Compressed Sensing SECM with Triangle Dictionaries

Shijie Zhou

August 4, 2020

In this note we design the algorithm for reconstructing SECM images from line scans with triangle dictionaries. Unlike discs, the triangle motifs have different orientations and different shapes, which means the triangle dictionaries are more complicated and the associated Lasso problem is also different. Here, we are going to model this multi-motif problem, find its solver step by step, and finally show the real data experiment results.

1 Modeling

To represent equilateral triangle samples with different orientations, we define the dictionary $\boldsymbol{D} \in \mathbb{R}^{n_1 \times n_2 \times k}$ as a 3D matrix array. Each $n_1 \times n_2$ matrix contains one equilateral triangle with a specific orientation as a dictionary element and k is the number of these dictionary elements. Similarly, sparse map signal $\boldsymbol{X} \in \mathbb{R}^{n_1 \times n_2 \times k}$ is also a 3D matrix array and each $n_1 \times n_2$ matrix is corresponding to one dictionary element. The final sample image reconstruction can be represented as the convolution sum of dictionary $\boldsymbol{D}$ and sparse maps $\boldsymbol{X}$, as is shown in Figure 1.

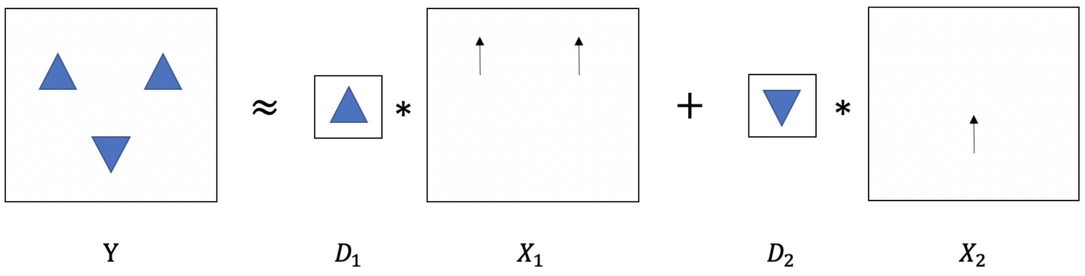


Figure 1: Sparse Representation for Triangle data

In our project, the given observation is microscopic line scans $\boldsymbol{R}$, the dictionary $\boldsymbol{D}$ is well-defined by ourselves, and the aim of our algorithm is to find sparse maps $\boldsymbol{X}$. The associated ideal Lasso problem can be written as:

$$\boldsymbol{X} = \underset{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_K}{\operatorname{argmin}} \frac{1}{2} || \boldsymbol{R} - \mathcal{S}\{\Psi * \mathcal{L}_\Theta[\sum_{i=1}^{K} \boldsymbol{D}_i * \boldsymbol{X}_i]\} ||_2^2 + \lambda \sum_{i=1}^{K} \|\boldsymbol{X}_i\|_1 , \tag{1.1}$$

where $\mathcal{S}$ stands for sampling, Ψ is the point spread function (PSF) and $\mathcal{L}_\Theta$ represents the line scanning process.

However, due to the high coherence of line projections and the nonidealities of PSF, this Lasso formulation in (1.1) does not provide a convincing solution for practical problems in SECM with line probe. To deal with these nonidealities, we reformulate the Lasso problem as:

$$\min_{\boldsymbol{X}\geq\mathbf{0},\boldsymbol{p}\in\mathcal{P}} \sum_{i=1}^{m} \frac{1}{2}||\boldsymbol{R}_i - \mathcal{S}\{\psi(\boldsymbol{p}_i) * \mathcal{L}_{\theta_i}[\sum_{k=1}^{K} \boldsymbol{D}_k * \boldsymbol{X}_k]\}||_2^2 + \sum_{i,j,k} \lambda_{i,j,k}^{(l)} \boldsymbol{X}_{i,j,k}, \tag{1.2}$$

where the parameter vector $\boldsymbol{p}_i$ can represent the peak value, the width of peak, and the rise/decay of PSF for the scan of angle θ_i. $\lambda_{i,j,k}^{(l)}$ is the reweighted penalty variable with l iterations. The following sections illustrate how to solve this optimization problem from a very beginning.

2 Algorithm

2.1 Proximal gradient method

Problem class: Given a function $F(x)$ to be a sum of two convex functions $f(x)$ and $g(x)$, minimize $F(x)$, i.e.,

$$\min_x F(x) = f(x) + g(x) \tag{2.1}$$

$f(x)$: convex, smooth;
$g(x)$: convex, not-neccessarily smooth but non-smooth in our problem
Method:
Firstly, consider the convex and differentiable part $f(x)$ only. For $f(x)$, gradient descent iteration is:

$$x_{k+1} = x_k - \gamma_k \nabla f(x_k) \tag{2.2}$$

This iteration comes from a first-order approximation to $f(\cdot)$ at the point $x = x_k$.
The **lower bound** would be formulated as:

$$f(x') \geq f(x) + < \nabla f(x), x' - x >, \tag{2.3}$$

where x' is a point near x.
Before finding the upper bound, we firstly introduce the L-Lipschitz.

Definition: We say that a differentiable function $f(x)$ has *L-Lipschitz gradient* if

$$||\nabla f(x_1) - \nabla f(x_2)|| \leq L||x_1 - x_2||, \forall x_1, x_2 \in \mathbb{R} \tag{2.4}$$

for some $L > 0$. The quantity L is the Lipschitz constant of ∇f. When this condition holds, a bit of calculus shows that we can complement the linear lower bound with a corresponding quadratic upper bound.

Suppose that f is differentiable, and ∇f is L-Lipschitz, then for every x, x':
The **upper bound** would be formulated as:

$$f(x') \leq f(x) + < \nabla f(x), x' - x > + \frac{L}{2} \left\|x' - x\right\|_2^2 \tag{2.5}$$

Denote this upper bound as $\hat{f}(x', x) = f(x) + < \nabla f(x), x' - x > + \frac{L}{2} \left\|x' - x\right\|_2^2$
Minimize this upper bound, with respect to x':

$$\operatorname*{argmin}_{x'} \hat{f}(x', x) = x - \frac{1}{L}\nabla f(x) \tag{2.6}$$

This is simply a gradient descent step, taken from x, with step size $\frac{1}{L}$.

