SLAP Demo 1:
Intro to Computational Imaging and Regularization

Kaspar Podgorski

Introduction

Scanned Line Angular Projection Microscopy (SLAP) is a new high-speed two-photon imaging technique that scans lines of illumination across the sample, rather than a point focus. The resulting measurements are linear projections of the 2D sample plane along different angles. SLAP imaging is much faster than raster scanning because it collects only four projection angles of the sample. It collects all the data for each frame with just four line scans, achieving >1 kHz frame rates for megapixel fields of view.

SLAP recordings require computational processing to turn the four angular projections into pixel-space images, making SLAP a **computational imaging** technique. Examples of other computational imaging techniques are Lightfield Microscopy (LFM), Super-resolution microscopy (*e.g.* PALM, STORM, SOFI), Multiview Imaging, some forms of Point-Spread-Function Engineering, and Structured Illumination Microscopy (SIM).

A commonly-used framework for computational imaging reconstructions is **maximum-likelihood (ML) reconstruction**, which finds an image that maximizes the probability of the measurements given some model of the sample and imaging process. A common algorithm for this is **Richardson-Lucy deconvolution (RL)**, which performs ML reconstruction under a Poisson noise model and can easily be extended to incorporate statistical priors on the underlying images. RL deconvolution is commonly used for LFM, Multiview Imaging, PSF Engineering, and SLAP. This workshop is a practical introduction to RL in the context of SLAP imaging.

Useful SLAP links:

- bioRxiv preprint on SLAP imaging
- a github repo for SLAP imaging code (github.com/KasparP/SLAP)
- HHMI Janelia Open Science site for SLAP microscope designs

Getting Started

1) Download the following directory from Dropbox to get the MATLAB code and example data: https://www.dropbox.com/sh/0f1w4go1s5cnlvd/AABzQRKbAAr3B4y8Lgw9JZ_ba?dl=0

2) Navigate MATLAB to this directory, or add it to your path

The Workshop

I. Particle Imaging

We'll start by looking at a raster image of fluorescent particles and some SLAP data of the same field of view, then reconstruct the SLAP measurements and compare them to the raster image.

Examine the data

Load the 'demoData.mat' file provided with the demo, which will add the variables **`P2D`**, **`scandata`**, and **`ref2D`** to your workspace.

`P2D` is sparse representation of the microscope's measurement matrix projected down to 2 dimensions for simplicity here (in real applications, SLAP reconstructions are 3D). Rows of **`P2D`** correspond to images of the line focus at particular times/positions during each frame. Columns of **`P2D`** correspond to how each pixel in the image is sampled by the measurements. The covariance matrix **`P2D*P2D`**' is interesting. It's a visualization of how different pixels become mixed together in measurements. Projection imaging methods, like SLAP, mix pixels together to image faster. Tomographic measurements are good for this because they mix any pair of pixels together only very weakly; the diagonal of **`P2D*P2D`**' is much larger than any element of the off-diagonal. This improves computational image reconstruction. To visualize a subset of the line positions, and the covariance matrix **`P2D*P2D`**', you can use the code:

```
demo_visualizeP(P2D)
```

`scandata` is a short recording produced by our SLAP microscope. This recording is 49 frames of a static sample of beads. Frames consist of several thousand measurements each, which correspond to different positions and angles of the line focus. You can plot the measurements corresponding to a single frame this way:

```
y = [scandata.frames.pmtData];
figure, plot(y(:,1)); xlabel('Measurement #'); ylabel('photons')
```

Each frame consists of the 4 projection axes appended in sequence. You can separate them like so:

```
figure, imagesc(reshape(mean(y,2), [],4));
xlabel('position'); ylabel('Projection axis')
```

`Ref2D` is a 2D raster image of the same sample. You can visualize it using the code:

```
figure, imagesc(ref2D); axis image;
```

Solve data with default options

Run this code and accept the default options in the GUI that pops up:

```
demo_solve(P2D,scandata,ref2D)
```

This produces a pixel-space reconstruction of the particles from the SLAP data and creates several plots. These plots illustrate the process of RL deconvolution. One of the plots, the backprojection (**`y'*P2D`**), is simply an overlay of the corresponding line PSF for every photon that was detected. Early iterations of RL look a lot like the backprojection, but later iterations become sharper, and look more like the ground truth shown in the raster image.

RL deconvolution is an optimization algorithm: each iteration decreases a **loss function**, the evolution of which is plotted in one of the figures. The loss function that RL optimizes is proportional to the negative log probability of the image under Poisson statistics. Each RL update changes the image in a direction related to the local gradient of the image probability. If RL converges, the resulting image is a local maximum of the probability function.

