Article

PySOFI: an open source Python package for SOFI

Yuting Miao[1,o], Shimon Weiss[1,2,3,+], and Xiyu Yi[4,*]

[1]Department of Chemistry and Biochemistry, University of California, Los Angeles 90095, USA
[2]Department of Physiology, University of California, Los Angeles 90095, US
[3]Department of Physics, Institute for Nanotechnology and Advanced Materials, Bar-Ilan University,Ramat-Gan 52900, Israel
[4]Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, CA 94550, USA
[+]co-correspondence: sweiss@chem.ucla.edu
[o]co-correspondence: ytmiao@ucla.edu
[*]correspondence: yi10@llnl.gov

ABSTRACT Super-resolution optical fluctuation imaging (SOFI) is a highly democratizable technique that provides optical super-resolution (SR) without requirement of sophisticated imaging instruments. An open source package for SOFI algorithm is needed to support not only the utilization of SOFI, but also the community adoption and participation for further development of SOFI. In this work, we developed *PySOFI*, an open source python package for SOFI analysis that offers the flexibility to inspect, test, modify, improve and extend the algorithm. We provide a complete documentation for the package and a collection of Jupyter Notebooks to demonstrate the usage of the package. We discuss the architecture of *PySOFI*, illustrate how to use each functional module. A demonstration on how to extend the *PySOFI* package with additional module is also included in the *PySOFI* package. We expect *PySOFI* to facilitate efficient adoption, testing, modification, dissemination and prototyping of new SOFI-relevant algorithms.

1. INTRODUCTION

Super-resolution optical fluctuation imaging (SOFI) [1] is a widely used optical super-resolution method applicable for a broad range of conditions, where sophisticated control on the instrument and sample preparations are not required. It has attracted a growing community of active practitioners and developers over a decade. The advancements utilizing this technology include innovations in blinking dyes and fluorescent proteins, sample preparation [2, 3, 4, 5, 6], illumination schemes, experiment designs, data processing methods [7, 8, 9, 10, 11, 12, 13, 14, 15], and integration with other methods [16, 17, 18, 19, 20, 21, 22, 23].

SOFI is compatible with simple wide-field imaging system to acquire image stacks of samples that exhibits optical fluctuation of fluorescence signal. It is assumed that the position of the signal sources are static over the time course of acquisition, and the optical fluctuations can be induced by either the stochastic blinking of the fluorophores, the diffusion of and stochastic binding of fluorophores to static binding sites, fluctuation of the scatters [24, 25].

Various of prior studies have provided detailed explanations on the SOFI principles [1, 8, 7, 24, 26], and interested readers are recommended to reference [27] for insights about moments, cumulants and their interplay. Here we provide a brief review on the theory of SOFI processing. Given a sample with N emitters that blink independently with a binary fluorescence intensity profile constituting a fluorescence 'on' stage and a fluorescence 'off' state, the fluorescence signal captured at a given camera pixel located at $\vec{r}$ and time t is

$$F(\vec{r},t) = \sum_{k=1}^{N} \epsilon_k b_k(t) U(\vec{k} - \vec{r_k}), \tag{1}$$

where k is the emitter index, $\vec{r_k}$ is the location of the k^{th} emitter, U is the point spread function of the imaging system, ϵ_k is the constant 'on'-state brightness, and $b_k(t)$ is the time-dependent stochastic blinking profile that is either 0 or 1. $b_k(t)$ equals to 1 when the emitter is in the 'on' state, and 0 when the emitter is in the 'off'-state.

The first step of SOFI is to calculate the fluctuation of fluorescence signal around the temporal average for each pixel

$$\delta F(\vec{r},t) = F(\vec{r}) - \langle F(\vec{r}) \rangle_t = \sum_{k=1}^{N} \epsilon_k \delta b_k(t) U(\vec{k} - \vec{r_k}). \tag{2}$$

The work from X. Yi was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Release number: LLNL-JRNL-827376

Here both ϵ_k and U are constant, so only the fluctuation in the blinking profile δb_k would affect δF. Then, the correlation functions can be calculated as follows:

$$G_n(\vec{r_1}, \vec{r_2}, \cdots, \vec{r_n}; \tau_1, \tau_2, \cdots, \tau_{n-1}) = \langle \delta F(\vec{r_1}, t) \delta F(\vec{r_2}, t+\tau_1) \cdots \delta F(\vec{r_n}, t+\tau_{n-1}) \rangle_t. \tag{3}$$

Equation 3 represents the auto-correlation function when $\vec{r} = \vec{r_1} = \vec{r_2} = \cdots = \vec{r_n}$, and cross-correlation function otherwise. Replacing $\delta F(\vec{r_i}, \tau_{i-1})$ with δF_i, equation 3 can be simplified as shown below:

$$G_n(\delta F_1, \delta F_2, \cdots, \delta F_n) = \langle \delta F_1 \delta F_2 \cdots \delta F_n) \rangle_t. \tag{4}$$

Here $G_n(\delta F_1, \delta F_2, \cdots, \delta F_n)$ indicates the joint-moment of the set $\{\delta F_i | i \in [0, 1]\}$. Next, the n^{th} order joint-cumulant of the set, $C_n(\delta F_1, \delta F_2, \cdots, \delta F_n)$, can be derived from joint-moments and joint-cumulants of lower orders based on a recursive relation (see section 3.1). Note here that the joint-moment and joint-cumulnat are generalized terms of correlation functions and cumulants calculated from either auto-correlations or cross-correlations with different choices of τ_i values and pixel combinations[11].