Now for the whole part $F(x)$, if the gradient $\nabla f(x)$ of the smooth term is Lipschitz, we can still make a upper bound to F, by upper bounding f by a quadratic and leaving the nonsmooth term g untouched:

$$\hat{F}(x', x) = f(x) + < \nabla f(x), x' - x > + \frac{L}{2} \|x' - x\|_2^2 + g(x') \tag{2.7}$$

To find x_{k+1}:

$$x_{k+1} = \operatorname*{argmin}_x \hat{F}(x, x_k) \tag{2.8}$$

Here

$$\begin{aligned} \hat{F}(x, x_k) &= f(x_k) + < \nabla f(x), x - x_k > + \frac{L}{2} \|x - x_k\|_2^2 + g(x) \\ &= \frac{L}{2} \left\| x - (x_k - \frac{1}{L}\nabla f(x_k)) \right\|_2^2 + g(x) + h(x_k) \end{aligned} \tag{2.9}$$

$h(x_k)$ depends on x_k only, so we throw it away when we find x_{k+1}.
Denote $w_k = x_k - \frac{1}{L}\nabla f(x_k)$:

$$\begin{aligned} x_{k+1} &= \operatorname*{argmin}_x \frac{L}{2} \left\| x - (x_k - \frac{1}{L}\nabla f(x_k)) \right\|_2^2 + g(x) \\ &= \operatorname*{argmin}_x \frac{L}{2} \|x - w_k\|_2^2 + g(x) \end{aligned} \tag{2.10}$$

In a sense, this problem asks us to make g as small as possible, while not staying too far from the point w. To simplify the notation, we introduce the proximal operator here. The proximal operator of a convex function g is:

$$prox_g(w) = \operatorname*{argmin}_x \{g(x) + \frac{L}{2} \|x - w\|_2^2\} \tag{2.11}$$

So

$$x_{k+1} = prox_{g/L}(w_k) = \operatorname*{argmin}_x \{g(x) + \frac{L}{2} \|x - w_k\|_2^2\}, \tag{2.12}$$

where $w_k = x_k - \frac{1}{L}\nabla f(x_k)$.

Pseudocode:

Algorithm 1 Proximal gradient method

```
  Input: $x_0 \in \mathbb{R}^n$
  Output: $x_*$
repeat
    $w_k \leftarrow x_k - \frac{1}{L}\nabla f(x_k)$
    $x_{k+1} \leftarrow prox_{g/L}(w_k)$
until $x_k$ converges to $x_*$
```

Convergence guarantee: $F(x_k) - F(x_*)$ converges at a rate of O(1/k). For any $k \geq 1$:

$$F(x_k) - F(x_*) \leq \frac{L \|x_0 - x_*\|_2^2}{2k} \tag{2.13}$$

2.2 Proximal gradient for Lasso

Problem class:

$$\min_x \frac{1}{2} \|y - Ax\|_2^2 + \lambda \|x\|_1 \tag{2.14}$$

Here, $y \in \mathbb{R}^m$ is a vector and $A \in \mathbb{R}^{m\times n}$ is a matrix, where $f(x) = \frac{1}{2}\|y - Ax\|_2^2$ and $g(x) = \lambda \|x\|_1$
Method:
The proximal proximal for this Lasso problem:

$$prox_g(w) = \underset{x}{\operatorname{argmin}}\{\lambda \|x\|_1 + \frac{1}{2}\|x - w\|_2^2\} \tag{2.15}$$

The objective function reaches minimum when the subdifferential of $\lambda \|x\|_1 + \frac{1}{2}\|x - w\|_2^2$ contains 0.

$$0 \in (x - w) + \lambda\partial \|x\|_1 = \begin{cases} x_i - w_i + \lambda & x_i > 0 \\ -w_i + \lambda[-1, 1] & x_i = 0 \\ x_i - w_i - \lambda & x_i < 0 \end{cases}$$

The solution to this equality constraint is the soft-thresholding function applied element-wise, $i = 1, ..., n$:

$$x_{i*} = soft(w_i, \lambda) = sign(w_i)max(|w_i| - \lambda, 0) \tag{2.16}$$

Therefore, for $x_{k+1} = prox_{g/L}(w_k) = \operatorname{argmin}_x\{g(x) + \frac{L}{2}\|x - w_k\|_2^2\}$:

$$x_{k+1} = soft(w_k, \lambda/L) \tag{2.17}$$

Also, $f(x) = \frac{1}{2}\|y - Ax\|_2^2$ whose gradient is clearly Lipschitz: the Lipschitz constant L can be the largest eigenvalue of matrix A^*A. Here A^* is the adjoint of matrix A, and $A^* = A^T$ when A is real. Here is the proof: From the Lipschitz,

$$\begin{aligned} ||\nabla f(x_1) - \nabla f(x_2)|| &= || - A^*(y - Ax_1) + A^*(y - Ax_2)|| && (2.18) \\ &= ||A^*A(x_1 - x_2)|| && (2.19) \\ &\le ||A^*A||||(x_1 - x_2)|| && (2.20) \\ &= \lambda_{max}(A^*A)||x_1 - x_2|| && (2.21) \\ &= L||x_1 - x_2|| && (2.22) \end{aligned}$$

Therefore $L = \lambda_{max}(A^*A) = ||A||^2$, where $||A|| = \sqrt{\lambda_{max}(A^*A)}$.
For $f(x)$ in this Lasso problem, $\nabla f(x) = A^*Ax - A^*y = A^*(Ax - y)$.
Pseudocode:
Proximal gradient descent algorithm for Lasso: iterative soft-thresholding algorithm (ISTA)

Algorithm 2 Iterative soft-thresholding algorithm (ISTA)

```
   Problem: $\min_x \frac{1}{2}\|y - Ax\|_2^2 + \lambda \|x\|_1$, given $y \in \mathbb{R}^m$, $A \in \mathbb{R}^{m\times n}$
   Input: $x_0 \in \mathbb{R}^n$ and $L \ge \lambda_{max}(A^T A)$
1: while $x_k$ not converged ($k = 1, 2, ...$) do
2:     $w_k \leftarrow x_k - \frac{1}{L}A^*(Ax_k - y)$
3:     $x_{k+1} \leftarrow soft(w_k, \lambda/L)$
4: end while
   Output: $x_* \leftarrow x_k$
```

2.3 Proximal gradient with more general maps

We will need to solve more general problems.
Problem class:

$$\min_{\boldsymbol{X}} \frac{1}{2}\|y - \mathcal{A}[\boldsymbol{X}]\|_2^2 + \lambda \|\boldsymbol{X}\|_1 \tag{2.23}$$

where $\boldsymbol{X} \in \mathbb{R}^{n_1 \times n_2}$ and $\mathcal{A} : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^m$ is a linear map.
To solve this kind of probelm, the adjoint of the linear map is essential. In this section, we illustrate how to find the adjoint of a linear map and how to solve the Lasso problem with general linear map. First, we introduce the definition of adjoint map.
Definition: For the linear map above $\mathcal{A} : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^m$, its adjoint map is the **unique** linear map $\mathcal{A}^* : \mathbb{R}^m \to \mathbb{R}^{n_1 \times n_2}$ satisfying:

$$\forall x, y, \quad < y, \mathcal{A}[x] >=< \mathcal{A}^*[y], x > \tag{2.24}$$

Here we illustrate how to find the adjoint of a linear map with 3 examples:
Examples:
1. Matrix multiplication: $\mathcal{A}[x] = Ax : \mathbb{R}^n \to \mathbb{R}^m$
$\mathcal{A} : \mathbb{R}^n \to \mathbb{R}^m$
$\mathcal{A}^* : \mathbb{R}^m \to \mathbb{R}^n$
$\forall x \in \mathbb{R}^n, y \in \mathbb{R}^m$:

$$< y, \mathcal{A}[x] > =< \mathcal{A}^*[y], x > \tag{2.25}$$
$$y^T Ax = (A^* y)^T x \tag{2.26}$$
$$y^T Ax = y^T (A^*)^T x \tag{2.27}$$
$$y^T [A - (A^*)^T] x = 0 \tag{2.28}$$
$$A = (A^*)^T \tag{2.29}$$
$$A^* = A^T \tag{2.30}$$