Try removing the SLM prior

When this sample was imaged, a spatial light modulator (SLM) was used to block excitation outside a circle inscribed in the field of view, and we know that nearly all fluorescence came from within this circle. Separately, we know that the sample is sparse, so the brightness along projections with very few photon counts is likely to be zero.

This information is incorporated into this solver by initializing the reconstruction to zero in regions where we know there is no fluorescence. Because the Richardson-Lucy solver only performs multiplications, these regions will remain 0 in the solution. It's interesting to see what happens when we don't use this information. Try turning it off in the options.

Try changing lambda

Lambda is an **L1 regularization** parameter. It adds a cost to the loss function that is linearly proportional to the total brightness of the reconstructed image. Addition of L1 regularization to most loss functions favors a sparser solution. Reconstructions can be extremely sensitive to lambda, and using this kind of regularization for computational imaging requires smart ways to select such parameters. If you have time, try tuning lambda by hand so it improves the reconstruction to your eye. This should be possible, but it's tricky!

Try changing the dark rate

The dark rate is a parameter in the model that determines the distribution of photon detections when there is no sample. Without dark noise, the model expects 0 photons in regions where the sample is dark, and a stray photon is infinitely unlikely! The **noise model** is a very important consideration for computational imaging techniques, because real data have multiple sources of noise, and getting the model wrong can alter your results substantially. The solver in this demo uses a simple dark noise model where stray ('dark') photons arrive independently at the detector with a fixed rate. Try increasing it, or decreasing it to 0.

II. Simulated Images

Next we will simulate synthetic images and see how well they are reconstructed with SLAP**.**

You can generate and solve simple particle images with the function:

```
    demo_simulatePoints(P2D);
```

This produces the same figures as earlier, but now with artificial data. First run the code with the default options, and look at the plot of the reconstruction loss. With artificial data, we now also plot the loss of the ground truth image as a black line. You should see the solver approach the ground truth value asymptotically.

Now try decreasing the brightness to 100. The reconstruction loss when the solver 'converges' is lower than the ground truth! This happens even if you initialize the solver with the correct solution (if you have time, try setting this option to `True` in the GUI). What does this mean? If the loss is higher than the ground truth, it can mean the solver found a local optimum of the loss function, but not the global optimum. If the loss is lower than the ground truth, it means that the solver is fitting closely to the noise in the simulated measurements, a phenomenon known as **overfitting**. These can both cause problems, and both can be reduced with appropriate **regularization**, which changes the shape of the loss function and can penalize noisy solutions or smooth out local optima. L1 regularization is one option, but SLAP uses a different regularization approach when performing activity imaging, which we'll discuss next.

Before we go on, try playing with the simulation parameters, like increasing **`nParticles`**, until the reconstructions break down. How many particles can be accurately reconstructed in this field of view? How does this depend on sample brightness?

III. Segmentation-based reconstruction

The real power of SLAP comes when a sample has known structure, because we can use that structure for very effective regularization. Imagine you are imaging the activity of a set of neurons over seconds or minutes. The spatial structure of those neurons doesn't change much (here we'll ignore sample movement, which it turns out is easy to measure and correct). If we take an image of that structure in advance, we can use that information to better reconstruct the things that do change- like activity-induced brightness changes! - from limited data. This is a different kind of regularization, and it's very powerful.

To demonstrate this, we'll simulate a toy example- a case where field of view contains donuts of various brightnesses. We know where the donuts are, but not how bright they are in any given frame. We'll compare reconstructions that use this **prior information** to the pixel-space reconstructions we were performing before, which don't.

You can simulate data and compare the solvers with the function:

```
demo_compareSolvers(P2D);
```

This generates several plots. You can compare the reconstructed images using the two methods, and evaluate overfitting in the convergence plots. Finally, a scatter plot is produced that compares the actual vs recovered brightness of each donut.

Take some time to explore how these solvers compare for different numbers of donuts and brightnesses. How many donuts can be tracked in this field of view when we have a segmentation?

If you want to simulate the same pattern of donuts while changing other parameters, you can fix MATLAB's random number generator before each function call with the command

```
rng(42);
```

That's it!

This workshop was meant to give you some core intuitions into how RL deconvolution and SLAP imaging work. Many realities of imaging are not modeled here. The RL-based solvers used for SLAP imaging incorporate a few other subtleties, such as knowledge of sample dynamics, richer SLM patterns, data alignment and motion correction, 3D point-spread-functions and reference images, acceleration and damping of RL updates, gradient clipping, and other solver algorithms, to name a few. Our github repo for SLAP imaging code is at github.com/KasparP/SLAP.

If you found this interesting, have a comment, found a bug, or want to contribute, please contact me:

Kaspar Podgorski

podgorskik@janelia.hhmi.org