The n^{th}-order cumulant functions can also be addressed based on fluorescence fluctuation of a multi-emitter system:

$$C_n(\vec{r}, \tau_1, \tau_2, \cdots, \tau_{n-1}) = \sum_{k=1}^{N} \epsilon_k^n \omega_{n,k}(\tau_1, \tau_2, \cdots, \tau_{n-1})) U^n(\vec{k} - \vec{r_k}), \tag{5}$$

where $\omega_{n,k}(\tau_1, \tau_2, \cdots, \tau_{n-1}))$ equals to n^{th}-order cumulant of $\delta b_k(t)$. Detailed derivation can be found at [1, 8, 11, 26]. With n^{th} order cumulant analysis, the theoretical resolution improvement of SOFI is $\sqrt{n}$ fold. Such improvement increases to n when combined with deconvolution, presenting a great potential for further advancements for SOFI [1].

However, there are imperfections in high-order SOFI cumulants (e.g., cusp-artifacts [26]), which can be explained under the framework of *virtual emitter* interpretation[26]). Specifically, comparing the similarity between Equation 1 and Equation 5. The high order SOFI cumulant image can be perceived as an imaged captured from a microscope with a PSF that is equivalent to the n^{th} power of the original PSF (compare the terms that contains U between Equation 1 and 5), with emitters located a the same location as in the original sample, but with emitter brightnesses replaced into $\epsilon_k^n \omega_{n,k}$. Because $\omega_{n,k}$ is the n^{th} order cumulant of the blinking profile of the k^{th} emitter, its value can be either positive or negative, which would introduce a cusp artifacts in the SOFI cumulant images when we display the absolute value of an image with adjacent positive and negative virtual emitters. We also demonstrated how the validity of one of the most widely used SOFI processing method, bSOFI [7], is negatively impacted. But such findings have not receive common awareness as of yet.

We believe an insightful and thorough understanding of the method is crucial to ensure solid advancements in both SOFI and SOFI-relevant innovations. However, for new investigators without prior experience with SOFI analysis, there is often a steep learning curve to fully understand, modify, and extend the existing open-source packages [7, 28]. The existing SOFI analysis routines are implemented in ImageJ [29], MATLAB [30, 28] or Igor Pro [28]. ImageJ requires professional programming skills if customization and modifications are required, while MATLAB and Igor Pro require paid licenses. Such limitations present a greater challenge for new investigators who are interested in joining the SOFI community but prefer not to use the existing packages blindly.

Here we present *PySOFI*, an open source package for SOFI analysis implemented in Python. Benefited from the active open-source community and the abundance of free learning materials for Python, *PySOFI* offers an easy option for investigators interested in adopting the SOFI algorithm. *PySOFI* focuses on engaging the community and is designed to be simple, modular, and highly customizable. *PySOFI* is hosted on GitHub to facilitate utilization, improvements, and continuous maintenance by the interested users and developers. A collection of examples is provided in the form of Jupyter Notebooks. One can use *PySOFI* to explore and characterize SOFI analysis, validate the results from the prior studies, and gain insights through exploration. *PySOFI* is also useful for the prototyping of new methods to extend SOFI algorithm. Similar Jupyter Notebooks can be adapted to promote the new methods and improve the reproducibility of the results. We expect *PySOFI* to appeal to both beginners and experts, it facilitates innovations where modification and extensions are required, and further promote the scientific advancements among scientists interested in SOFI.

The rest of the manuscript is organized as follows. Section 2 provides an overview of the *PySOFI* package. Section 3 discusses the *PySOFI* software architecture design and analysis pipeline, together with analysis examples for various of modules. Section 4 summarizes the work and discusses future directions.

2. *PYSOFI* OVERVIEW

We designed a straightforward architecture for the *PySOFI* package. As shown in (figure 2), *PySOFI* contains eight independent function modules (in the `functions` folder) and one data class (`PysofiData`). A detailed description of *PySOFI* is available in our online documentation. To get started with the installation, the user can follow this page.

