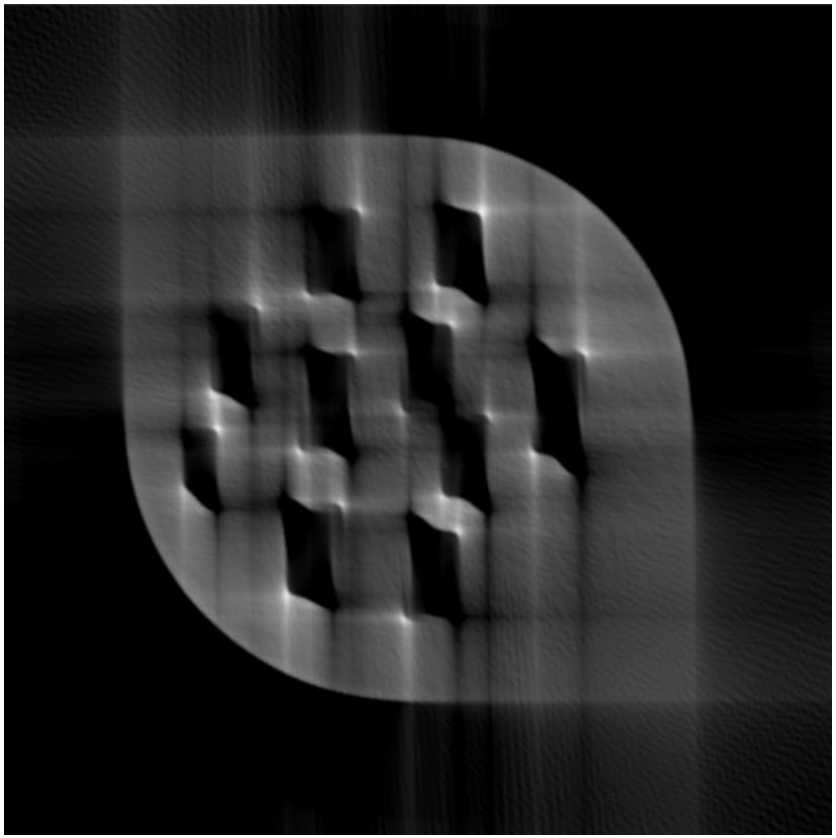
Reconstructions via Challenge Dataset (B)



Score: 0.341525 | λ = 0.0001 | θ = 30

Score: 0.535358 | λ = 0.0001 | θ = 60

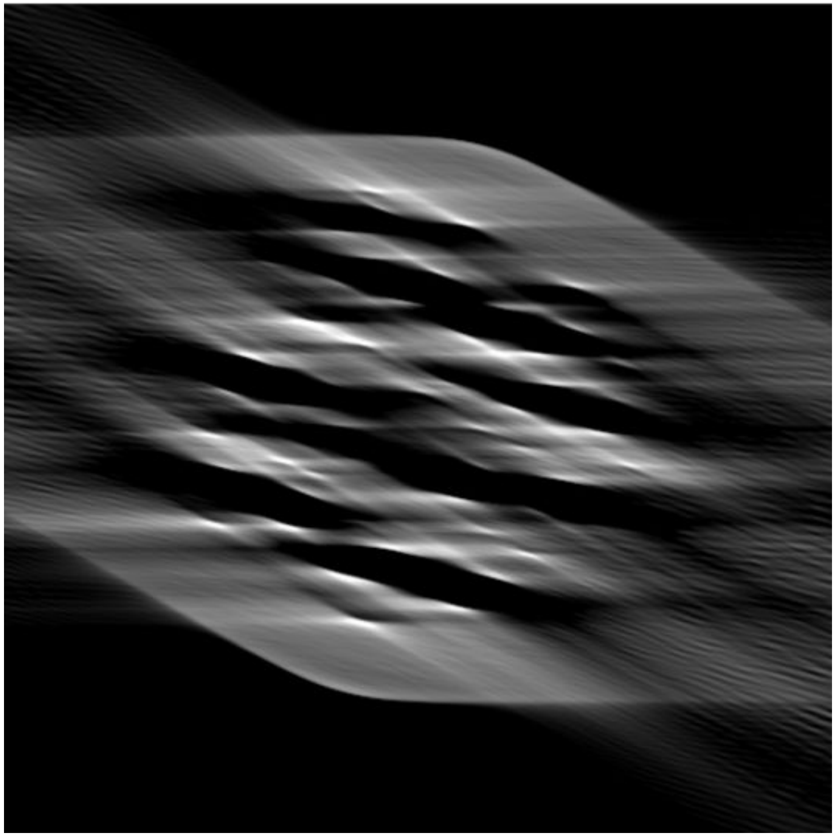
Reconstructions via Challenge Dataset (B)



