Optimization (continued): Conjugate Gradient, (L-)BFGS, Examples

Yu Feng, Grigor Aslanyan
Berkeley Center for Cosmological Physics
UC Berkeley

Astro Hack Week
Berkeley Institute for Data Science
September 1, 2016

Linear Problems

Want to solve: $\mathbf{Ax} = \mathbf{b}$

$\mathbf{A}$ is a symmetric positive definite n x n matrix

What does it have to do with optimization?

Minimize: $\phi(\mathbf{x}) = \frac{1}{2}\mathbf{x^T Ax} - \mathbf{b^T x}$

Indeed: $\nabla\phi(\mathbf{x}) = \mathbf{Ax} - \mathbf{b} \equiv r(\mathbf{x})$

But why minimize? Can't we just invert the matrix using linear algebra?

- Sure, if your matrix is not huge. Matrix inversion is $O(n^3)$ish. Will work for n=1,000, will struggle for n=10,000 and is hopeless for n = 1,000,000.
- May not even be able to store the whole matrix in memory!

Conjugate Gradient

$\{\mathbf{p_0}, \mathbf{p_1}, \ldots, \mathbf{p_l}\}$ is called *conjugate* with respect to $\mathbf{A}$ if

$$\mathbf{p_i^T A p_j} = 0, \ (i \neq j)$$

$\mathbf{A}$ is a *symmetric positive definite* n x n matrix.

We can minimize ϕ in n steps by successively minimizing it along the individual directions in a conjugate set:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_k \mathbf{p_k}$$

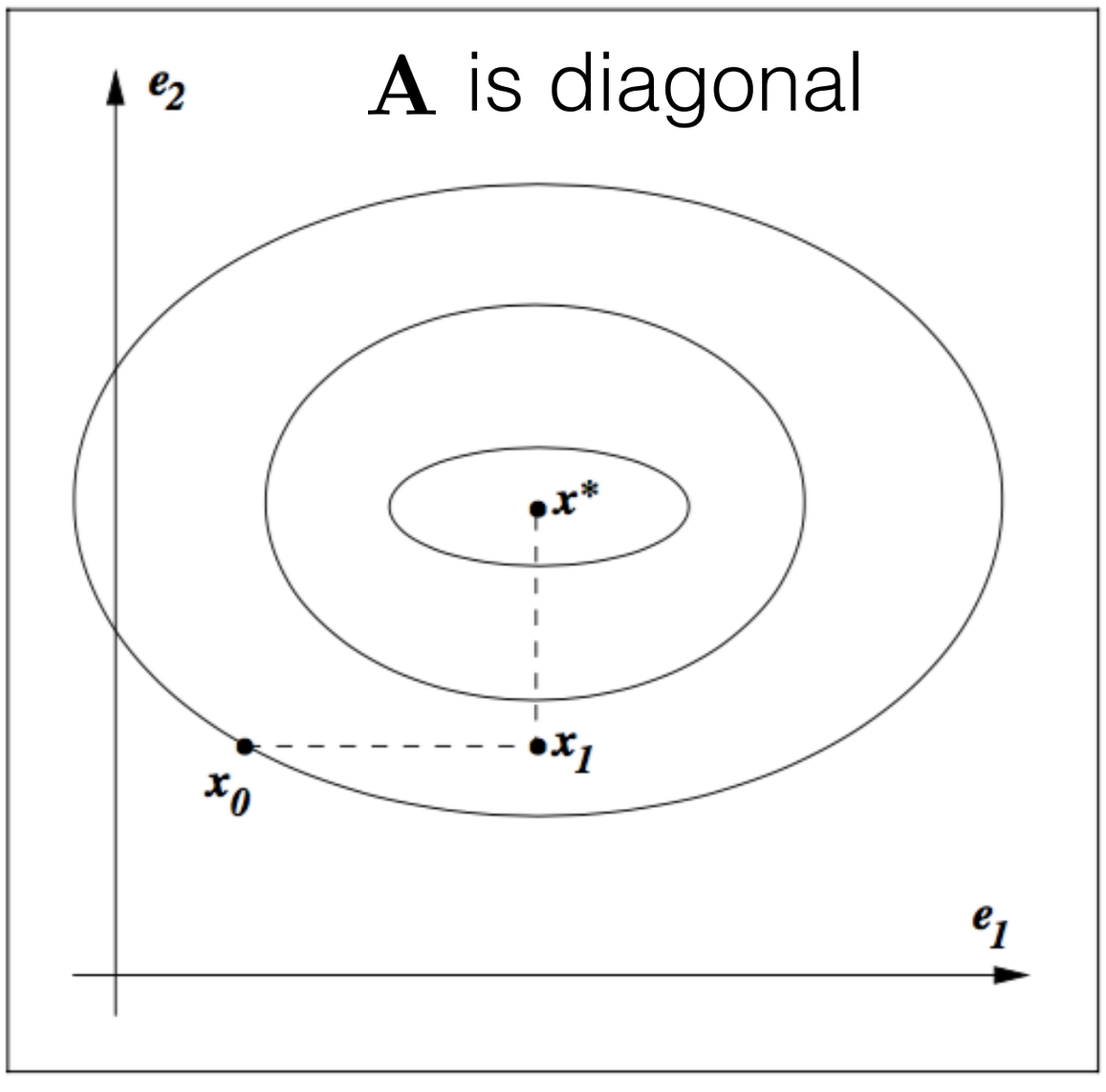
where α_k is the one-dimensional minimizer of ϕ at each step:

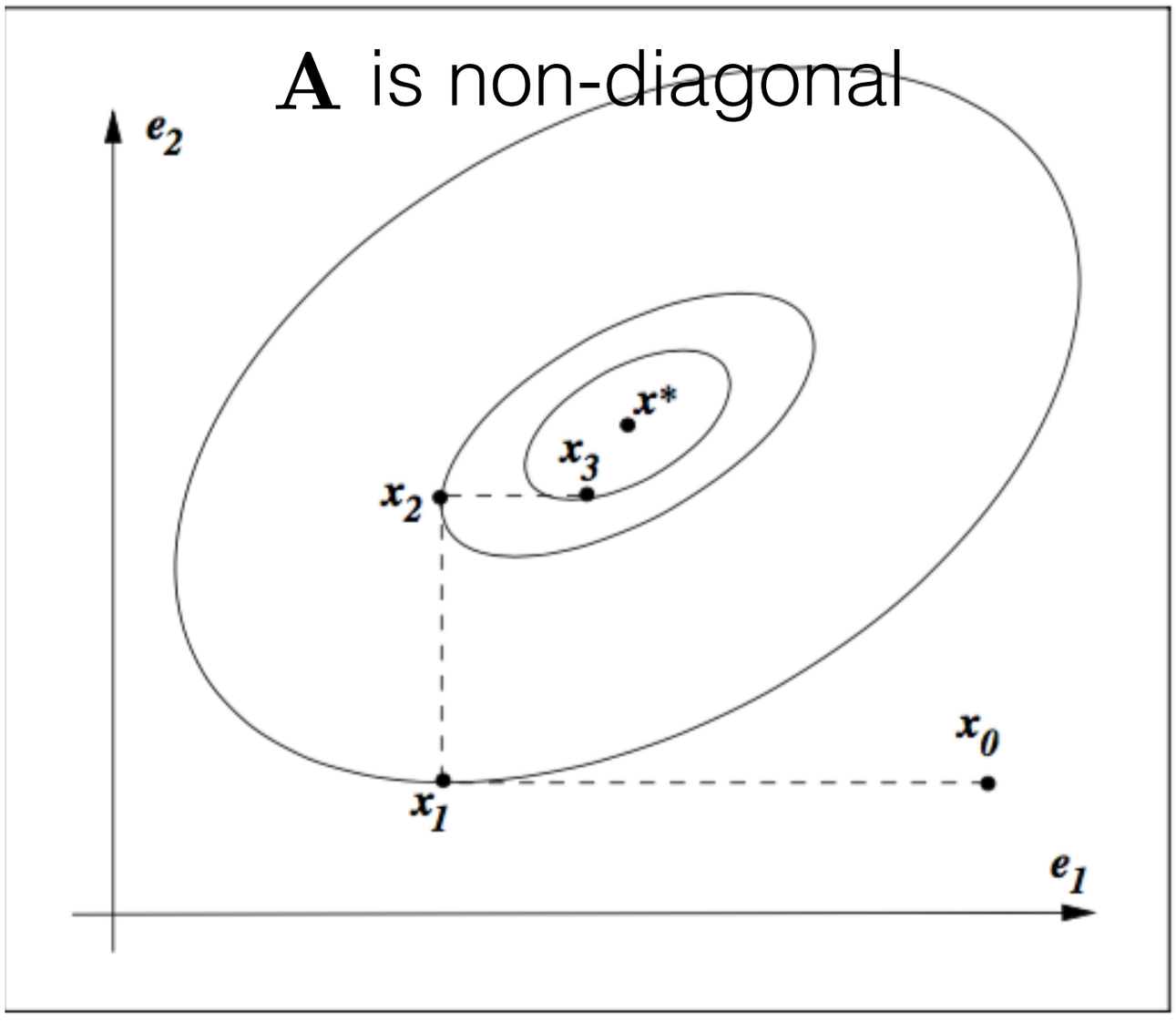
$$\alpha_k = -\frac{\mathbf{r_k^T p_k}}{\mathbf{p_k^T A p_k}}$$

Nocedal, Wright, “Numerical Optimization”

Conjugate Gradient

Successive minimization along coordinate axes





SUCCESS

FAILURE

Nocedal, Wright, “Numerical Optimization”

Conjugate Gradient

Important property: $\mathbf{r_k^T p_i} = 0,\ i = 0, 1, \ldots, k-1$

$\mathbf{x_k}$ is the minimizer of ϕ over

$$\{x | x = x_0 + \text{span}\{\mathbf{p_0}, \mathbf{p_1}, \ldots, \mathbf{p_{k-1}}\}\}$$

Conjugate gradient method: current direction is chosen as a linear combination of the negative residual and the previous direction:

$$\mathbf{p_k} = -\mathbf{r_k} + \beta_k \mathbf{p_{k-1}}$$

The coefficient β_k is determined by the requirement that $\mathbf{p_k}$ and $\mathbf{p_{k-1}}$ are conjugate with respect to $\mathbf{A}$

Need to only remember the previous direction!

Nocedal, Wright, “Numerical Optimization”

Conjugate Gradient Algorithm

Given x_0;

Set $r_0 \leftarrow Ax_0 - b$, $p_0 \leftarrow -r_0$, $k \leftarrow 0$;

while $r_k \neq 0$

$$\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T A p_k};$$

$$x_{k+1} \leftarrow x_k + \alpha_k p_k;$$

$$r_{k+1} \leftarrow r_k + \alpha_k A p_k;$$

$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k};$$

$$p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k;$$

$$k \leftarrow k + 1;$$

end (while)

Nocedal, Wright, “Numerical Optimization”

Convergence

If $\mathbf{A}$ has r distinct eigenvalues then CG will converge in at most r iterations.