Figure (1) provides the data-flow diagram that demonstrate the connections (arrows) between different processing steps (green squares) and different types of data (purple ovals). Three collections of SOFI analysis routines are implemented in the `PysofiData` class, including the "Shared Processes" that contains the traditional SOFI analysis steps [1], the "SOFI 2.0" collection that contains the routines for SOFI 2.0 processing [11]. In the "Shared Processes" block, the processing steps including bleaching correction (BC), Fourier interpolation (FI), and moment and cumulant calculations. The processing steps can be performed in various sequences (green arrows). In the "SOFI 2.0" block, one can perform noise filtering and local dynamic range compression (ldrc) on the image. In the data processing workflow, one can save and load the intermediate results for each processing step (purple arrows). For example, in the "Shared Processes" collection, the intermediate results (purple ovals) can be saved as separate new TIFF files or stored as attributes in the `PysofiData` class object, and then passed to another processing step (purple arrow).

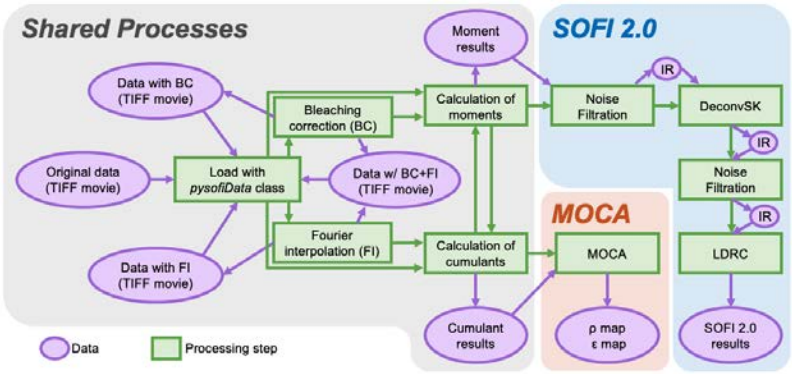

Figure 1: **Data-flow diagram for *PySOFI*.** Three collections of SOFI analysis routines are implemented in *PySOFI* as depicted in the diagram: Shared Processes, SOFI 2.0 analysis, and MOCA analysis. Green squares represent data processing steps with functionalities labeled for each step. The purple ovals represent the data types as labeled in the diagram. Intermediate results are abbreviated as "IR". The green arrow represents the direction of the data flow between different steps, and the purple oval represents input and output data types at different processing steps. Note that the MOCA process and DeconvSK processing step involve non-peer-reviewed work, which is beyond the scope of this manuscript and won't be discussed.

The following modules are implemented to facilitate the *PySOFI* analysis pipeline. The `reconstruction.py` module provides capabilities for SOFI moments and cumulants calculations [1], as well as bleaching correction for a TIFF movie. The `finterp.py` module provides Fourier interpolation on a TIFF stack, which is a necessary step for fSOFI alike analysis [9]. `filtering.py` and `ldrc.py` constitute a collection of modules relevant to SOFI 2.0 [11] analysis. Specifically, the `filtering.py` module is for pixel-wise noise filtering along the time axis, and the `ldrc.py` is for local dynamic range compression (*ldrc*) of images with a large dynamic range of pixel values [10]. The `masks.py` module is used to generate Gaussian kernels, and the `visualization.py` module provides visualization options using an interactive visualization package `bokeh`. The data class module (`PysofiData`) is encapsulated in the `pysofi.py` file. The input parameters from the users, the raw data, and the intermediate results are bundled in the `PysofiData` object as attributes, and the processing steps as methods. The processing steps are implemented as function modules, and imported and used in the data class module. In summary, the specific functions are implemented in the function modules, while `PysofiData` serves the purpose of organizing the data processing workflow.

In general, we adopted a simple architecture for *PySOFI* with a collection of independent function modules and only one class module (the data class). The functions are imported and used inside the data class across different methods as needed, therefore the implementation is flexible with minimum repetition of codes. The function modules can be implemented, modified, and tested independently, ensuring flexibility and convenience for maintenance. Extending the package can be done by implementing additional function modules. It can be used as a standalone process, or be integrated into the data processing workflow through the `PysofiData` class. The investigators also have the flexibility to disseminate the *PySOFI* package and construct their own data processing workflow (similar to the `PysofiData` class).

3. IMPLEMENTATION OF SOFI ANALYSIS USING *PYSOFI*

We provide a collection of Jupyter Notebooks (outlined in Figure 3) as examples for *PySOFI* implementations and applications. The prefix (E#) of each filename is used as a reference to each notebook in the following text for simplicity. We present example *PySOFI* analysis steps (E1, E2, E4 to E6), visualization of the result with combined color-map and transparency-map (E8), and

***PySOFI* modules**

Data class	pysofi.py	Defines the main data class called "PysofiData"
Function modules	reconstruction.py	Contains tools for cumulant and moment reconstruction.
	finterp.py	Contains tools for Fourier interpolation on *.tiff stacks for fSOFI processing.
	filtering.py	Contains tools for noise filtration along the time axis.
	deconvsk.py	Contains tools for shrinking kernal deconvolution (DeconvSK).
	ldrc.py	Contains tools for local dyanmic range compression of images.
	visualization.py	Contains tools for visualization of the results.
	moca.py	Contains tools for Multi Order Cumulant Analysis (MOCA).
	masks.py	Contains tools for generating Gaussian kernels.