2. Trace: $\mathcal{A}[X] = tr[X] : \mathbb{R}^{n \times n} \to \mathbb{R}$
$\mathcal{A} = tr[X] : \mathbb{R}^{n \times n} \to \mathbb{R}$
$\mathcal{A}^* = tr^*[y] : \mathbb{R} \to \mathbb{R}^{n \times n}$
$\forall X \in \mathbb{R}^{n \times n}, y \in \mathbb{R}$:

$$< y, \mathcal{A}[X] > =< \mathcal{A}^*[y], X > \tag{2.31}$$
$$ytr[X] = tr[(tr^*[y])^T X] \tag{2.32}$$
$$tr[yX] = tr[(tr^*[y])^T X] \tag{2.33}$$
$$yIX = (tr^*[y])^T X \tag{2.34}$$
$$yI = (tr^*[y])^T \tag{2.35}$$
$$tr^*[y] = yI \tag{2.36}$$

3. Diagonal:$\mathcal{A}[X] = diag[X] : \mathbb{R}^{n \times n} \to \mathbb{R}^n$
$\mathcal{A} = diag[X] : \mathbb{R}^{n \times n} \to \mathbb{R}^n$
$\mathcal{A}^* = diag^*[y] : \mathbb{R}^n \to \mathbb{R}^{n \times n}$
$\forall X \in \mathbb{R}^{n \times n}, y \in \mathbb{R}^n$:

$$< y, \mathcal{A}[X] > =< \mathcal{A}^*[y], X > \tag{2.37}$$
$$y^T diag[X] = tr[(diag^*[y])^T X] \tag{2.38}$$
$$y^T diag[X] = tr[D^T X] \tag{2.39}$$
$$\sum_{i=1}^{n} y_i X_{ii} = \sum_{i=1}^{n} D_{ii} X_{ii} \tag{2.40}$$

Which means we can find an $n \times n$ matrix $D = diag^*[y]$ (its diagonal are $y_1, y_2 ... y_n$ and 0 everywhere else) satisfies the adjoint. Because the adjoint is unique, so this matrix $D = diag^*[y]$ is the adjoint of $diag[X]$.

$$diag^*[y] = \begin{bmatrix} y_1 & 0 & 0 & \dots & 0 \\ 0 & y_2 & 0 & \dots & 0 \\ 0 & 0 & y_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & y_n \end{bmatrix} \tag{2.41}$$

After getting the adjoint map, we can implement the iterative soft-thresholding algorithm by following pseudocode.

Pseudocode:
A solver for this more general problem.

Algorithm 3 Iterative soft-thresholding algorithm with linear map $\mathcal{A}$

```
   Problem: $\min_X \frac{1}{2}\|y - \mathcal{A}[X]\|_2^2 + \lambda\|X\|_1$, given $y \in \mathbb{R}^m$, $\mathcal{A} : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^m$
   Input: $X_0 \in \mathbb{R}^{n_1 \times n_2}$ and step size $\alpha$
1: while $X_k$ not converged $(k = 1, 2, ...)$ do
2:     $W_k \leftarrow X_k - \alpha\mathcal{A}^*[\mathcal{A}[X_k] - y]$
3:     $X_{k+1} \leftarrow soft(W_k, \lambda\alpha)$
4: end while
   Output: $X_* \leftarrow X_k$
```

2.4 Proximal gradient for deconvolution

Now we need to solve deconvolution problem in which we observe $\boldsymbol{Y} = \boldsymbol{D} * \boldsymbol{X_0}$. $\boldsymbol{D}$ is a known signal, $\boldsymbol{X_0}$ is sparse, and $*$ denotes two dimensional convolution. Here, our goal is to find $\boldsymbol{X_0}$. We will find the adjoint of the two dimensional convolution mapping in this section.

$$\min_{\boldsymbol{X}} \frac{1}{2} \|\boldsymbol{Y} - \boldsymbol{D} * \boldsymbol{X}\|_F^2 + \lambda \|\boldsymbol{X}\|_1 \tag{2.42}$$

First of all, let's recall the two dimensional convolution:

$$D * X = \sum_{k,l} D_{k,l} X_{i-k,j-l} = \sum_{k,l} D_{i-k,j-l} X_{k,l} \tag{2.43}$$

Now the linear map is $\boldsymbol{X} \mapsto \mathcal{A}[\boldsymbol{X}] = \boldsymbol{D} * \boldsymbol{X} : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^{n_1 \times n_2}$
The adjoint is $\mathcal{A}^*[\boldsymbol{Y}] : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^{n_1 \times n_2}$
$\forall \boldsymbol{X} \in \mathbb{R}^{n_1 \times n_2}, \boldsymbol{Y} \in \mathbb{R}^{n_1 \times n_2}$:

$$\langle \mathcal{A}[X], Y \rangle_F = \langle X, \mathcal{A}^*[Y] \rangle_F \tag{2.44}$$

Recall that the Frobenius inner product is a binary operation that takes two matrices and returns a number. It is often denoted $\langle \boldsymbol{A}, \boldsymbol{B} \rangle_{\mathrm{F}}$. The operation is a component-wise inner product of two matrices as though they are vectors. The two matrices must have the same dimension—same number of rows and columns—but are not restricted to be square matrices.

$$\langle \boldsymbol{A}, \boldsymbol{B} \rangle_{\mathrm{F}} = \sum_{i,j} \overline{\boldsymbol{A_{ij}}} \boldsymbol{B_{ij}} = tr(\overline{\boldsymbol{A^T}} \boldsymbol{B}) \tag{2.45}$$

Now let's calculate the Frobenius inner product (2.44) to find the adjoint.
Left:

$$\langle \mathcal{A}[X], Y\rangle_F = \sum_{i,j} \overline{\mathcal{A}[X]_{i,j}} Y_{i,j} \tag{2.46}$$

$$= \sum_{i,j}\sum_{k,l} \overline{X_{k,l} D_{i-k,j-l}} Y_{i,j} \tag{2.47}$$

$$= \sum_{i,j}\sum_{k,l} \overline{X_{i,j}}\, \overline{D_{k-i,l-j}} Y_{k,l} \tag{2.48}$$

Right:

$$\langle X, \mathcal{A}^*[Y]\rangle_F = \sum_{i,j} \overline{X_{i,j}} \mathcal{A}^*[Y]_{i,j} \tag{2.49}$$

Since Left = Right, we can get:

$$\mathcal{A}^*[Y]_{i,j} = \sum_{k,l} \overline{D_{k-i,l-j}} Y_{k,l} \tag{2.50}$$

Denote $D'_{i,j} = \overline{D_{-i,-j}}$, then:

$$\mathcal{A}^*[Y]_{i,j} = \sum_{k,l} \overline{D'_{i-k,j-l}} Y_{k,l} \tag{2.51}$$

$$\mathcal{A}^*[Y] = D' * Y \tag{2.52}$$

Where D' is the conjugate of D with $180°$ rotation.