Score: 0.762010 | λ = 0.0001
| θ = 90

Score: 0.998853 | λ = 0.0001
| θ = 360

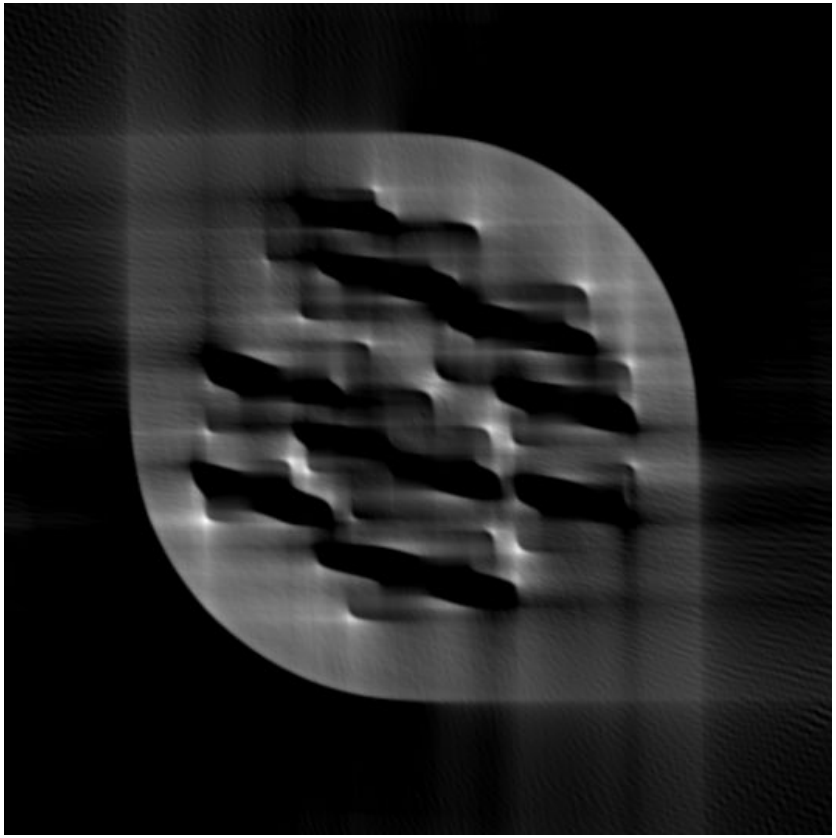
Reconstructions via Challenge Dataset (C)



Score: 0.492479 | λ = 0.0001
| θ = 30

Score: 0.436286 | λ = 0.0001
| θ = 60

Reconstructions via Challenge Dataset (C)



Score: 0.688492 | λ = 0.0001 | θ = 90

Score: 0.994723 | λ = 0.0001 | θ = 360

Takeaways

- With respect to the proximal operators
  - highly sensitive to regularization parameters
  - soft-thresholding “better” than Euclidean proximal (still bound by problem setting)
- With respect to optimization process itself
  - apply non-negativity constraint to eliminate streaky lines
  - use min-max normalization or establish a scale factor for a smoother optimization process
  - introduce stochasticity by randomizing learning rate on k-th iteration
  - normalize arbitrarily if Lipschitz constant is too small

Appendix

```python
class FPGM(aomip.Optimization):
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)
        self.lmbd = 1.0
        self.f = aomip.L1()

    def optimize(self, n=100) -> np.ndarray:
        x, z = self.x0, self.x0
        t = 1.0
        for i in range(n):
            xprev, zprev = x, z
            gradient = self.calculate_gradient(z)
            L = np.linalg.norm(gradient, ord=2) ** 2
            tprev = t
            t = (1 + np.sqrt(1 + 4 * t**2)) / 2
            self.lmbd = (tprev - 1) / t
            z = self.f.proximal(xprev - 1 / L * gradient, lmbd=(1 / L))
            x = z + self.lmbd * (z - zprev)
        return x
```

Appendix

```python
class BB1(aomip.Optimization):
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)

    def optimize(self, n=100, lmbd=1e-3) -> np.ndarray:
        x = self.x0
        gradient = self.calculate_gradient(x)
        prev_gradient = gradient
        prev_x = x
        for i in range(n):
            # avoid division by zero for the first iteration
            if not i:
                step = lmbd
            else:
                gradient_diff = gradient - prev_gradient
                x_diff = x - prev_x
                step = np.dot(x_diff.T, gradient_diff) / gradient_diff.T * gradient_diff
            next_x = x - step * gradient
            error = self.calculate_norm(next_x)
            objective = self.calculate_norm(x)
            terminal_bound = error < objective
            if terminal_bound:
                break
            prev_x = x
            x = next_x
            prev_gradient = gradient
            gradient = self.calculate_gradient(x)
        # descent with optimal step
        for _ in range(n):
            x -= step * gradient
        return x
```

Thank you for listening!