Figure 2: ***PySOFI* modules.** *PySOFI* contains one data class and eight function modules. Detailed descriptions are available in the online documentation for *PySOFI*. Note that the moca.py and deconvsk.py modules involves non-peer-reviewed work and is beyond the scope of this work, while this work focuses on the introduction of the software package *PySOFI*, therefore, moca.py and deconvsk.py won't be discussed in this manuscript.

the effect of data acquisition length on SOFI reconstruction performance (E9, E10). We also demonstrate SOFI 2.0 analysis (E11) and characterization of cusp-artifacts (E12 and E13). Two processing steps (E3 and E7) address non-peer-reviewed methods that is beyond the scope of this work, therefore wont be discussed in this manuscript. The analysis processes are integrated through the `PysofiData` class for all the notebooks except for the demonstration of noise filtration (E2).

In the text below, we provide brief descriptions of E1, E2, E4, E5, and E6. The complete detailed description and examples are provided in the Jupyter Notebooks in the online Github repository.

3.1 Moment and cumulant reconstructions (E1)

Traditionally, SOFI achieves resolution enhancement by computing different orders of cumulants of optical signal fluctuations in time. The theoretical resolution enhancement for SOFI is $1/\sqrt{n}$ fold for the n^{th} order SOFI cumulant. Once combined with deconvolution, the theoretical resolution enhancement can increase to $1/n$.

To obtain the n^{th} order SOFI cumulant, one way is to construct the n^{th} order cumulant as a polynomial consists of moments from the first order to the n^{th} order, as shown in the previous work [1]. Another way, which is used by *PySOFI*, is to construct the following recursive relation: $Cum_n = G_n - \sum_{i=1}^{n-1} C_{n-1}^{i} \cdot Cum_{n-i} \cdot G_i$, where Cum_n represents the n^{th} order cumulant, G_n represents the n^{th} order moment, and C_N^M means the number of combinations of "N choose M". Regarding the moment calculations, *PySOFI* support calculations of moments directly from the time series of each pixel. The moments can be also calculated as a reconstruction from a series of cumulants as used in our previous study [11].

The calculation of cumulants and moments are the fundamental processing elements in the SOFI analysis. The `PysofiData` class organizes the analysis workflow and can be used to calculate both moments and cumulants. Essentially, the relevant function modules are imported and integrated in the `PysofiData` to support such analysis. For example, the following scripts would calculate the 4^{th} order moment and cumulant of the specified TIFF stack named `Block1.tif` through the `PysofiData` class:

```
# load data into PysofiData object
filepath = '../sampledata'
filename = 'Block1.tif'
d = pysofi.PysofiData(filepath, filename)
# calculate the 4th order moment image
m_im = d.moment_image(order=4)
# calculate the 4th order cumulant image
k_set = d.cumulants_images(highest_order=4)
```

We can also directly import the function module, `reconstruction.py`, to perform the relevant calculations. This option is designed to support dissemination of the *PySOFI* package to facilitate independent analysis, which is often useful when developing new methods built upon SOFI analysis. The following scripts demonstrate how to perform such analysis with moment and cumulant calculations up to 4^{th} order:

```
# import the relevant function modules, and define the path and name for the data.
from pysofi import reconstruction as rec
filepath = '../sampledata'
```

Examples for *PySOFI* analysis (Jupyter Notebooks)

Group	Notebook Name	Data	Section #
Processing steps demonstration	E1_MomentCumulantReconstructions	Block1.tif (live cell imaging)	3.1
	E2_NoiseFiltration	Block1.tif - Block20.tif (live cell imaging)	3.2
	E3_ShrinkingKernelDeconvolution	Block10.tif (live cell imaging)	N.A.
	E4_LDRCMethod	Block1.tif (live cell imaging)	3.3
	E5_FourierInterpolation	Block10.tif (live cell imaging)	3.4
	E6_BleachingCorrection	Bleach_SlowVaryingRho_frame2000_Emi51.tif (simulation)	3.5
	E7_MOCA	3Emitters_frame5000_Emi3_close.tif (simulation) SlowVaryingRho_frame2000_Emi51.tif (simulation) RndomCurves_frame15000_rho04.tif (simulation)	N.A.
Analysis explorations	E8_ResultVisualization	RndomCurves_frame15000_rho04.tif (simulation)	3.6
	E9_ReconstructionConvergence	frame1000_10000.npy frame11000_15000.npy frame16000_20000.npy (generated in E8)	N.A.
	E10_ReconstructionConvergence_SampleAnalysis	nobleach_frame20000_3.tif (simulation)	N.A.
SOFI 2.0 demonstration	E11_PysofiExample_LiveCellActinFilaments	Block1.tif - Block20.tif (live cell imaging)	N.A.
Cusp-artifacts demonstrations	E12_CuspArtifactsDemo1_3Emitters	3Emitters_frame5000_Emi3_close.tif (simulation)	N.A.
	E13_CuspArtifactsDemo2_SlowVaryingRho	SlowVaryingRho_frame2000_Emi51.tif (simulation)	N.A.