2.5 From a single motif to several motifs

We will need to work with problems in which we have multiple motifs $\boldsymbol{D_1}, ..., \boldsymbol{D_K}$. Let $\boldsymbol{X_1}, ..., \boldsymbol{X_K} \in \mathbb{R}^{n_1\times n_2}$. Define $\boldsymbol{X} = [\boldsymbol{X_1}, ..., \boldsymbol{X_K}] \in \mathbb{R}^{n_1\times n_2\times K}$. Define a mapping $\mathcal{A} : \mathbb{R}^{n_1\times n_2\times K} \to \mathbb{R}^{n_1\times n_2}$, via

$$\mathcal{A}[\boldsymbol{X}] = \sum_{k=1}^{K} \boldsymbol{D}_k * \boldsymbol{X}_k \tag{2.53}$$

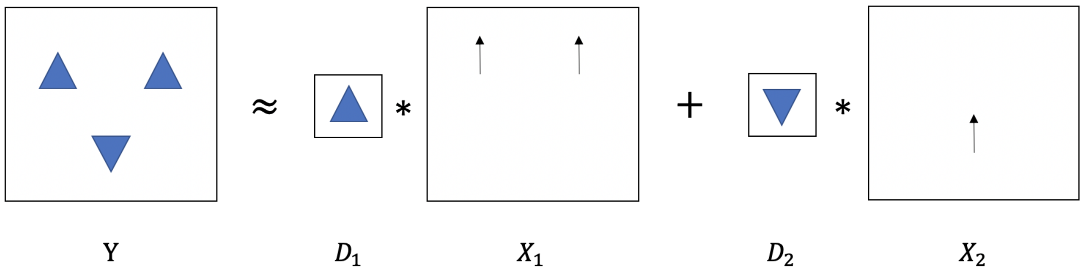


Figure 2: Example of K = 2

The adjoint of this map is a mapping $\mathcal{A}^* : \mathbb{R}^{n_1\times n_2} \to \mathbb{R}^{n_1\times n_2\times K}$. Now let's find what it is.
First of all, let's rewrite $\mathcal{A}[\boldsymbol{X}]$:

$$\mathcal{A}[\boldsymbol{X}] = \sum_{k=1}^{K} \boldsymbol{D}_k * \boldsymbol{X}_k = \sum_{k}\sum_{m,n} \boldsymbol{X}_{m,n,k} \boldsymbol{D}_{i-m,j-n,k} \tag{2.54}$$

Then, use the definition of adjoint:
$\forall \boldsymbol{X} \in \mathbb{R}^{n_1 \times n_2 \times K}, \boldsymbol{Y} \in \mathbb{R}^{n_1 \times n_2}$:

$$\langle \mathcal{A}[X], Y \rangle_F = \langle X, \mathcal{A}^*[Y] \rangle_F \tag{2.55}$$

Left:

$$\begin{aligned}
\langle \mathcal{A}[X], Y \rangle_F &= \sum_{i,j} \overline{\mathcal{A}[X]_{i,j}} Y_{i,j} &(2.56)\\
&= \sum_{i,j} \sum_{m,n,k} \overline{X_{m,n,k} D_{i-m,j-n,k}} Y_{i,j} &(2.57)\\
&= \sum_{i,j} \sum_{m,n,k} \overline{X_{i,j,k}}\, \overline{D_{m-i,n-j,k}} Y_{m,n} &(2.58)
\end{aligned}$$

Right:

$$\langle X, \mathcal{A}^*[Y] \rangle_F = \sum_{i,j,k} \overline{X_{i,j,k}} \mathcal{A}^*[Y]_{i,j,k} \tag{2.59}$$

Since Left = Right, we can get:

$$\mathcal{A}^*[Y]_{i,j,k} = \sum_{m,n} \overline{D_{m-i,n-j,k}} Y_{m,n} \tag{2.60}$$

Denote $D'_{i,j,k} = \overline{D_{-i,-j,k}}$, then:

$$\begin{aligned}
\mathcal{A}^*[Y]_{i,j,k} &= \sum_{m,n} \overline{D'_{i-m,j-n,k}} Y_{m,n} &(2.61)\\
\mathcal{A}^*[Y]_k &= D'_k * Y &(2.62)\\
& &(2.63)
\end{aligned}$$

Where D'_k is the conjugate of D_k with $180°$ rotation.

$$\mathcal{A}^*[Y] = [D'_1 * Y, ..., D'_K * Y] \tag{2.64}$$

2.6 Deconvolution with multiple motifs

In this section, we use the result of the previous item to give a solver for the multi-motif deconvolution problem:

$$\min_{\boldsymbol{X_1},...,\boldsymbol{X_K}} \frac{1}{2} ||\boldsymbol{Y} - \sum_{i=1}^{K} \boldsymbol{D}_i * \boldsymbol{X}_i||_F^2 + \lambda \sum_{i=1}^{K} \|\boldsymbol{X}_i\|_1 \tag{2.65}$$

Regard $\boldsymbol{X} = [\boldsymbol{X_1}, ..., \boldsymbol{X_K}] \in \mathbb{R}^{(n_1 \times n_2) \times K}$ as a 3D matrix array.
The linear map and its adjoint in this problem are:

$$\begin{aligned}
\mathcal{A}[\boldsymbol{X}] &= \sum_{i=1}^{K} \boldsymbol{D}_i * \boldsymbol{X}_i &(2.66)\\
\mathcal{A}^*[\boldsymbol{Y}] &= [\boldsymbol{D'_1} * \boldsymbol{Y}, ..., \boldsymbol{D'_K} * \boldsymbol{Y}] &(2.67)
\end{aligned}$$

Where $D'_{i,j,k} = \overline{D_{-i,-j,k}}$
Recall that soft thresholding operator: $soft(W_i, \lambda) = sign(W_i) max(|W_i| - \lambda, 0)$. Then we can give a solver for this multi-motif deconvolution problem based on the previous result of general linear map.
Pseudocode:
A solver for multi-motif deconvolution problem.