If $\mathbf{A}$ has eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ then:

$$||\mathbf{x_{k+1}} - \mathbf{x}^*||_A^2 \leq \left(\frac{\lambda_{n-k} - \lambda_1}{\lambda_{n-k} - \lambda_1} \right)^2 ||\mathbf{x_0} - \mathbf{x}^*||_A^2$$

where $\mathbf{x}^*$ is the solution and

$$\frac{1}{2}||\mathbf{x} - \mathbf{x}^*||_A^2 \equiv \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^T \mathbf{A}(\mathbf{x} - \mathbf{x}^*) = \phi(\mathbf{x}) - \phi(\mathbf{x}^*)$$

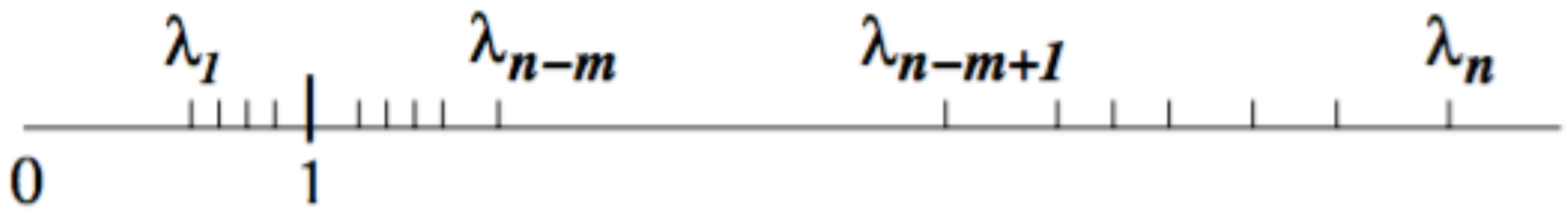
Also: $$||\mathbf{x_k} - \mathbf{x}^*||_\mathbf{A} \leq \left(\frac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1} \right)^k ||\mathbf{x_0} - \mathbf{x}^*||_\mathbf{A}$$

with: $\kappa(\mathbf{A}) = \lambda_n / \lambda_1$ (Euclidean condition number)

Nocedal, Wright, “Numerical Optimization”

Convergence

Example:



Have m large eigenvalues and n-m clustered around 1.
Can get very close to the solution in m+1 steps!

One more property: If the eigenvalues are in r distinct clusters then CG will *approximately* solve the problem in about r steps.

Nocedal, Wright, “Numerical Optimization”

Preconditioning

Make a linear transformation:

$$\hat{\mathbf{x}} = \mathbf{C}\mathbf{x}$$

so that $\mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-1}$ has "good properties" for convergence:

- eigenvalues are clustered or
- the condition number is smaller.

Don't need to do the transformation explicitly. Can reformulate the algorithm to include preconditioning.

Nocedal, Wright, "Numerical Optimization"

Preconditioned CG

Given x_0, preconditioner M;
Set $r_0 \leftarrow Ax_0 - b$;
Solve $My_0 = r_0$ for y_0;
Set $p_0 = -y_0, k \leftarrow 0$;
while $r_k \neq 0$

$$\alpha_k \leftarrow \frac{r_k^T y_k}{p_k^T A p_k};$$

$$x_{k+1} \leftarrow x_k + \alpha_k p_k;$$

$$r_{k+1} \leftarrow r_k + \alpha_k A p_k;$$

Solve $My_{k+1} = r_{k+1}$;

$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T y_{k+1}}{r_k^T y_k};$$

$$p_{k+1} \leftarrow -y_{k+1} + \beta_{k+1} p_k;$$

$$k \leftarrow k + 1;$$

end (while)

Nocedal, Wright, “Numerical Optimization”

Nonlinear CG

Use CG to minimize general convex and/or nonlinear functions f.

Fletcher-Reeves method:

- Need to do line search to calculate α_k,
- Calculate the residual using the gradient of f.

Given x_0;
Evaluate $f_0 = f(x_0), \nabla f_0 = \nabla f(x_0)$;
Set $p_0 \leftarrow -\nabla f_0, k \leftarrow 0$;
while $\nabla f_k \neq 0$
    Compute α_k and set $x_{k+1} = x_k + \alpha_k p_k$;

    Evaluate ∇f_{k+1};

$$\beta_{k+1}^{\mathrm{FR}} \leftarrow \frac{\nabla f_{k+1}^T \nabla f_{k+1}}{\nabla f_k^T \nabla f_k};$$
$$p_{k+1} \leftarrow -\nabla f_{k+1} + \beta_{k+1}^{\mathrm{FR}} p_k;$$
$$k \leftarrow k+1;$$

end (while)

Nocedal, Wright, “Numerical Optimization”

Optimization: BFGS

Quasi-Newton method.

Needs the first, but not second derivatives.

At each iteration the inverse Hessian is estimated from previous iterations (never stored explicitly). A direction of move is deduced, followed by line search.

L-BFGS: Limited memory BFGS. Store and use only a few previous iterations. Works almost as well as BFGS!

BFGS

Expand the function f to second order at the current point x_k:

$$m_k(\mathbf{p}) = f_k + \nabla f_k^T \mathbf{p} + \frac{1}{2}\mathbf{p^T B_k p}$$

Minimizer: $\mathbf{p_k} = -\mathbf{B_k}^{-1}\nabla f_k$

is used as a search direction, and the new point becomes:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_k \mathbf{p_k}$$

Denote: $\mathbf{H_k} = \mathbf{B_k}^{-1}$

$$\mathbf{s_k} = \mathbf{x_{k+1}} - \mathbf{x_k} \qquad \mathbf{y_k} = \nabla f_{k+1} - \nabla f_k$$

Requirement: $\mathbf{B_{k+1}s_k} = \mathbf{y_k} \Leftrightarrow \mathbf{H_{k+1}y_k} = \mathbf{s_k}$

$\mathbf{H_{k+1}} = \min_{\mathbf{H}} ||\mathbf{H} - \mathbf{H_k}||$ subject to $\mathbf{H} = \mathbf{H^T}$, $\mathbf{Hy_k} = \mathbf{s_k}$

Nocedal, Wright, “Numerical Optimization”

BFGS

$$\mathbf{H_{k+1}} = \min_{\mathbf{H}} ||\mathbf{H} - \mathbf{H_k}|| \quad \text{subject to} \quad \mathbf{H} = \mathbf{H^T},\, \mathbf{H y_k} = \mathbf{s_k}$$

Solution:

$$\mathbf{H_{k+1}} = (\mathbf{I} - \rho_\mathbf{k} \mathbf{s_k} \mathbf{y_k^T}) \mathbf{H_k} (\mathbf{I} - \rho_\mathbf{k} \mathbf{y_k} \mathbf{s_k^T}) + \rho_\mathbf{k} \mathbf{s_k} \mathbf{s_k^T}$$

$$\text{with} \quad \rho_\mathbf{k} = \frac{1}{\mathbf{y_k^T} \mathbf{s_k}}$$

Note: We don't need to store any matrices. Can apply the above equation iteratively to compute $\mathbf{H_k} \nabla f_k$

Just need to store all of $\mathbf{s_k}, \mathbf{y_k}$

For L-BFGS store only last m iterations.

Nocedal, Wright, "Numerical Optimization"

Linear Model

data $\longrightarrow$ $\mathbf{d} = \mathbf{R}\mathbf{s} + \mathbf{n}$

signal noise

$$\mathbf{S} = \langle \mathbf{s}\mathbf{s}^\dagger \rangle \quad \mathbf{N} = \langle \mathbf{n}\mathbf{n}^\dagger \rangle$$

$$\mathbf{C} \equiv \langle \mathbf{d}\mathbf{d}^\dagger \rangle = \mathbf{R}\mathbf{S}\mathbf{R}^\dagger + \mathbf{N}$$

Binning: $\mathbf{S}_l = \{s_{m_l}\}\ (m_l = 1, \ldots, M_l)$

$$\mathbf{Q}_l = \mathbf{R}\mathbf{\Pi}_l\mathbf{R}^\dagger$$

projection matrix

$$\mathbf{R}\mathbf{S}\mathbf{R}^\dagger = \sum_l \Theta_l \mathbf{Q}_l$$

Likelihood: $\mathcal{L}(\mathbf{d}|\mathbf{\Theta}) = (2\pi)^{-N/2} \det(\mathbf{C})^{-1/2} \exp\left(-\frac{1}{2}\mathbf{d}^\dagger \mathbf{C}^{-1}\mathbf{d}\right)$

Minimum variance estimator (**Wiener Filter**): $\hat{\mathbf{s}} = \mathbf{S}\mathbf{R}^\dagger\mathbf{C}^{-1}\mathbf{d}$

For gaussian fields this is the same as the maximum probability estimator!

U.Seljak *astro-ph/9710269*

Linear Model

Can do a bunch of linear algebra to calculate Fisher matrix,…

Fisher matrix: $$F_{ll'} = -\left\langle \frac{\partial^2 \ln \mathcal{L}(\mathbf{d}|\mathbf{\Theta})}{\partial\Theta_l \partial\Theta_{l'}} \right\rangle_{\hat{\mathbf{\Theta}}}$$

The inverse is an estimate of the covariance matrix of the parameters:

$$\left\langle \hat{\mathbf{\Theta}}\hat{\mathbf{\Theta}}^\dagger \right\rangle - \left\langle \hat{\mathbf{\Theta}} \right\rangle \left\langle \hat{\mathbf{\Theta}}^\dagger \right\rangle = \mathbf{F}^{-1}$$

Calculation: $$F_{ll'} = \frac{1}{2} tr\left(\mathbf{Q}_l \mathbf{C}^{-1} \mathbf{Q}_{l'} \mathbf{C}^{-1}\right)$$

Window: $$W_{ll'} = \frac{F_{ll'}}{\sum_{l'} F_{ll'}}$$

Power spectrum quadratic estimator:

$$\hat{\Theta}_l = \frac{1}{2} \sum_{l'} F^{-1}_{ll'} \left[\mathbf{d}^\dagger \mathbf{C}^{-1} \mathbf{Q}_{l'} \mathbf{C}^{-1} \mathbf{d} - b_{l'}\right]$$

$$b_l = tr\left[\mathbf{N}\mathbf{C}^{-1}\mathbf{Q}_l\mathbf{C}^{-1}\right]$$

It's unbiased:

$$\left\langle \hat{\Theta}_l \right\rangle = \sum_{l'} W_{ll'} \Theta_{l'}$$

U.Seljak *astro-ph/9710269*

Can also do it with optimization!

Can also do it with optimization!

Noise bias: simulate noise: $\mathbf{d}_n$ Pass through optimizer: $\hat{\mathbf{s}}_n$

$$b_l = \mathbf{\Pi}_l \mathbf{S}^{-1} \hat{\mathbf{s}}_n^\dagger \hat{\mathbf{s}}_n \mathbf{S}^{-1} \mathbf{\Pi}_l$$

Fisher matrix: simulate signal: $\mathbf{s}_s$ Pass through optimizer: $\hat{\mathbf{s}}_s$

For each bin l' simulate extra signal in that bin only: $\Delta\mathbf{s}_{l'}$

$\mathbf{s}_{l'} = \mathbf{s}_s + \Delta\mathbf{s}_{l'}$ Pass through optimizer: $\hat{\mathbf{s}}_{l'}$

$$F_{ll'} = \frac{K_{l'}}{2\Theta_l^2} \left\langle \frac{\sum_{k_l} |\Delta\hat{s}_l(k_l)|^2}{\sum_{k_{l'}} |\Delta s_{l'}(k_{l'})|^2} \right\rangle$$