Figure 3: **Jupyter notebook examples for *PySOFI*.** We provide 13 *PySOFI* demonstrations as Jupyter Notebooks which can be categorized into to 4 Groups (first column). The filenames (second column) indicates the focus of each Jupyter Notebook. The relevant data sets (third column) are shared on figshare. Brief descriptions of most processing steps (E1, E2, E4, E5, E6) and their notebooks are provided in the relevant section (fourth column). The theory behind E9 to E13 are not included in this manuscript but the relevant concepts are discussed in [26] and [11]. The notebooks are the *PySOFI* implementations of the relevant methods to support the utilization of them. In particular, in E11, we show the general guidelines for performing SOFI 2.0 analysis on live-cell fluorescence imaging results using *PySOFI*. Note that MOCA and DeconvSK processing step involve non-peer-reviewed work which is beyond the scope of this w, therefore notebooks E3 and E7 won't be discussed in this manuscirpt.

```
filename = 'Block1.tif'
# calculate the 4th order moment image
m_set = rec.calc_moments(filepath, filename, highest_order=4)
# calculate the 4th order cumulant image
k_set = rec.calc_cumulants_from_moments(m_set)
```

More detailed demonstrations are available in the corresponding Jupyter Notebook (E1).

3.2 Temporal Noise Filtering (E2)

Temporal noise filtering is fundamental in the image processing for fluorescence microscopy, especially in scenarios where continuous and prolonged live cell imaging is desired where the excitation power is maintained at a low level to minimize photo toxicity and photo-bleaching. The lower excitation power often results in reduced signal to noise ratio. Traditional noise filtering is performed with a spatial filter where each image for every given time instance is spatially filtered independently. However, because noise filtering in the spatial spectrum domain is equivalent to a convolution operation of the image with the kernel corresponds to the inverse Fourier transform of the low-pass filter, it is conceivable that the spatial noise filtering would reduce the spatial resolution. On the other hand, to achieve a super-resolution movie, we are focusing on the sample conditions where the semi-static assumption is valid, which requires slow dynamics in the sample and the temporal noise filtering has been proven useful [11]. This is because slow dynamics ensures that the signal of interest exists in the low frequency domain while the noise is populated in the high frequency domain in the time axis, therefore the temporal spectrum filtering can be effective. Additionally, because this filtering is performed along the time axis, the spatial resolution is not directly influenced.

We have implemented such temporal noise filtering in *PySOFI* as a function module `filtering.py`. It is useful when analyzing multiple TIFF stacks corresponding to consecutive time-blocks. In such scenario, the feature is assumed to be

semi-static within each individual time block, and the corresponding TIFF stack is analyzed independently. We can perform the temporal noise filtering on the results across all the time blocks to further enhance the image quality.

For example, we can perform the temporal noise filtering on the 6^{th} order moment images calculated from 20 blocks of TIFF stacks (each contains 200 frames) using the following scripts:

```
# First, we define the list of TIFF stacks that corresponds to 20 different time blocks of a movie:
filenum = 20
filepath = '../sampledata'
filenames = ['Block' + str(i+1) + '.tif' for i in range(filenum)]
# Second, we perform the sixth order moment calculations for all the blocks
dset = {}; m_set = {}
for filename in filenames:
    dset[filename] = pysofi.PysofiData(filepath, filename)
    m_set[filename] = dset[filename].moment_image(order=6, finterp=False)
# Third, we generate a noise filter as a 1-Dimensional Guasisan profile:
nf = masks.gauss1d_mask(shape = (1,21), sigma = 2)
# Last, we perfrom the time-axis noise filtering
m_filtered_set = filtering.noise_filter1d(dset, m_set, nf, return_option=True, return_type='dict')
```

The results from the temporal noise filtering are stored as a dictionary in the `m_filtered_set`, where keys for elements are file names for each block of TIFF images, and values are the corresponding filtered images. The filtered images are also updated to each `PysofiData` objects as a `PysofiData.filtered` attribute. More detailed demonstrations are available in the corresponding Jupyter Notebook (E2).

3.3 Local dynamic range compression (*ldrc*) (E4)

One of the key challenges for high order SOFI cumulant calculations is the high dynamic range (HDR) of pixel intensities [1]. The HDR issue also exists in the high order moment images [11]. The 'local dynamic range compression (*ldrc*)'[11] method was developed to mitigate such issue (and is implemented in PySOFI) by rescaling the pixel intensities of a given image based on a reference image. First, a reference image with the same feature but a more confined dynamic range is defined (e.g., the average image, the second-order moment or cumulant SOFI image). The compression is performed locally in a small window that scans across the image with a stride of 1 pixel. In each window, the pixel intensities of the original image are linearly re-scaled to share the same dynamic range as the reference window [11]. The final value of each pixel is the average of the corresponding re-scaled values of them across all windows covering it.