Algorithm 4 Deconvolution with multiple motifs

```
   Problem: $\min_{\boldsymbol{X}=[\boldsymbol{X_1},...,\boldsymbol{X_K}]} \frac{1}{2}||\boldsymbol{Y} - \sum_{i=1}^{K} \boldsymbol{D}_i * \boldsymbol{X}_i||_F^2 + \lambda \sum_{i=1}^{K} \|\boldsymbol{X}_i\|_1$
   Given: $Y \in \mathbb{R}^{n_1 \times n_2}$, $\mathcal{A}[\boldsymbol{X}] = \sum_{i=1}^{K} \boldsymbol{D}_i * \boldsymbol{X}_i$, $\mathcal{A}^*[\boldsymbol{Y}] = [\boldsymbol{D_1'} * \boldsymbol{Y}, ..., \boldsymbol{D_K'} * \boldsymbol{Y}]$
   Input: $X^0 \in \mathbb{R}^{n_1 \times (n_2 \times K)}$ and step size $\alpha$
1: while $X^n$ not converged $(n = 1, 2, ...)$ do
2:     $W^n \leftarrow X^n - \alpha \mathcal{A}^*[\mathcal{A}[X^n] - Y]$
3:     $X^{n+1} \leftarrow soft(W^k, \lambda\alpha)$
4: end while
   Output: $X^* \leftarrow X^n$
```

2.7 Compressed sensing with line scans

Now we consider about the line scans, recall the Lasso problem (1.2):

$$\min_{\boldsymbol{X} \geq \boldsymbol{0}, \boldsymbol{p} \in \boldsymbol{\mathcal{P}}} \sum_{i=1}^{m} \frac{1}{2} ||\boldsymbol{R}_i - \mathcal{S}\{\psi(p_i) * \mathcal{L}_{\theta_i}[\sum_{k=1}^{K} \boldsymbol{D}_k * \boldsymbol{X}_k]\}||_2^2 + \sum_{i,j,k} \lambda_{i,j,k}^{(l)} \boldsymbol{X}_{i,j,k}, \tag{2.68}$$

To make it simple, we denote the line scanning process, point spread function (PSF) and sampling as line projection operator $\mathcal{L}_p$, whose adjoint is back projection $\mathcal{L}'_p$. We rewrite the smooth part $h(\boldsymbol{X}, \boldsymbol{p})$ as:

$$h(\boldsymbol{X}, \boldsymbol{p}) = \frac{1}{2} ||\boldsymbol{R} - \mathcal{L}_p[\sum_{k=1}^{K} \boldsymbol{D}_k * \boldsymbol{X}_k]||_2^2 \tag{2.69}$$

Then we calculate the derivative of smooth function $h(\boldsymbol{X}, \boldsymbol{p})$:

$$\partial_{\boldsymbol{X}} h(\boldsymbol{X}, \boldsymbol{p}) = D' * \mathcal{L}'_p[\mathcal{L}_p[\sum_{i=1}^{K} \boldsymbol{D}_i * \boldsymbol{X}_i] - \boldsymbol{R}], \tag{2.70}$$

$$\partial_p h(\boldsymbol{X}, \boldsymbol{p}) = J'_p(\mathcal{L}_p[\sum_{i=1}^{K} \boldsymbol{D}_i * \boldsymbol{X}_i] - \boldsymbol{R}), \tag{2.71}$$

where J_p is the Jacobian of $\mathcal{L}_p[\sum_{i=1}^{K} \boldsymbol{D}_i * \boldsymbol{X}_i]$ w.r.t parameter $\boldsymbol{p}$.
Now we optimize both the parameter p and the sparse maps $\boldsymbol{X}$ via alternating minimization using Accelerated iPalm, see Algorithm 5.

Moreover, to cope with the coherence phenomenon, we adopt the reweighting scheme by solving Lasso formulation multiple times while updating penalty variable λ in each iterate, see Algorithm 6. This is the formal algorithm to solve (2.68).

Algorithm 5 iPalm($\boldsymbol{X}_{init}, \boldsymbol{p}_{init}, \boldsymbol{\lambda}, h, \mathcal{P}$): Accelerated iPalm for calibrating sparse regression

```
Require: Initialization $\boldsymbol{X}_{init} \in \mathbb{R}^{n_1 \times n_2 \times k}$ and $\boldsymbol{p}_{init} \in \mathcal{P}$, sparse penalty $\boldsymbol{\lambda} \in \mathbb{R}^{n_1 \times n_2 \times k}$, smooth function $h$, and number of iterations $L$.
Let $\boldsymbol{X}^{(0)} \leftarrow \boldsymbol{X}_{init}$; $\boldsymbol{p}^{(0)} \leftarrow \boldsymbol{p}_{init}$; $\alpha \leftarrow 0.9$; $t_{X0}, t_{p0} \leftarrow 1$
  for $l = 1, ..., L$ do
    // Accelerated Proximal Gradient for map $\boldsymbol{X}$.
    $\boldsymbol{Y}^{(l)} \leftarrow \boldsymbol{X}^{(l)} + \alpha(\boldsymbol{X}^{(l)} - \boldsymbol{X}^{(l-1)}); t \leftarrow t_{X0};$
    repeat
      $t \leftarrow t/2$;
      $\boldsymbol{X}^{(l+1)} \leftarrow Soft^{+}_{t\boldsymbol{\lambda}}[\boldsymbol{Y}^{(l)} - t\partial_{\boldsymbol{X}} h(\boldsymbol{Y}^{(l)}, \boldsymbol{p}^{(l)})]$;
    until $h(\boldsymbol{X}^{(l+1)}, \boldsymbol{p}^{(l)}) \leq h(\boldsymbol{Y}^{(l)}, \boldsymbol{p}^{(l)}) + \langle \partial_{\boldsymbol{X}} h(\boldsymbol{Y}^{(l)}, \boldsymbol{p}^{(l)}), \boldsymbol{X}^{(l+1)} - \boldsymbol{Y}^{(l)} \rangle + \frac{1}{2t} \left\| \boldsymbol{X}^{(l+1)} - \boldsymbol{Y}^{(l)} \right\|_2^2$;
    $t_{X0} \leftarrow 4t$;
    // Accelerated Proximal Gradient for parameters $p$.
    $\boldsymbol{q}^{(l)} \leftarrow \boldsymbol{p}^{(l)} + \alpha(\boldsymbol{q}^{(l)} - \boldsymbol{q}^{(l-1)}); t \leftarrow t_{p0}$;
    repeat
      $t \leftarrow t/2$;
      $\boldsymbol{p}^{(l+1)} \leftarrow Proj_{\mathcal{P}}[\boldsymbol{q}^{(l)} - t\partial_{\boldsymbol{p}} h(\boldsymbol{X}^{(l+1)}, \boldsymbol{q}^{(l)})]$;
    until $h(\boldsymbol{X}^{(l+1)}, \boldsymbol{p}^{(l+1)}) \leq h(\boldsymbol{X}^{(l+1)}, \boldsymbol{q}^{(l)}) + \langle \partial_{\boldsymbol{p}} h(\boldsymbol{X}^{(l+1)}, \boldsymbol{q}^{(l)}), \boldsymbol{p}^{(l+1)} - \boldsymbol{q}^{(l)} \rangle + \frac{1}{2t} \left\| \boldsymbol{p}^{(l+1)} - \boldsymbol{q}^{(l)} \right\|_2^2$;
    $t_{p0} \leftarrow 4t$;
  end for
Ensure: $(\boldsymbol{X}^{(L)}, \boldsymbol{p}^{(L)})$ as the approximated minimizers of $\min_{\boldsymbol{X} \geq \boldsymbol{0}, \boldsymbol{p} \in \boldsymbol{\mathcal{P}}} h(\boldsymbol{X}, \boldsymbol{p}) + \sum_{i,j,k} \lambda_{i,j,k} \boldsymbol{X}_{i,j,k}$,
```