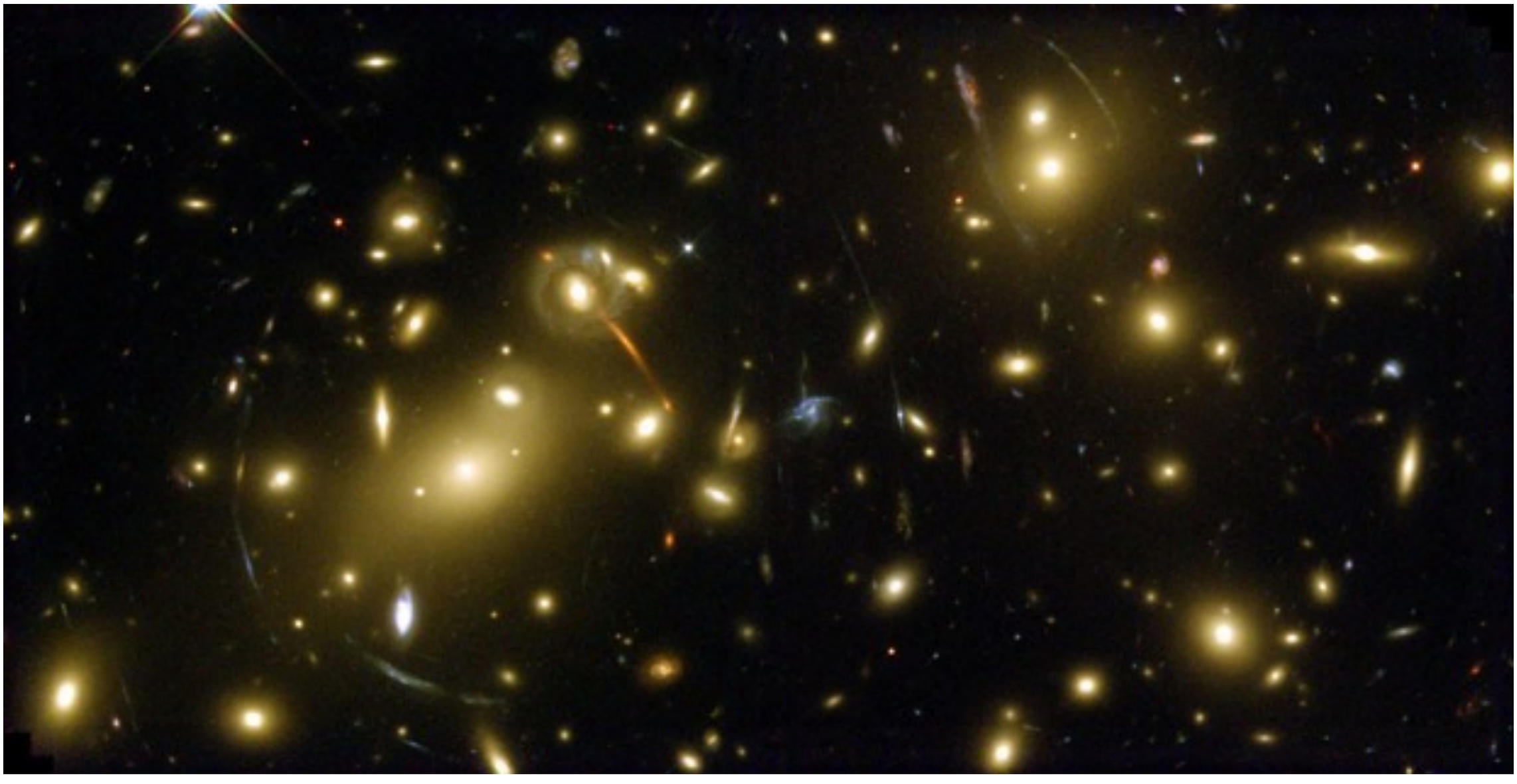
with $\Delta\hat{\mathbf{s}}_{l'} = \hat{\mathbf{s}}_{l'} - \hat{\mathbf{s}}_s$

$K_{l'}$ is the number of modes

Power spectrum:

$$\hat{\Theta}_l = \sum_{k_l} \left(|\hat{s}(k_l)|^2 - \left\langle |\hat{s}_n(k_l)|^2 \right\rangle \right) \left\langle \frac{\sum_{l'} W_{ll'} K_{l'}^{-1} \sum_{k_{l'}} |s_s(k_{l'})|^2}{\sum_{k_l} |\hat{s}_s(k_l)|^2} \right\rangle$$

Linear case: Weak Lensing



WARNING: This is **strong** lensing

Linear case: Weak Lensing

Have a 2D convergence field κ

which is essentially the *projected* density field

Our **signal** (what we will try to reconstruct) is the Fourier modes of κ

The **data** will be the **shear fields** in real space.
Shear is related to convergence:

$$\gamma_1(\mathbf{k}) = \kappa(\mathbf{k})\cos 2\phi_{\mathbf{k}},\ \gamma_2(\mathbf{k}) = \kappa(\mathbf{k})\sin 2\phi_{\mathbf{k}}$$

So this is a linear problem: $\mathbf{d} = \mathbf{R}\mathbf{s} + \mathbf{n}$

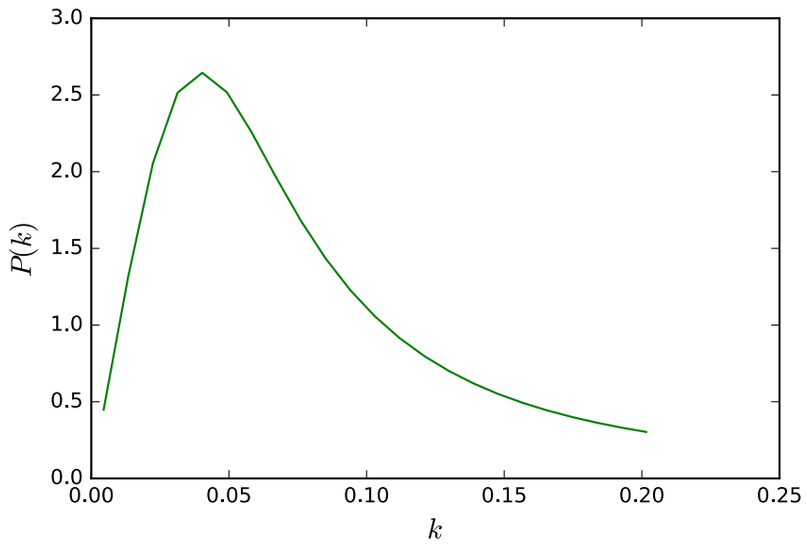
where $\mathbf{s} = \{\kappa(\mathbf{k})\}$ $\mathbf{d} = \{\gamma_1(\mathbf{x}), \gamma_2(\mathbf{x})\}$

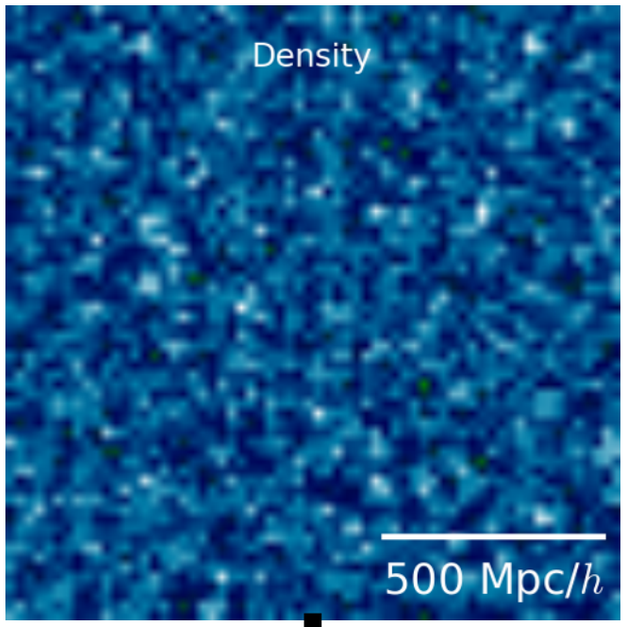
and $\mathbf{R}$ first transforms convergence to shear in Fourier space, then Fourier transforms shear to real space.