In *PySOFI*, *ldrc* is implemented in the function module ldrc.py and integrated in the `PysofiData.ldrc()` method. The following scripts will calculate the 6^{th} order moment (*m6*) and the average image (*mean*), and perform *ldrc* on *m6* using *mean* as the reference:

```
# first, import the two relevant function modules, reconstruction and ldrc
from pysofi import reconstruction as r
from pysofi import ldrc
# define the path and file name of the data file.
filepath = '../sampledata'
filename = 'Block1.tif'
# calculate the 6-th order moment (m6) and the average image (mean) using the reconstruction module
m6 = r.calc_moment_im(filepath, filename, order=6, frames=[0, 50])
mean = r.average_image(filepath, filename)
# compress the dynamic range of m6 with reference to mean using ldrc
ldrc_im = ldrc.ldrc(mask_im=mean, input_im=m6, order=6, window_size=[20, 20])
```

We can also perform the *ldrc* processing directly through the `PysofiData.ldrc()` method using the following script:

```
# load data into PysofiData object
filepath = '../sampledata'
filename = 'Block1.tif'
# load teh data into a PysofiData class object
d = pysofi.PysofiData(filepath, filename)
# calculate moments
d.moment_image(order=6, finterp=False)
# perfrom ldrc
d.ldrc(mask_im=d.ave, input_im=d.moments_set[6], order=6, window_size=[20, 20])
```

Note that the direct *ldrc* processing on *m6* often yields noisy results (We have demonstrated the results in the relevant Jupyter Notebook (E4)). However, *ldrc* plays an important role in the SOFI 2.0 pipeline where the noise filtering and deconvolution are performed. In Figure 4, we compare the the partially processed SOFI 2.0 image (excluded *ldrc*) and the full SOFI 2.0 processed

image (included *ldrc*). We can see that the feature in the image are preserved without *ldrc*, but imperceptible due to the HDR issue. On the other hand, *ldrc* mitigates the HDR issue and provide an image where the dim features are shown more clearly. More detailed demonstrations are available in the corresponding Jupyter Notebook (E4).

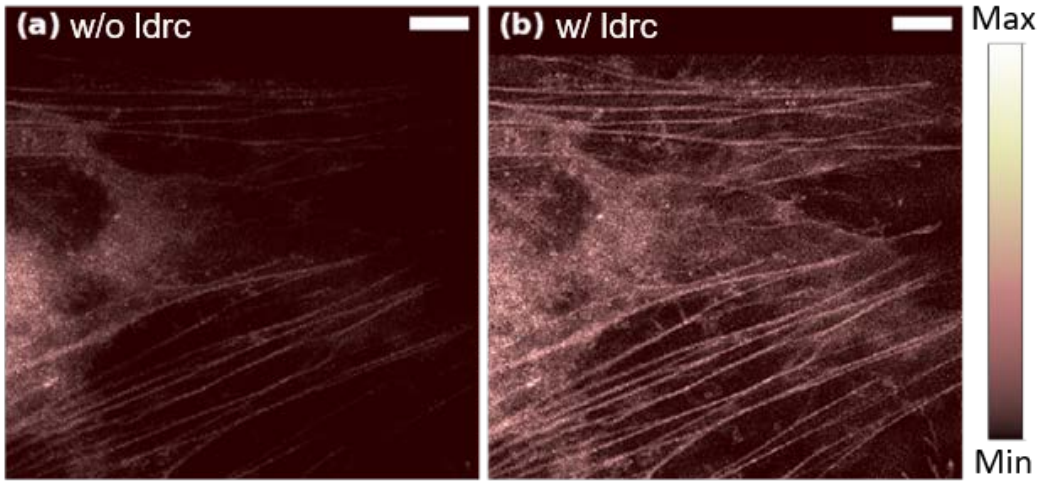

Figure 4: **ldrc demonstration.** Experimental demonstration of *ldrc* algorithm on HeLa cells transfected with Dronpa-C12 fused to β-Actin. Both images are processed using 6^{th} order moment, noise filtering and deconvolution, and obtained during the SOFI 2.0 analysis pipeline, before (a) and after (b) the *ldrc* step. Scale bars: $8\mu m$.

3.4 Fourier interpolation (E5)

Fourier interpolation stochastic optical fluctuation imaging (fSOFI) solves the finite pixelation problem of SOFI by adding virtual pixels using Fourier transforms [9]. We have implemented the Fourier interpolation method in *PySOFI* to integrate the fSOFI analysis as an optional processing step. In our implementation, for the forward Fourier Transform, the Fourier transformation matrix was created with a size the same as the input image. We created the inverse Fourier transformation matrix to include the extra interpolation position coordinates, and omitted the "zero-padding" step in the Fourier space to avoid burdening the computation.With the Fourier interpolation, the input image/video is 'projected' onto a more refined grid with finer pixel size.