Algorithm 6 Reweighted iPalm

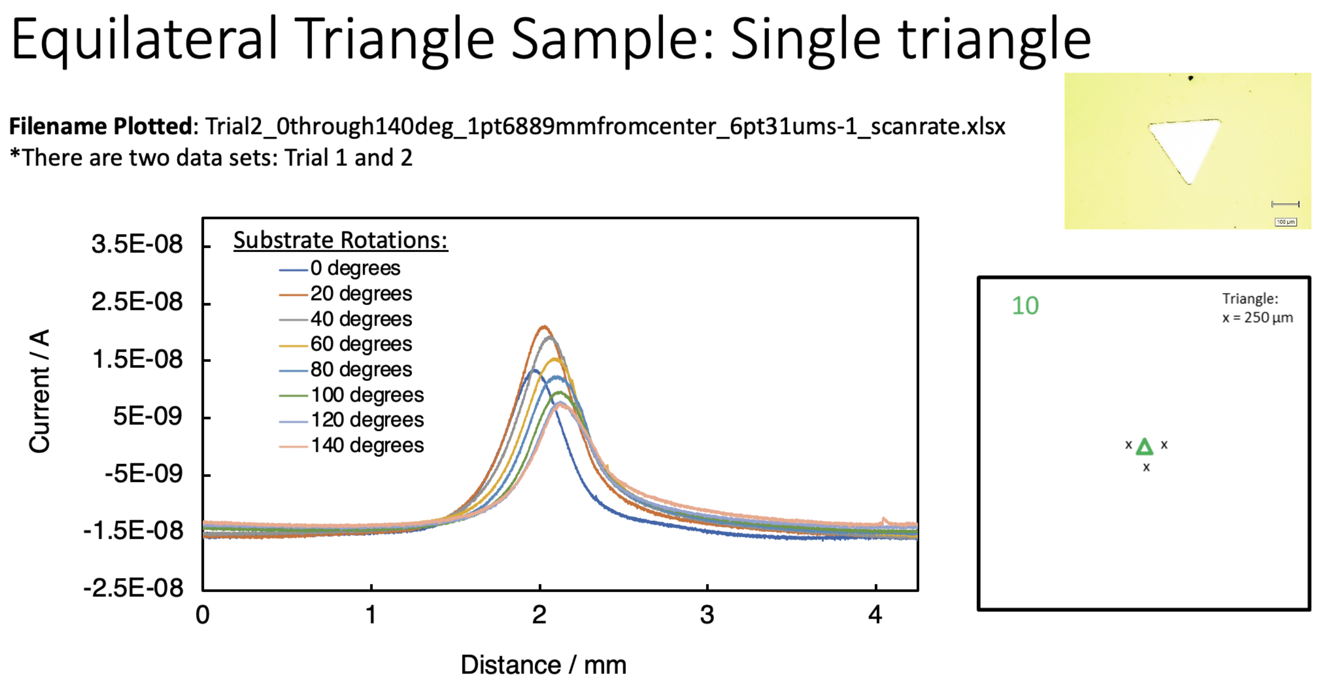
```
Require: Line scans $\{\boldsymbol{R}_i\}_{i=1}^m$, scan angles $\{\boldsymbol{\theta}_i\}_{i=1}^m$, dictionary $\boldsymbol{D}$, estimated PSF $\hat{\psi}$, initial guess of parameters $\boldsymbol{p}_{init} \in \mathcal{P}$ convex, number of iterations $L$, and adjustment constant $C_1, C_2$.
Let $\boldsymbol{X}^{(0)} \leftarrow \boldsymbol{0}$, $\boldsymbol{p}^{(0)} \leftarrow \boldsymbol{p}_{init}$,
Let $h(\boldsymbol{X}, \boldsymbol{p}) \leftarrow \sum_{i=1}^m \frac{1}{2} ||\boldsymbol{R}_i - \hat{\psi} * \mathcal{L}_{\theta_i}[\boldsymbol{p}][\sum_{k=1}^K \boldsymbol{D}_k * \boldsymbol{X}_k]||_2^2$;
  for $l = 1, ..., L$ do
    if $l = 1$ then
      $\boldsymbol{\lambda} \leftarrow C_1 max_{ijk}\{pos(D' * \mathcal{L}'_p[\boldsymbol{R}])\}$;
    else
      $\boldsymbol{\lambda}_{ijk}^{(l)} \leftarrow \frac{C_2 h(\boldsymbol{X}^{(l)}, \boldsymbol{p}^{(l)})}{\boldsymbol{X}_{ijk}^{(l-1)} + \epsilon}$
    end if
    $(\boldsymbol{X}^{(l+1)}, \boldsymbol{p}^{(l+1)}) \leftarrow iPalm(\boldsymbol{X}^{(l)}, \boldsymbol{p}^{(l)}, \boldsymbol{\lambda}, h, \mathcal{P})$;
  end for
Ensure: Reconstructed image $\boldsymbol{Y} \leftarrow \sum_{k=1}^K \boldsymbol{D}_k * \boldsymbol{X}_k^{(L)}$
```

3 Experiment

In this section, we present two groups of real data experiments results, and also make comparisons on Algorithm 5 and Algorithm 6.

3.1 Single triangle

First of all, here is the data information provided by Anna:



Equilateral Triangle Sample: Single triangle

Filename Plotted: Trial2_0through140deg_1pt6889mmfromcenter_6pt31ums-1_scanrate.xlsx
*There are two data sets: Trial 1 and 2

Figure 3: Single triangle data information

We input this data, read the line scans and run back projection successfully.

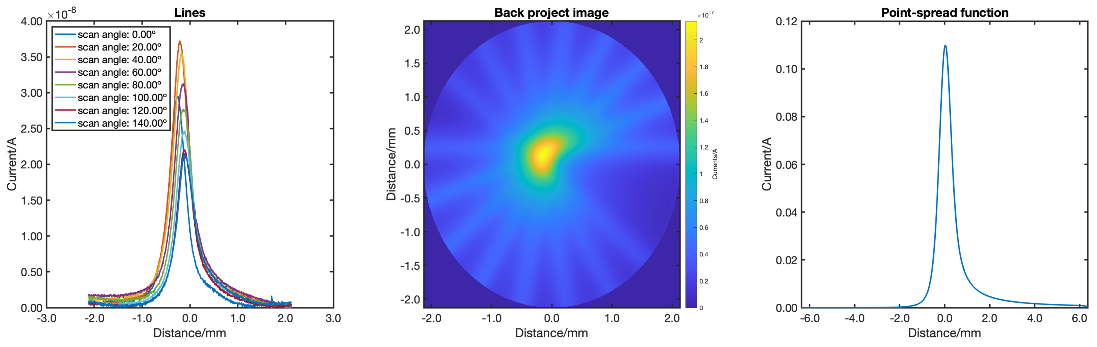


Figure 4: Lines, BP image, and PSF

For this data set, we set adjustment constant $C_1 = 0.2$ and $C_2 = 1 \times 10^{-5}$. Figure 5 shows the results of iPalm (algorithm 5) with 50 iterations. Figure 6 shows the results of Reweighted iPalm (algorithm 6) with 15 iterations and each iteration contains 30 iPalm iterations.