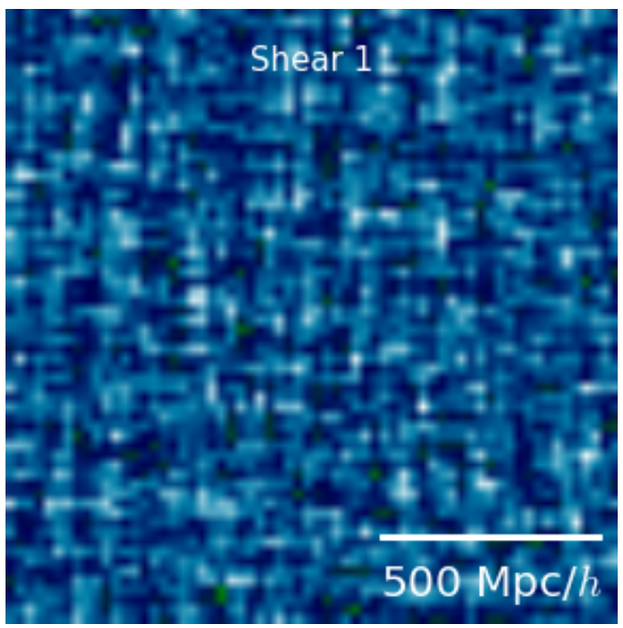
Linear case: Weak Lensing

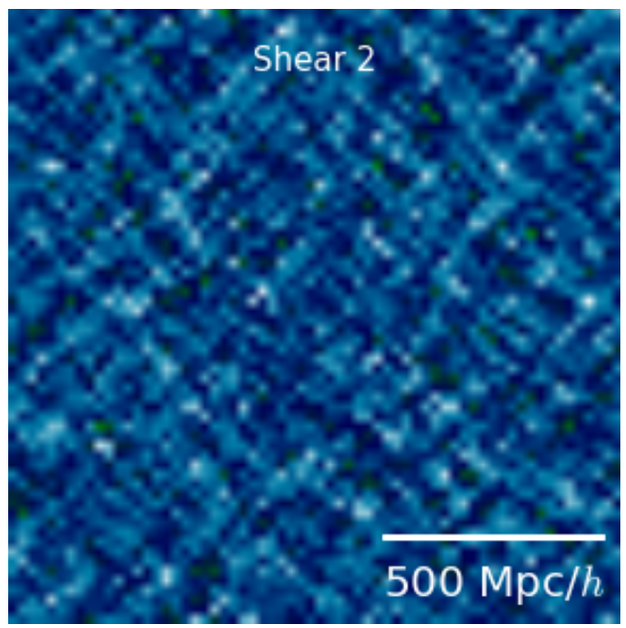
Toy Example: 64x64 grid

Power spectrum



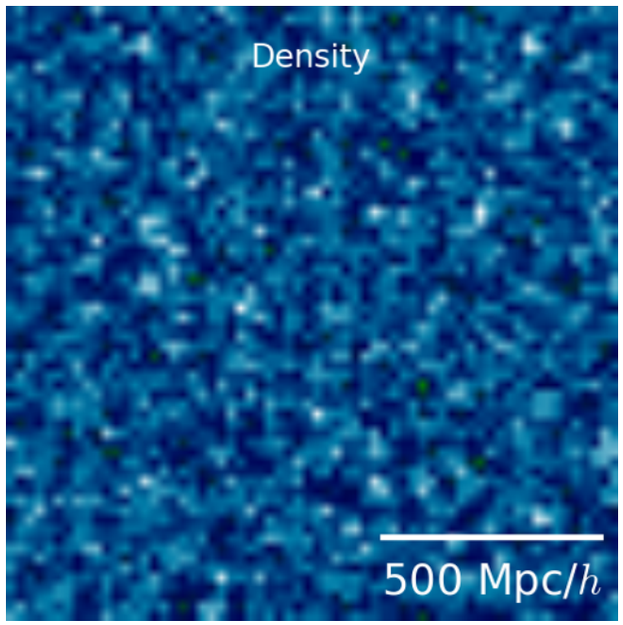


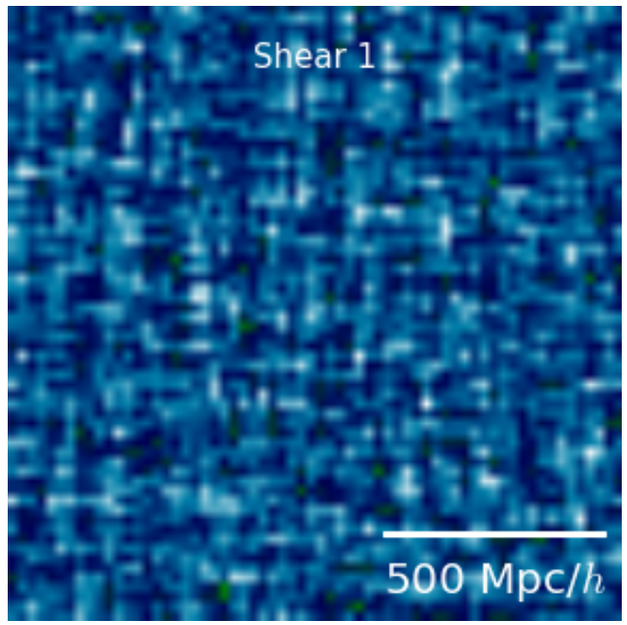


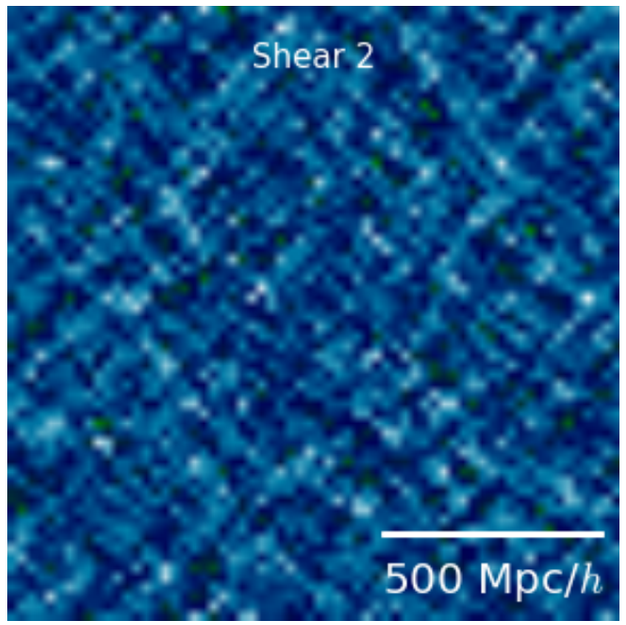


Linear case: Weak Lensing

Toy Example: 64x64 grid

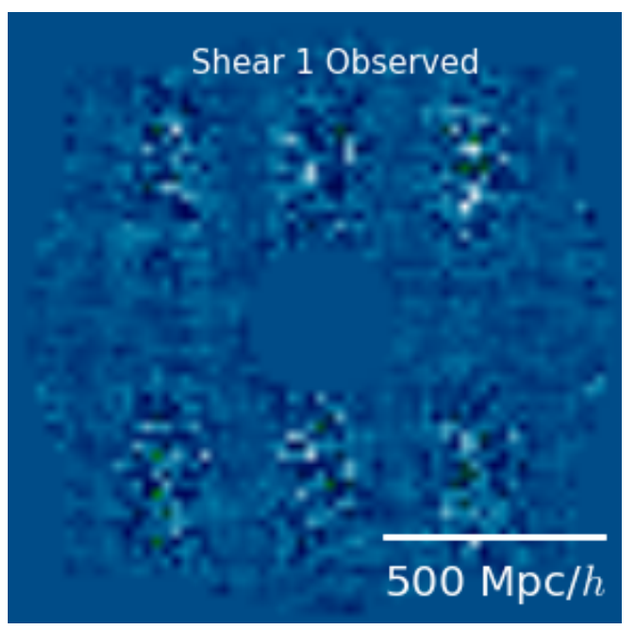


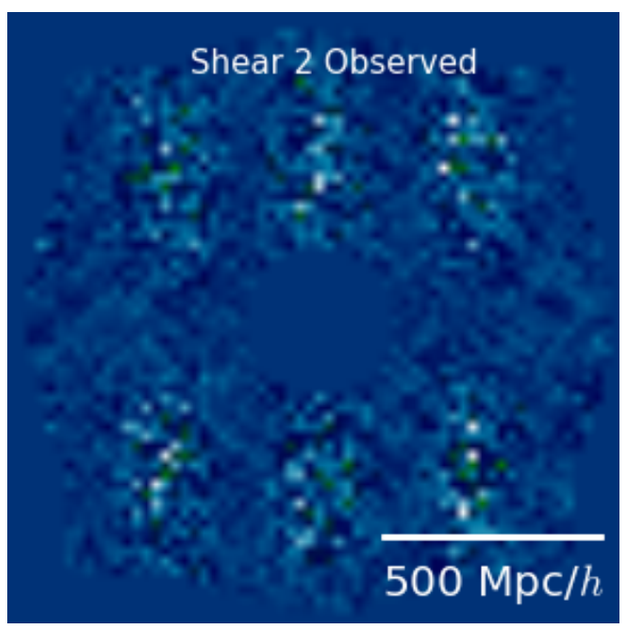




Add noise and mask

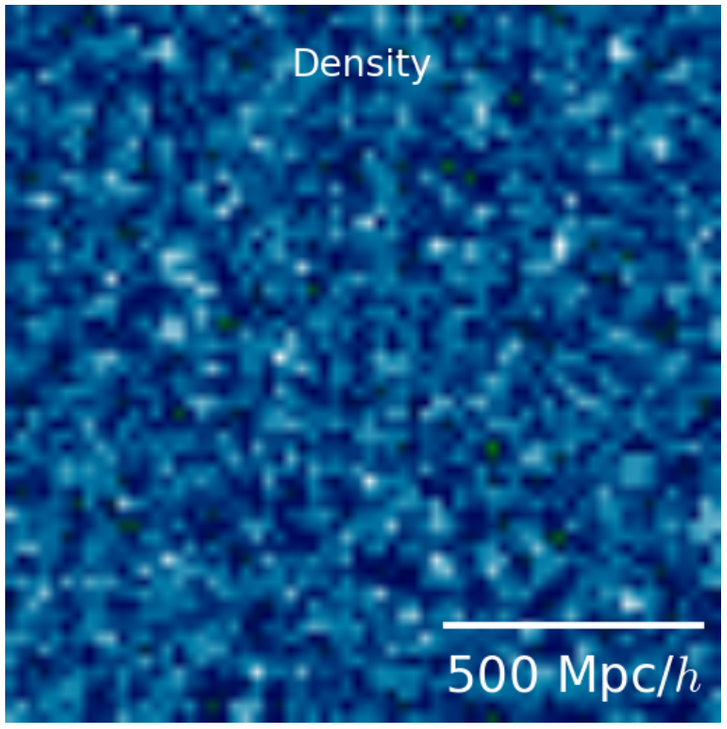
Observed data:



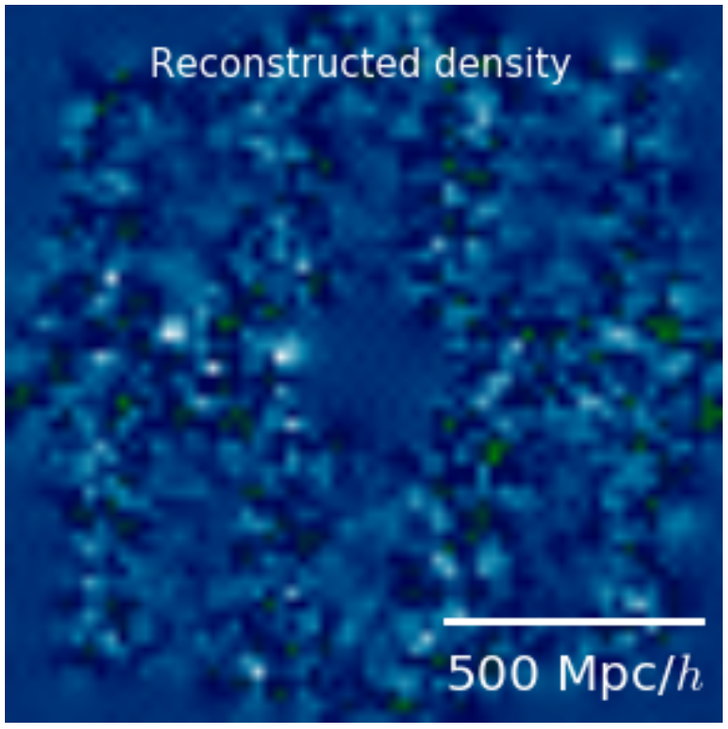


Results

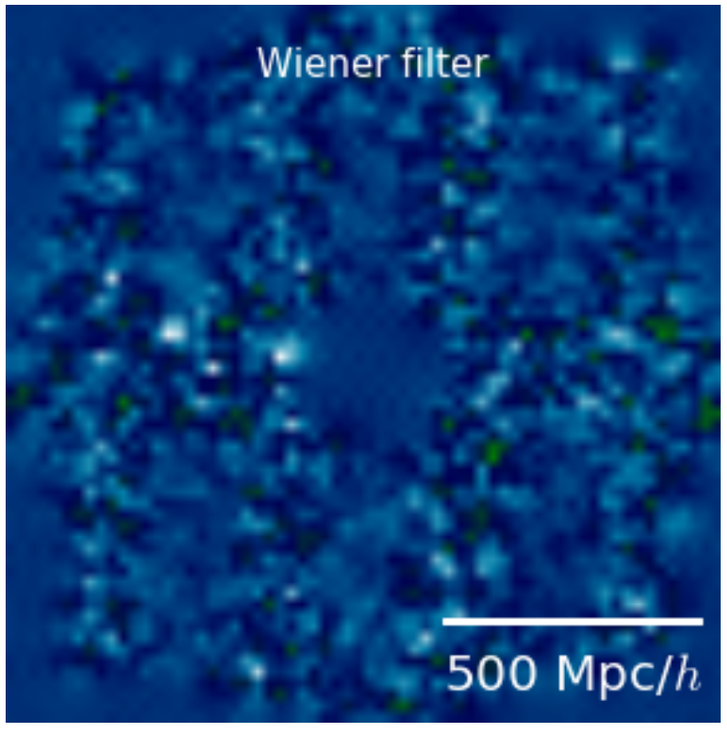
Original

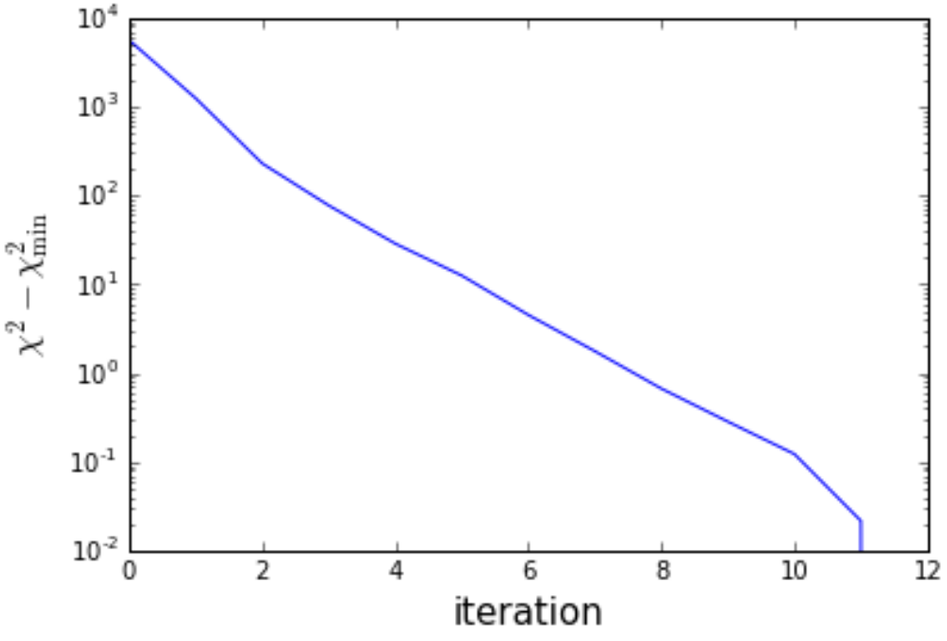


L-BFGS

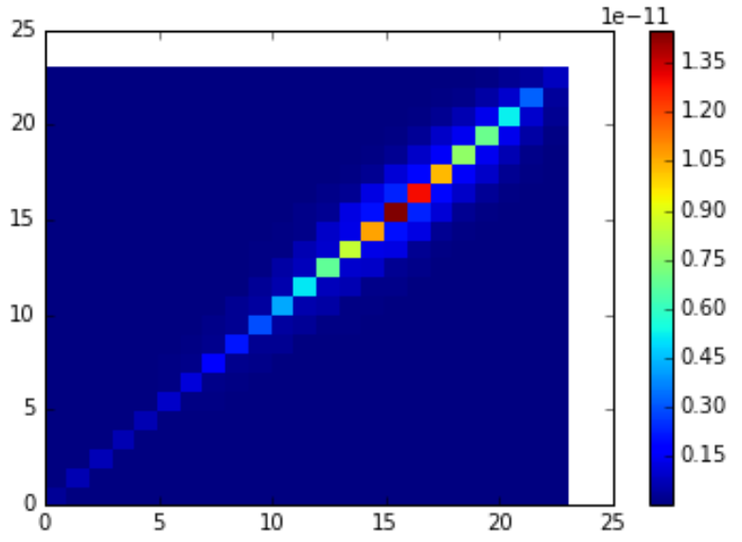


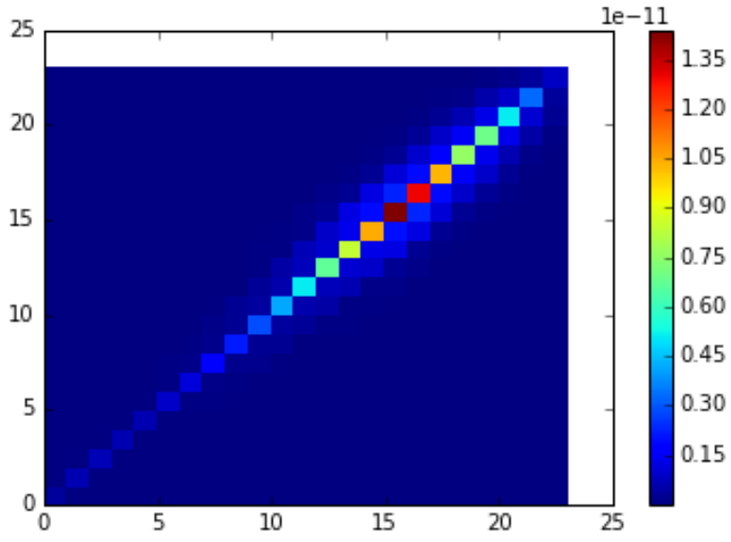
Linear Algebra





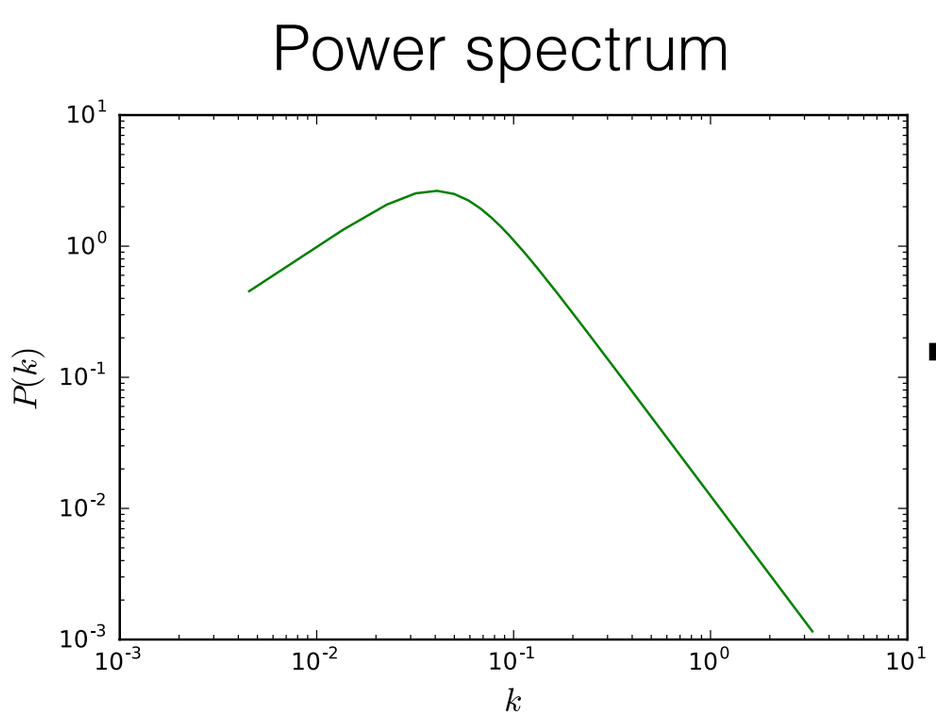
Fisher matrix:

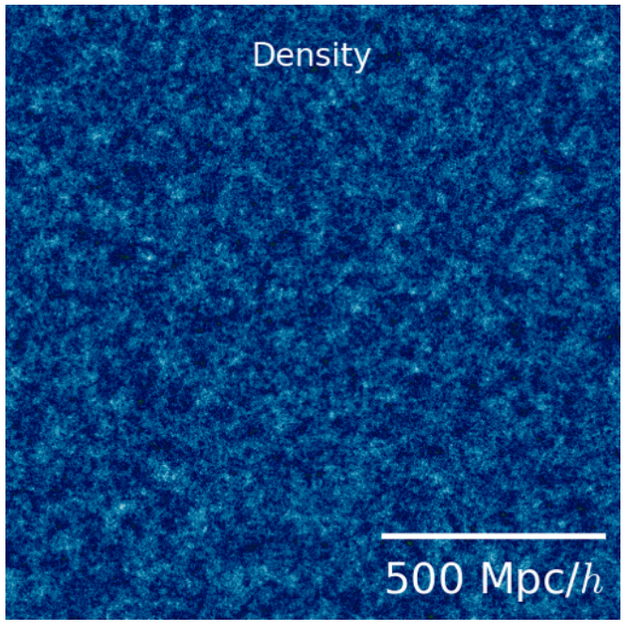


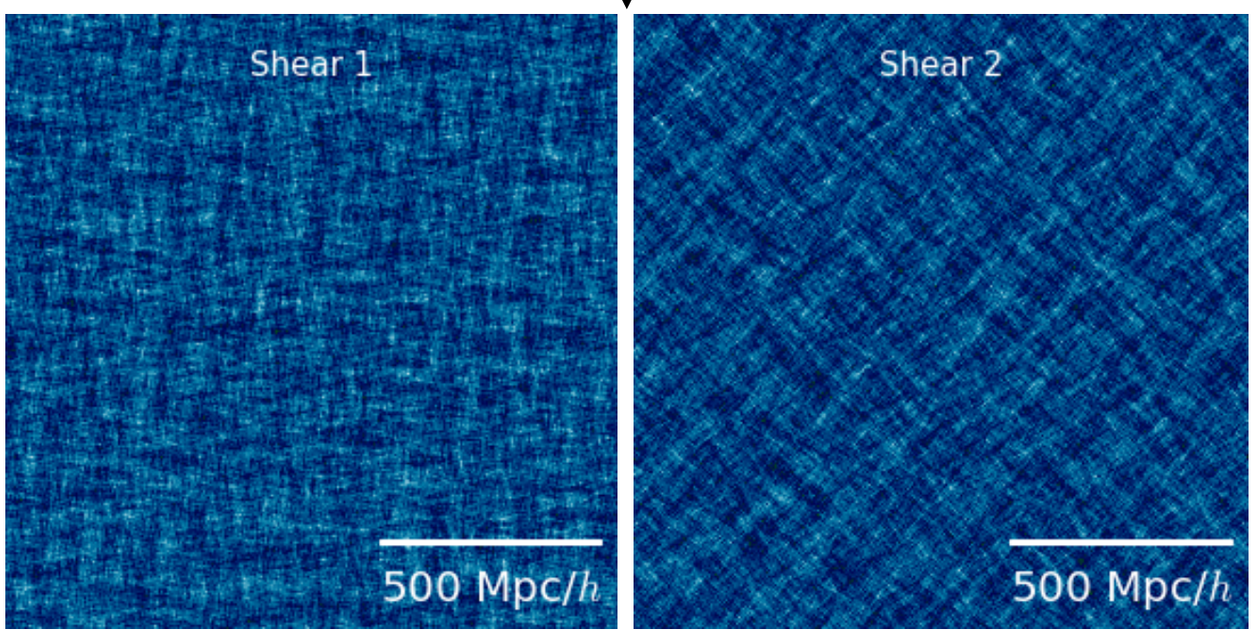


Linear case: Weak Lensing

1024x1024 grid: ~million parameters

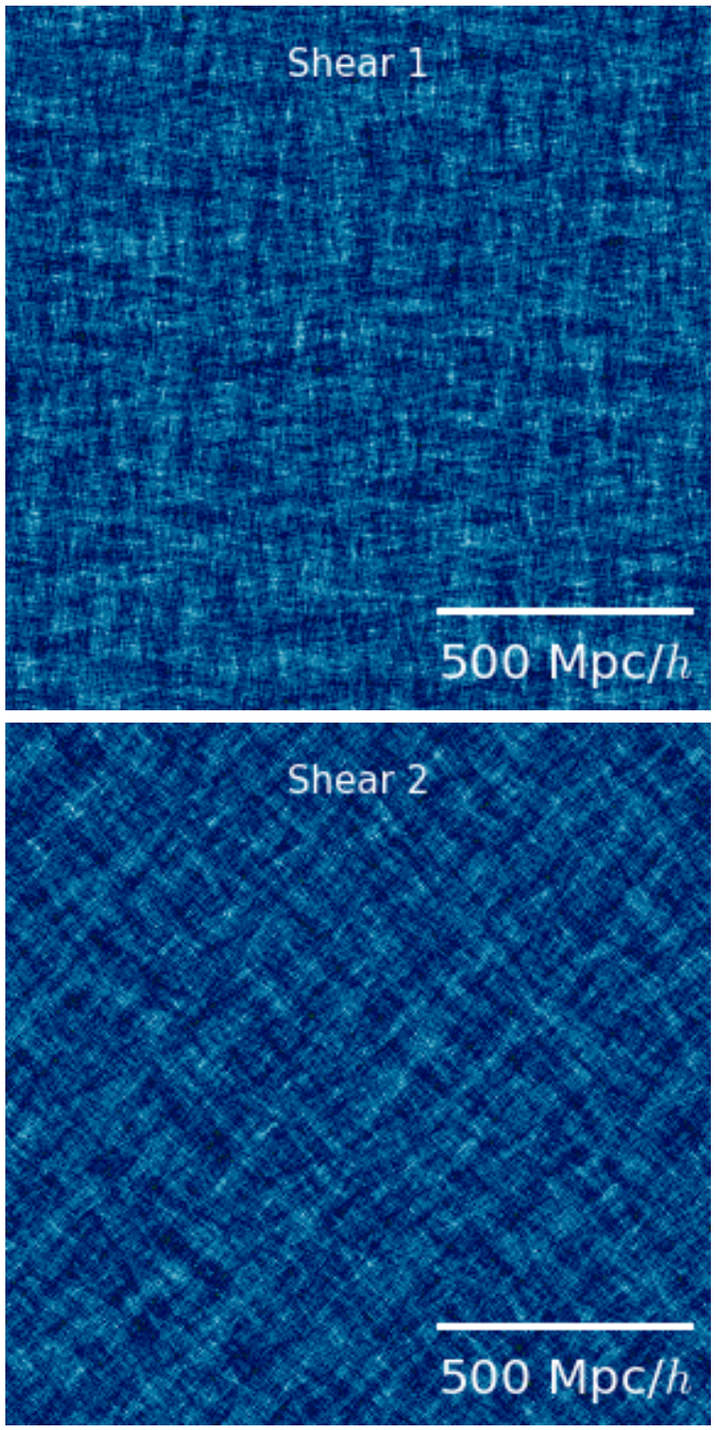


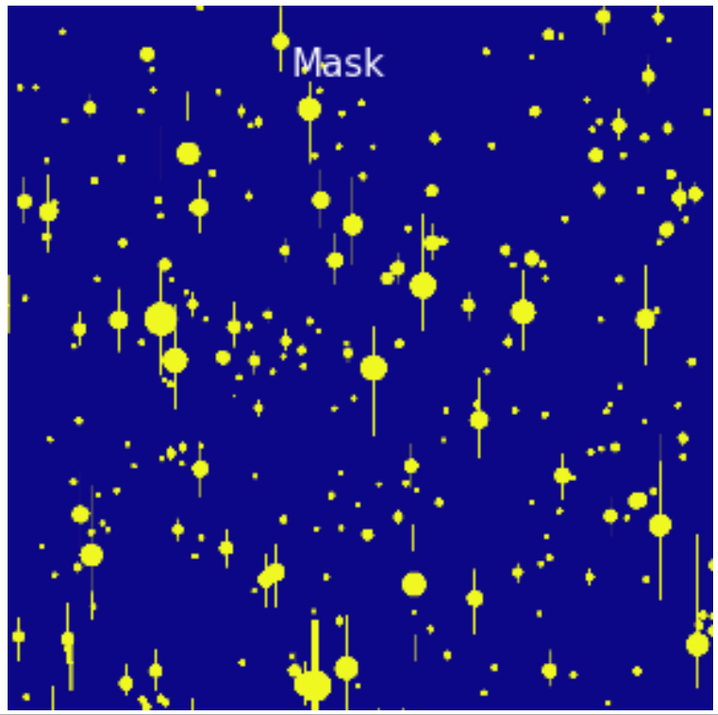


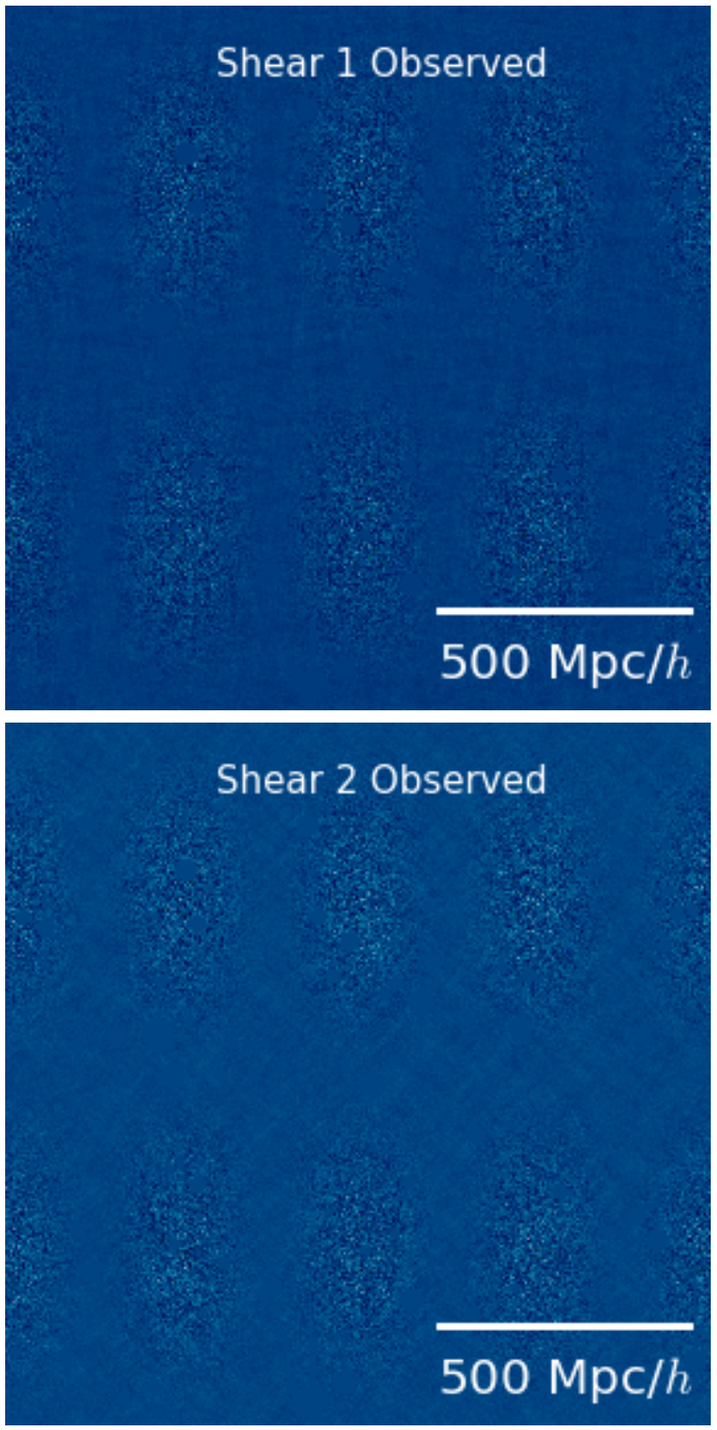


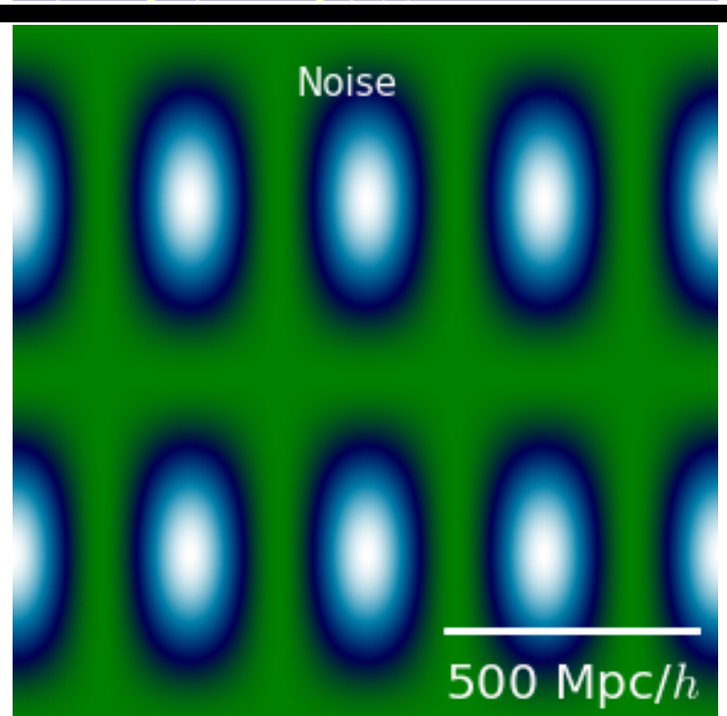
Linear case: Weak Lensing

1024x1024 grid: ~million parameters



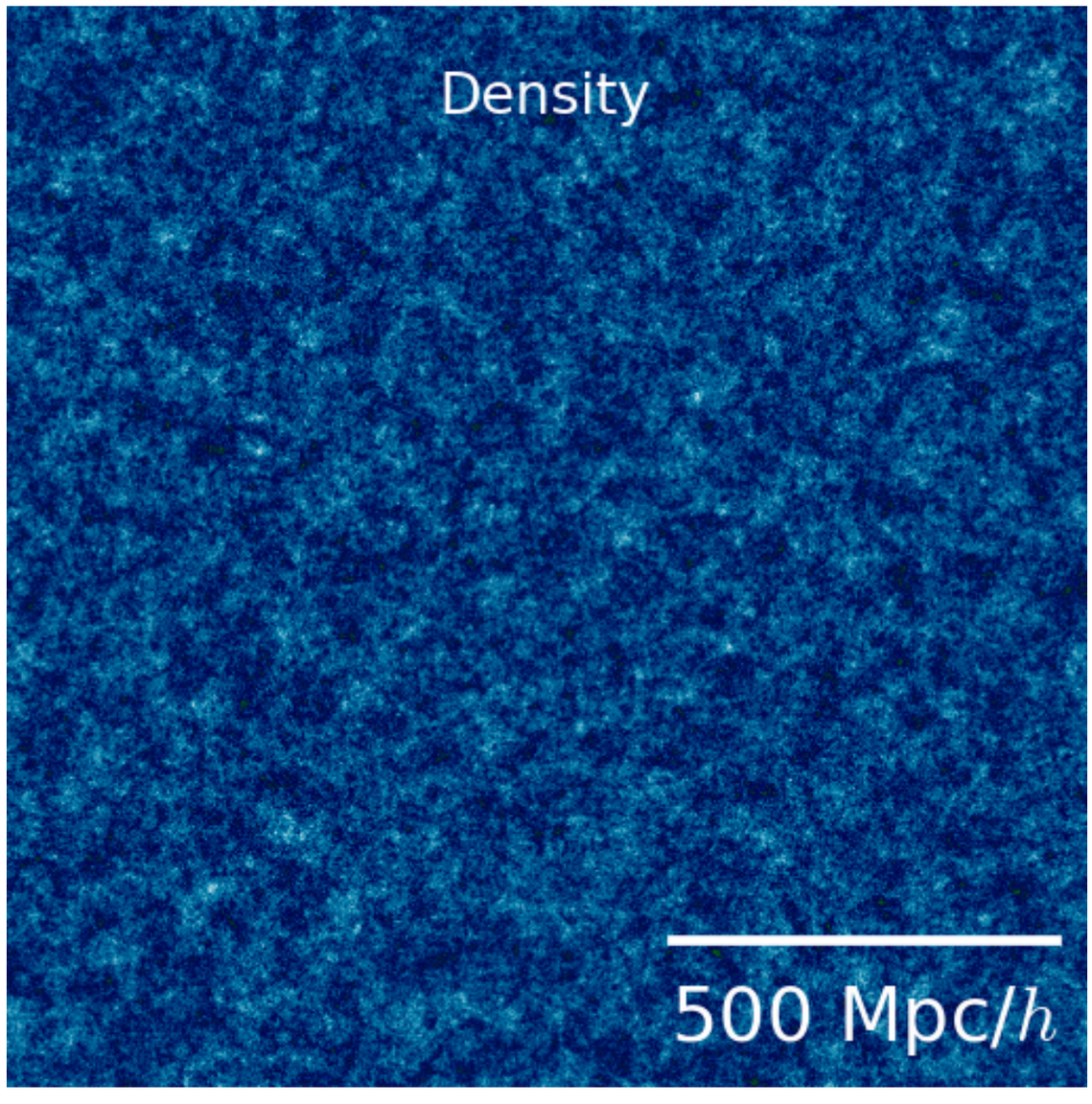