In *PySOFI*, Fourier interpolation is implemented in the function module finterp.py and integrated in the `PysofiData.finterp_tiffstack()` method. We can perform the Fourier interpolation and save the output as as a series of .tiff stacks. For example, the following scripts will calculate the 2- and 4- fold Fourier interpolation of the initial 100 frames from the example data set `block10.tiff`, and save the interpolated images into two .tiff stacks: `block10_InterpNum2.tiff` and `block10_InterpNum4.tiff` respectively.

```
# import the relevant tools
from pysofi import pysofi
# load data into PysofiData object
filepath = '../sampledata'
filename = 'Block10.tif'
d = pysofi.PysofiData(filepath, filename)
# calculte the Fourier interpolation
d.finterp_tiffstack(interp_num_lst=[2,4], frames=[0,100], save_option=True, return_option=False)
```

We can also perform the Fourier interpolation by using the `finterp.py` module as shown below:

```
# import the relevant tools
import tifffile as tiff
from functions import finterp
# load a single image from the relevant data file
filepath = '../sampledata'
filename = 'Block10.tif'
im = tiff.imread(filepath + '/' + filename, key=15) # read a frame
# perform Fourier interpolation
```

```
finterp_im2 = finterp.fourier_interp_array(im, [10]) # perform a 10-fold interpolation in the image.
```

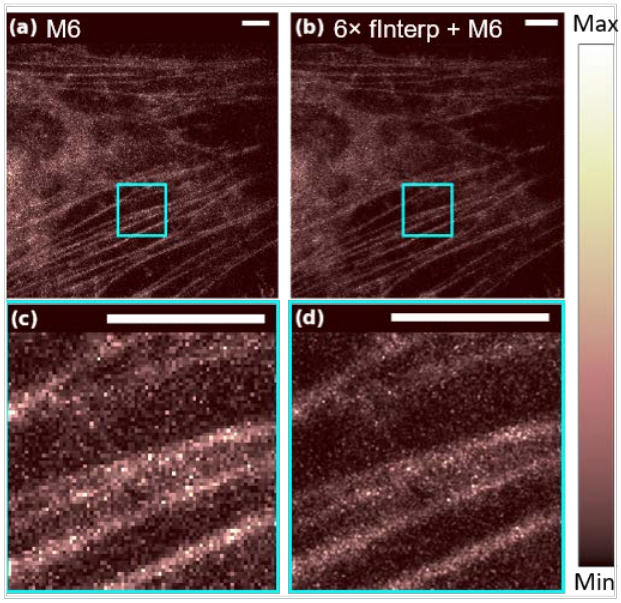

Figure 5: **Fourier interpolation demonstration.** Experimental demonstration of Fourier interpolation algorithm on HeLa cells transfected with Dronpa-C12 fused to β-Actin. (a) The 6^{th} order moment-reconstructed image of the original wide-field acquisition. (b) The 6^{th} order moment image after the Fourier interpolation. ldrc is performed on both (a) and (b) to compress the dynamic range of the reconstruction. (c) A zoom-in box of (a). (d) A zoom-in box of (d). Scale bars: $8\mu m$.

Figure 5 demonstrates the performance of the Fourier interpolation. Based on the Nyquist-Shannon sampling theorem [31, 32], we recommend setting the interpolation factor at least two times the highest order for moment-/cumulant reconstructions. For instance, if we plan to start the SOFI 2.0 pipeline with the 6^{th} order moment image, we should pass `interp_num_lst = [12]` to `d.finterp_tiffstack`. However, in practice, depending on the dimension and length of the input file, Fourier interpolation might consume large processing memory and time. If computation resources are limited, we recommend saving the interpolated image stack as tiff files firs tinstead of returning them, and then process the new file. Besides `d.finterp_tiffstack`, another option to include Fourier interpolation in the SOFI processing pipeline is to pass (`finterp = True`) and a interpolation factor (`interp_num=6`) when calculating the moment/cumulant reconstructions (see section 3.4).

More detailed demonstrations are available in the corresponding Jupyter Notebook (E5).

3.5 Bleaching correction (E6)

Photobleaching of fluorescent probes is a general concern for super-resolution imaging analysis methods. As for SOFI, photobleaching can cause errors in virtual brightness displayed in moment or cumulant images [26]. Photobleaching leads to the loss of the fluorescence signal, which is mathematically equivalent as if the fluorophore is switched to a prolonged "off" state, degrading the quality of SOFI results. Therefore, a bleaching correction is critical.

PySOFI employs a bleaching correction technique [11] that divides the whole video into shorter blocks based on the total signal intensity, $I(t)$, where t is the time index, and $I(t)$ is the summation of all the pixel values of the image at time index t. The individual blocks are processed independently and combined subsequently to form a SOFI movie. First, the time series of the total signal intensity is smoothened to obtain a monotonically decreasing curve as an estimation of the bleaching profile of the movie. Then, based on the signal evolution over time, the sizes of the shorter blocks are determined so that the fractional signal decrease within each block (characterized by the bleaching correction factor, f_{bc}) is identical [11]. The final SOFI moment/cumulant images with bleaching correction are the average of those calculated from individual blocks. Fig. 6 shows that with the help of bleaching correction, the virtual brightness distribution and the photophysical properties (7c, 7f) are successfully restored, yielding similar values as compared to the simulated case without bleaching (7b).