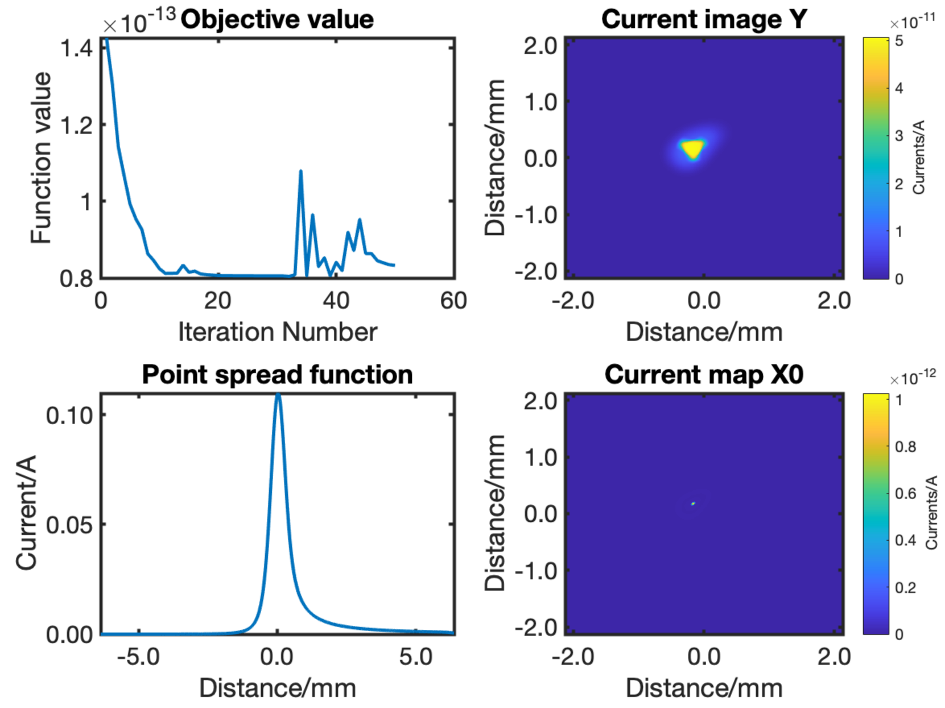


Figure 5: Reconstruction result of iPalm (single triangle)

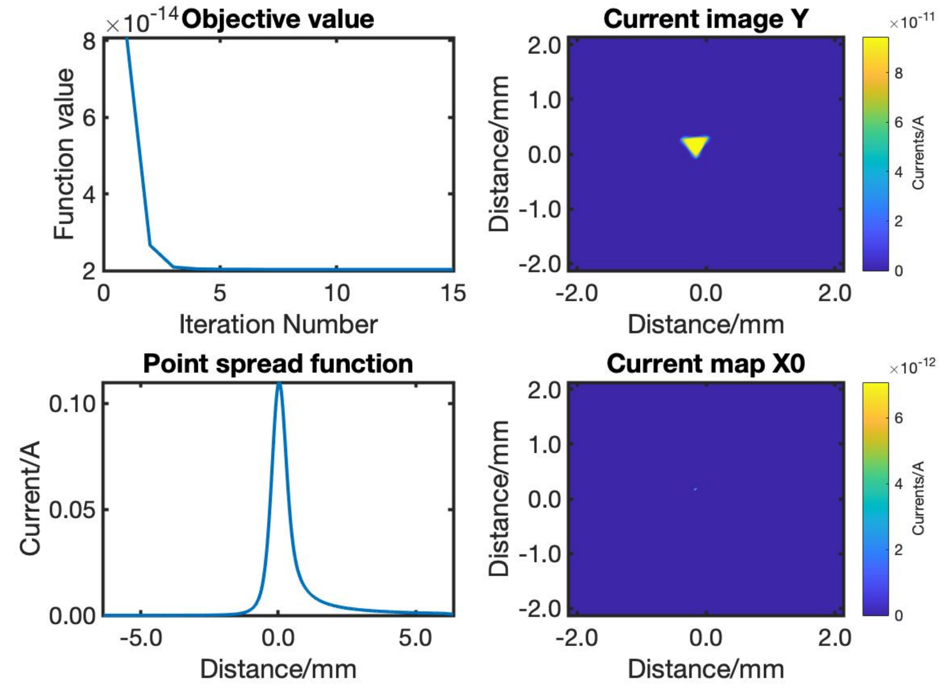
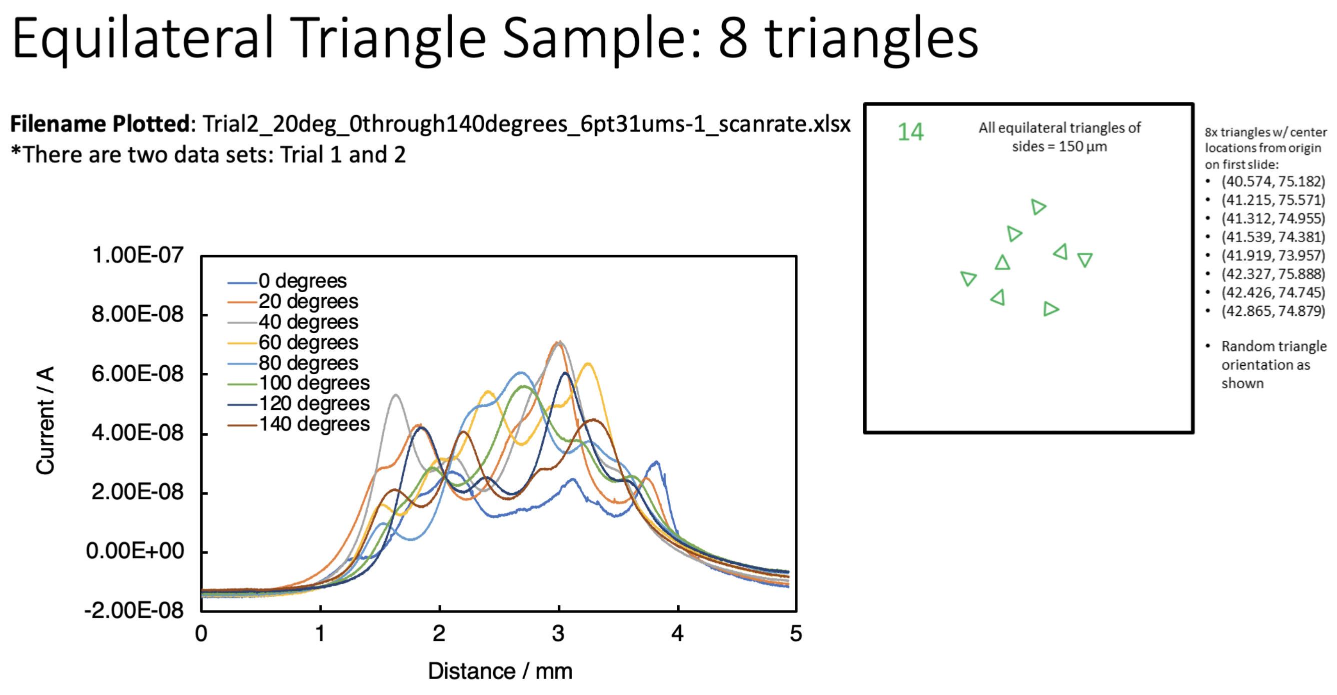