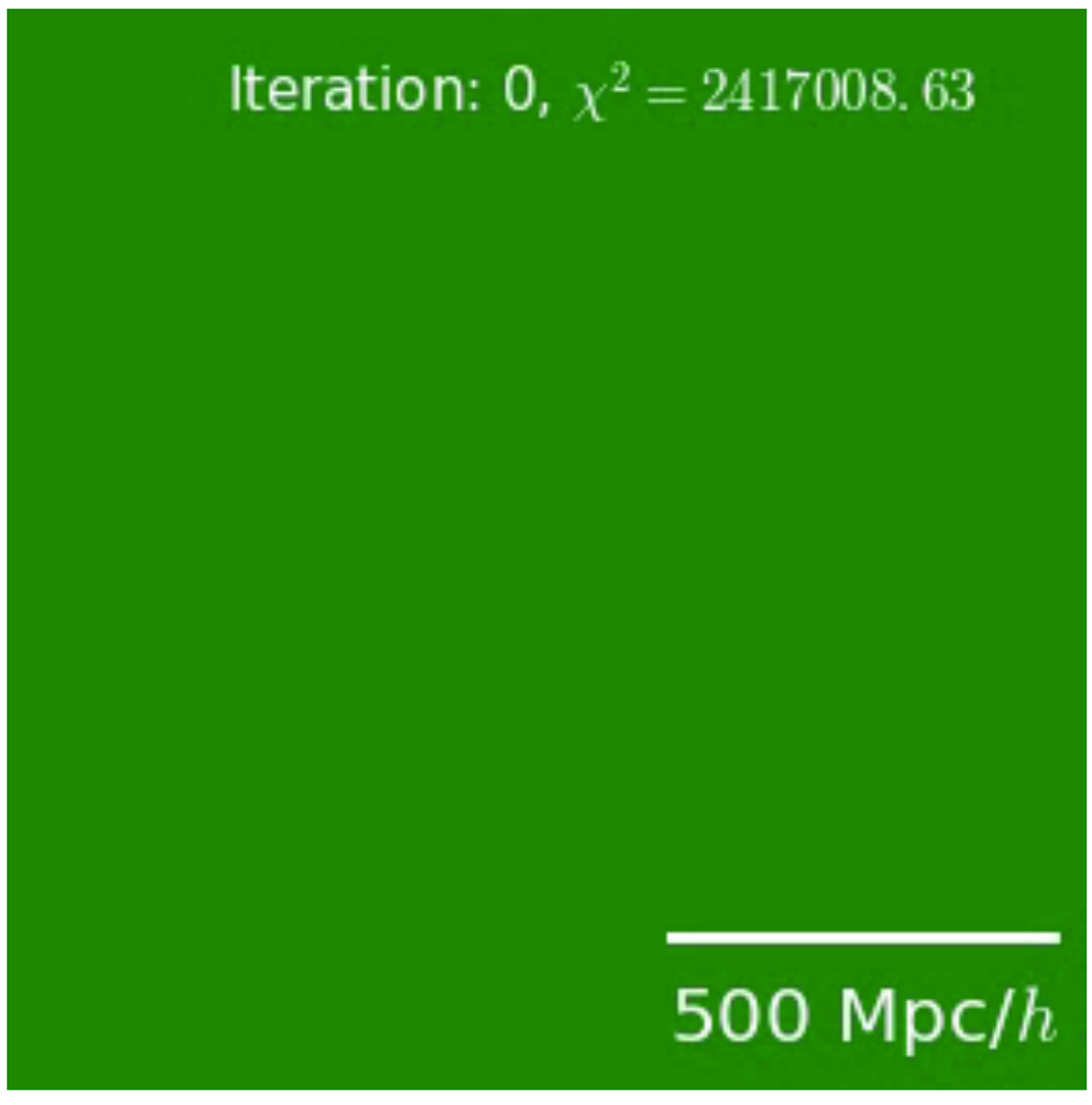


Reconstruction

Original

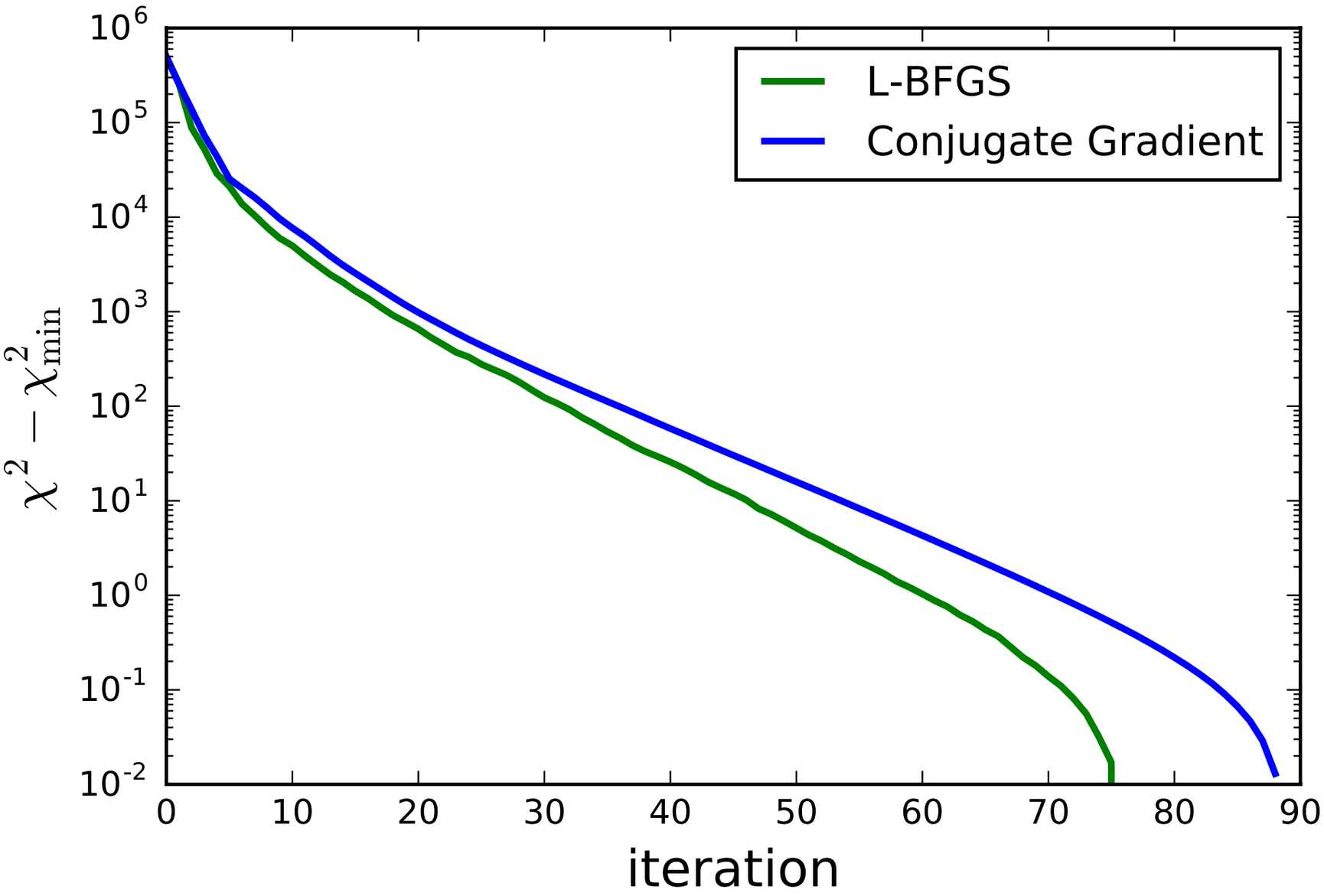


L-BFGS in action



CPU time ~ 1 min.

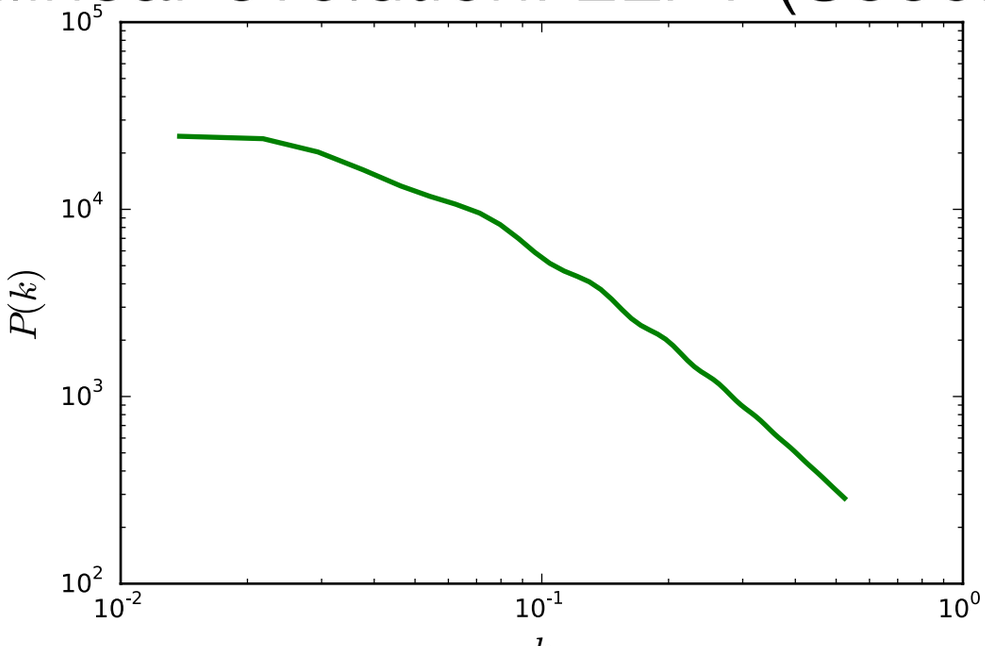
L-BFGS vs. Conjugate Gradient



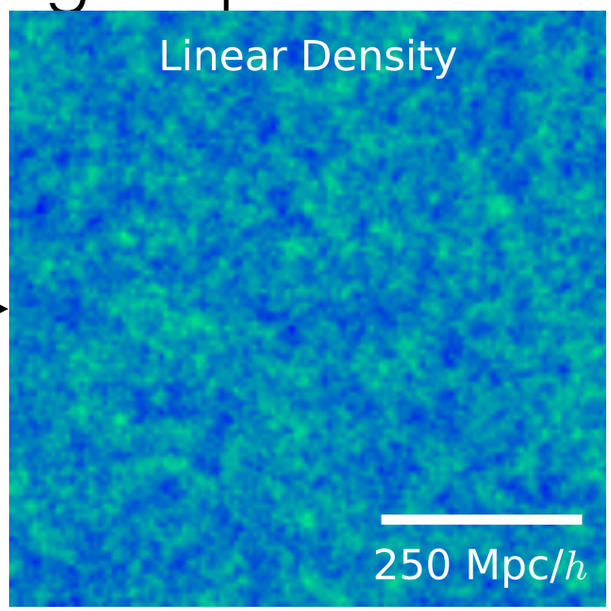
Nonlinear Case: Large Scale Structure

Grid: 128^3, Box size: $(750\ \mathrm{Mpc}/h)^3$

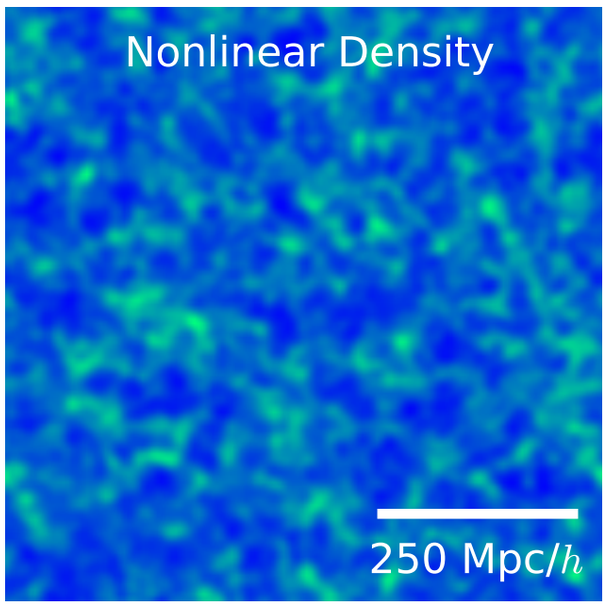
Nonlinear evolution: 2LPT (Second order Lagrangian perturbation theory)



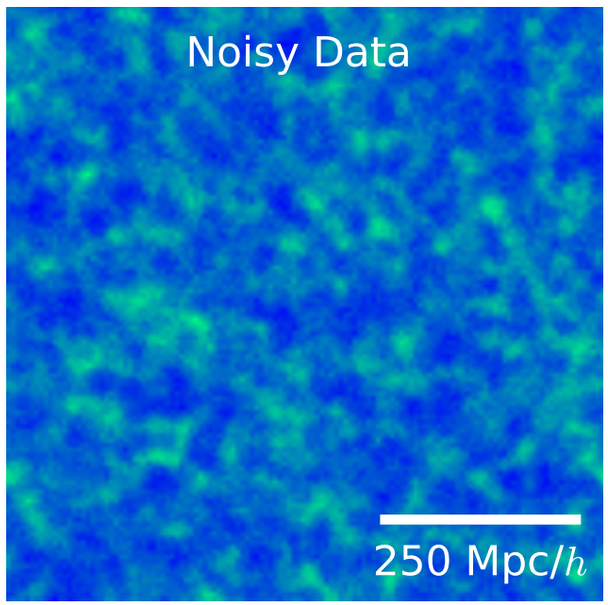
Simulation



2LPT



Noise

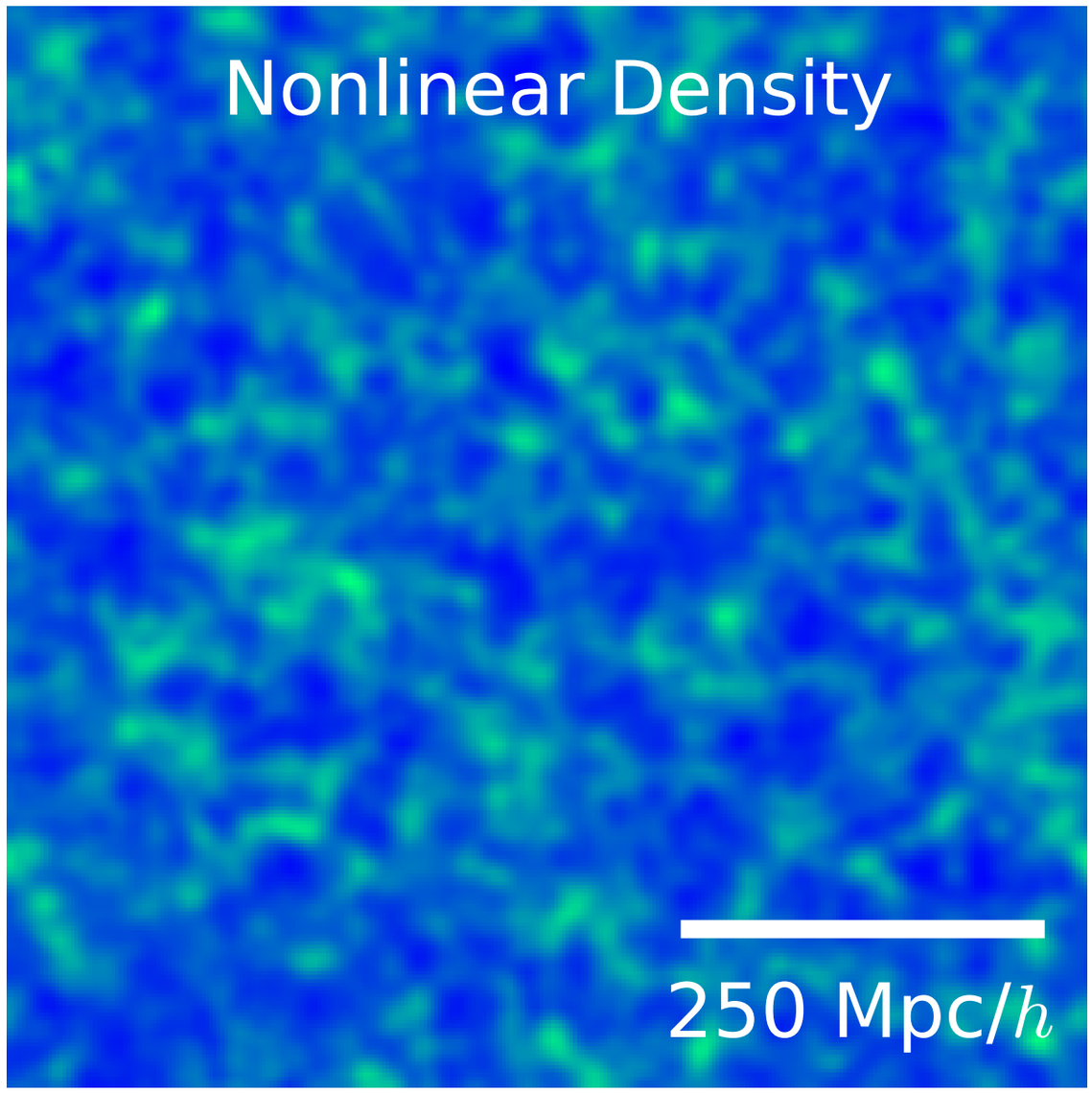


Code used: **fastPM** (Yu Feng, Man-Yat Chu, Uros Seljak, *1603.00476)*

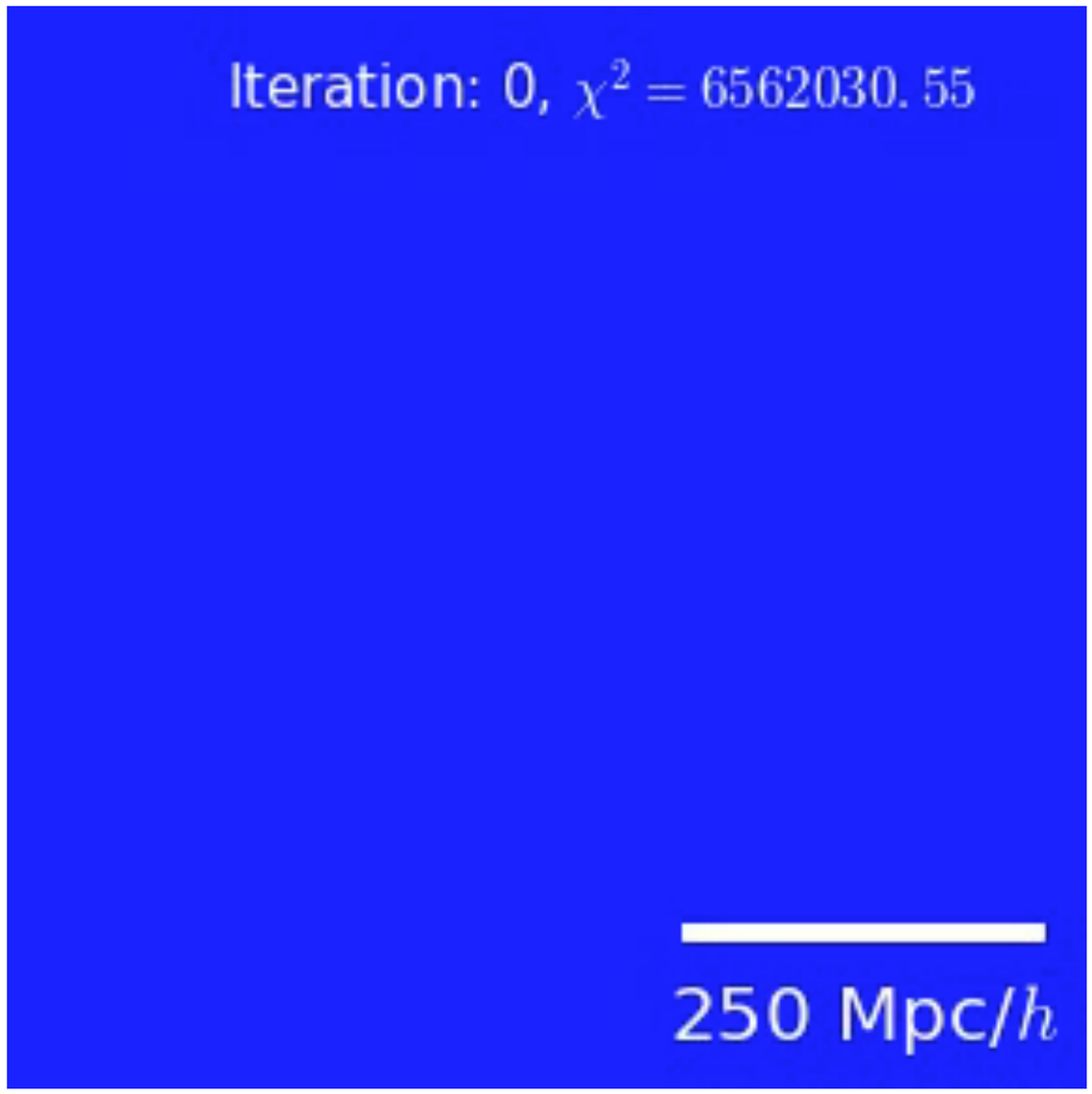
Reconstruction

Nonlinear density

Original

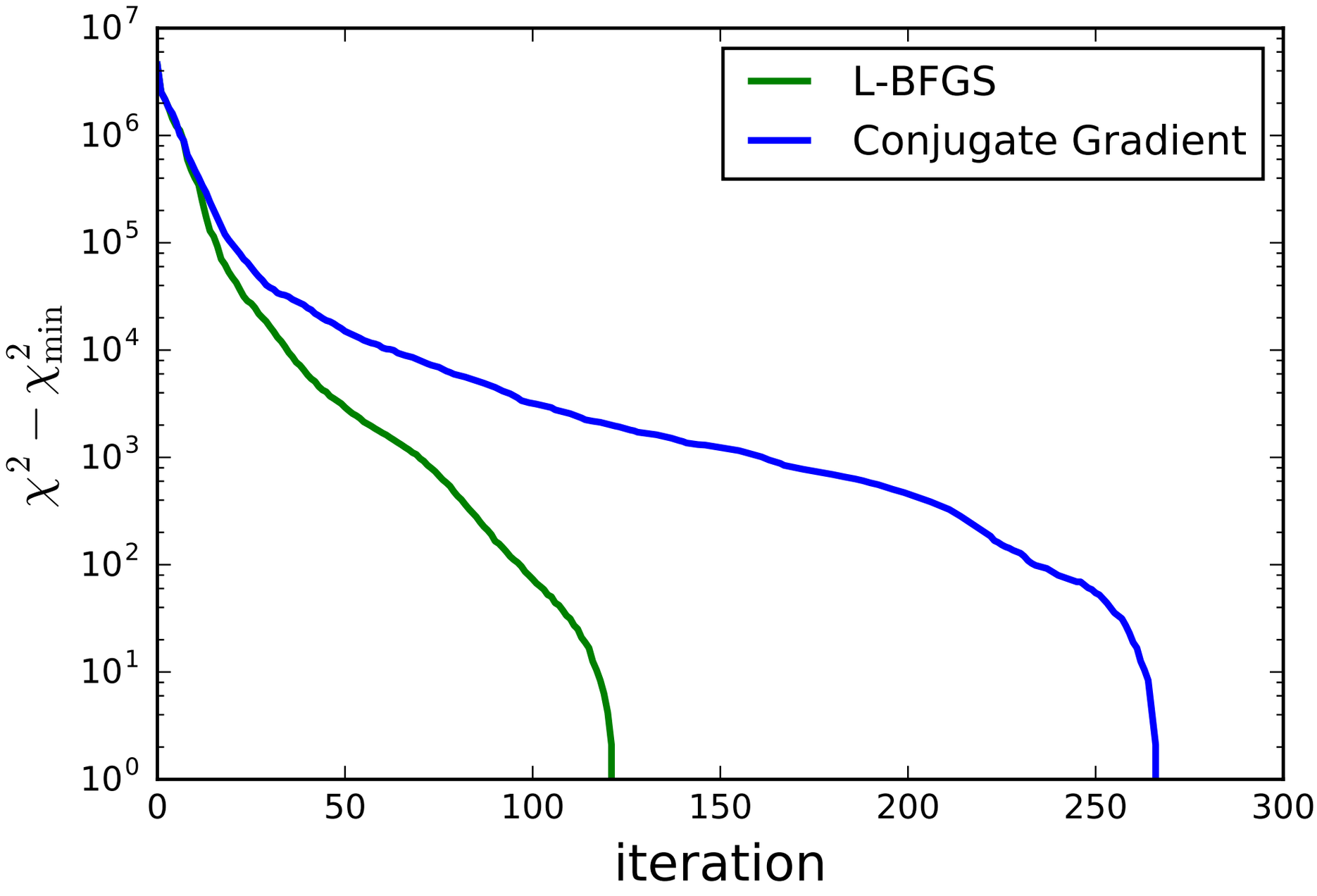


L-BFGS in action



CPU time ~ 1 hour

L-BFGS vs. Conjugate Gradient



(Some) Packages

- **scipy** https://www.scipy.org/ (python)
- **ceres solver** http://ceres-solver.org/ (c++)
- **alglib** http://www.alglib.net/ (c++, c#, …)
- **tensorflow** https://www.tensorflow.org/ (python, c++)
- **cosmo++** https://github.com/aslanyan/cosmopp (c++)

Can’t recommend one, they are all different

but… COSMO++ is clearly the best choice!