Figure 6: Reconstruction result of reweighted iPalm (single triangle)

3.2 Multiple triangles

We take 8 triangles data as an example, here is the data information:



Equilateral Triangle Sample: 8 triangles

Filename Plotted: Trial2_20deg_0through140degrees_6pt31ums-1_scanrate.xlsx
*There are two data sets: Trial 1 and 2

14

All equilateral triangles of sides = 150 µm

8x triangles w/ center locations from origin on first slide:

- (40.574, 75.182)
- (41.215, 75.571)
- (41.312, 74.955)
- (41.539, 74.381)
- (41.919, 73.957)
- (42.327, 75.888)
- (42.426, 74.745)
- (42.865, 74.879)

- Random triangle orientation as shown

Figure 7: 8 triangles data information

Also, we input this data, read the line scans and run back projection successfully.

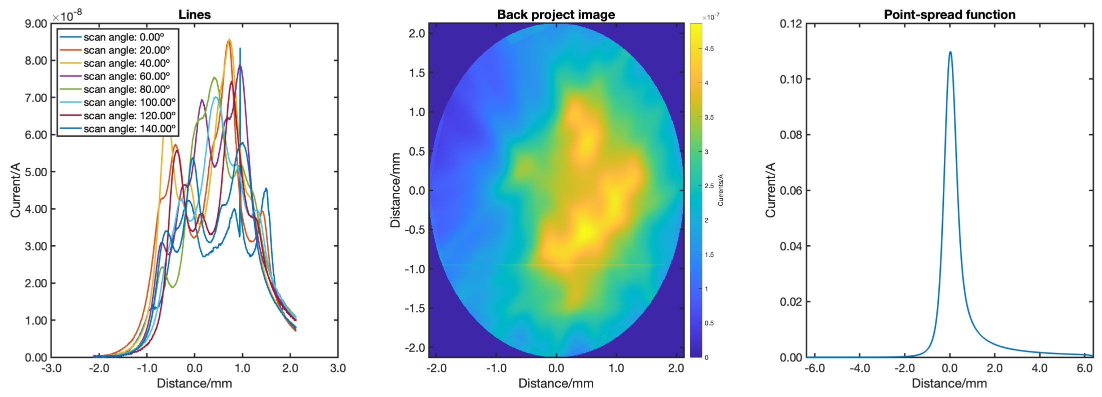


Figure 8: Lines, BP image, and PSF

For this data set, we set adjustment constant $C_1 = 0.2$ and $C_2 = 1.225 \times 10^{-6}$. Figure 9 shows the results of iPalm (algorithm 5) with 50 iterations. Figure 10 shows the results of Reweighted iPalm (algorithm 6) with 50 iterations and each iteration contains 30 iPalm iterations. We can see that reweighting is helpful to cope with the coherence phenomenon. The reconstruction is successful based on the visual quality.

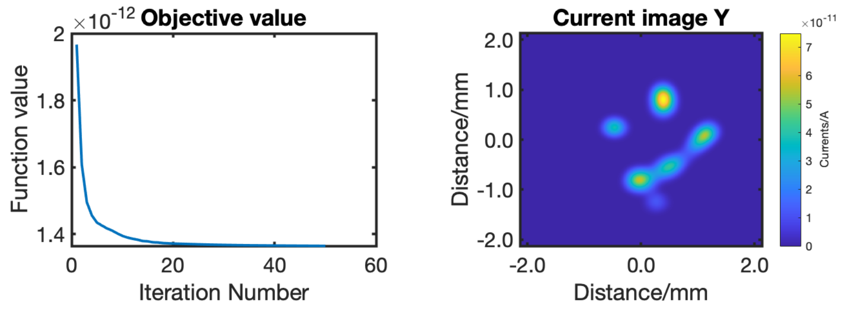
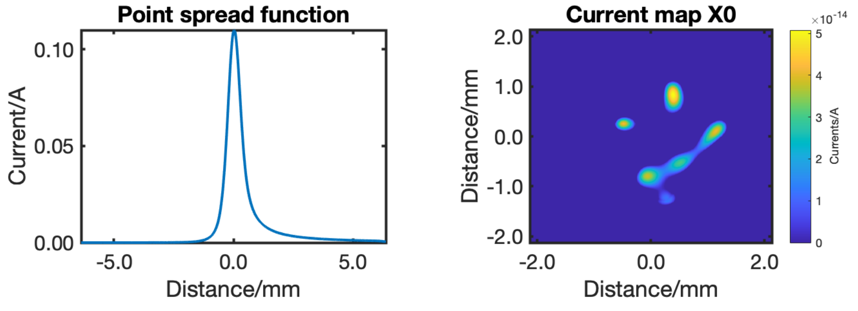



Figure 9: Reconstruction result of iPalm (8 triangles)

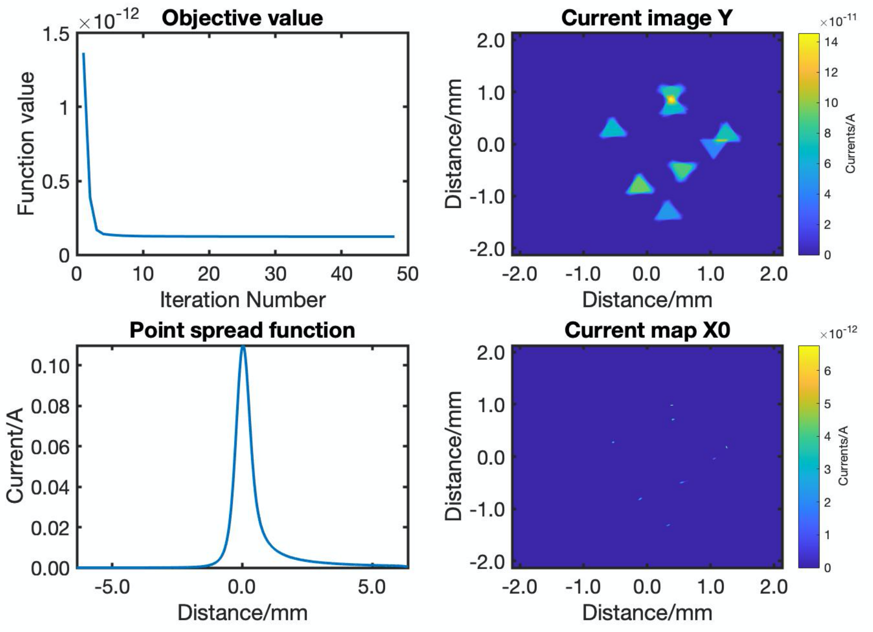


Figure 10: Reconstruction result of reweighted iPalm (8 triangles)