PySOFI offers two ways for bleaching correction. One way is through the `PysofiData` class as shown below:

```
# import the relevant tools
```

```
from pysofi import pysofi
# load data into PysofiData object
filepath = '../sampledata'
filename_bleach = 'Bleach_SlowVaryingRho_frame2000_Emi51.tif'
# load the dataset with bleaching into a PysofiData class object d_bleach
d_bleach = pysofi.PysofiData(filepath, filename_bleach)
# calculate the sofi cumulants with bleach correction
k_set_bleach_corrected = d_bleach.cumulants_images(highest_order=7, bleach_correction=True,
                                                   smooth_kernel=251, fbc=0.04)
```

We can also directly import the relevant function module `reconstruction.py` and perfrom bleaching correction as shown below:

```
# improt the function module and define the path and file name of the dataset
from pysofi import reconstruction as r
filepath = '../sampledata/simulations'
filename_bleach = 'Bleach_SlowVaryingRho_frame2000_Emi51.tif'
# perform bleaching correction on the designated dataset.
r.correct_bleaching(filepath, filename_bleach, fbc=0.04, smooth_kernel=251,
                        save_option=True, return_option=False)
```

In this example, we applied bleaching correction to a TIFF stack, and the bleaching corrected movie is saved as a separate TIFF stack with the string "_bc" appended to the original file name.

More detailed demonstrations are available in the corresponding Jupyter Notebook (E6).

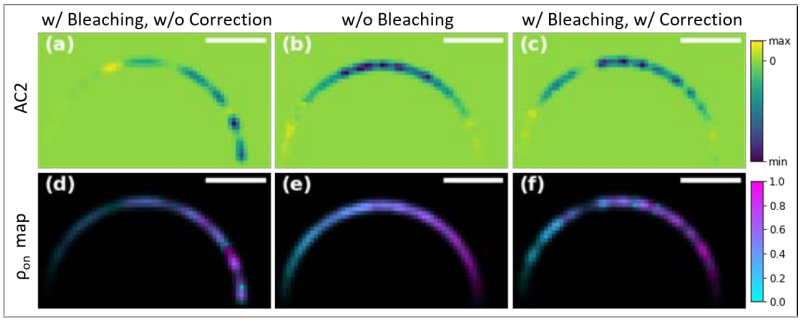

Figure 6: **Bleaching correction demonstration on a simulation data.** The fourth-order cumulant image (a-c) and multi-order cumulant analysis (MOCA) (d-f) is performed on a simulated video. A semicircle is populated with emitters with on-time ratios ranging from 0.01 (left) to 0.99 (right) with around 0.02 intervals. For emitters with photobleaching but without a bleaching correction step, the reconstructed pixel intensities (a) and emitters on-time ratio estimation (d) are far off from the true values (b, e), while the bleaching correction restores the information (c, f). Scale bars: 1.4μm.

3.6 Result visualization (E8)

We provide some simple visualization options in *PySOFI* to display either single or multiple images, with the options to adjust image contrast, and display the image with a transparency map defined as an input parameter. `Bokeh` is used to offer interactive display. More detailed demonstrations are available in the corresponding Jupyter Notebook (E8).

4. DISCUSSION

In this work, we developed *PySOFI*, an open source python package for SOFI analyses. *PySOFI* contains the essential functionalities for conventional SOFI analysis as well as several derivative methods [9, 11, 26, 10].

PySOFI adopts a simple architecture, where all the data processing steps are implemented as independent function modules, and only one class module (the data class `PysofiData`) is used to manage the data processing workflow. The functions can be tested independently and used in different processing pipelines. A fast prototype on new analysis can be achieved by disseminating and reorganizing the processing step. One can implement additional processing steps as independent python functions with the help of existing *PySOFI* functions. New functions can be used as standalone modules, or can be integrated into the `PysofiData` class to support the new analysis pipeline. New classes can be constructed for different analysis pipelines as well.

We adopted Sphinx to manage the *PySOFI* documentation, which is available as an online documentation to facilitate community usage. Additionally, each processing element of the analysing pipeline are demonstrated in individual Jupyter Notebooks. In each notebook, we also provide instructions on how to tune processing variables and explore input data.

PySOFI is housed on GitHub as an open source repository, any interested individuals can learn, inspect, validate and contribute to the package. The user interactions on GitHub (e.g., fork, create pull requests, and report issues) engage the community communications. We expect *PySOFI* to benefit general SOFI users for existing SOFI analysis, as well as developers and new investigators interested in developing new SOFI-relevant analysis method.

ACKNOWLEDGMENTS

The work from Y. M. and S. W. was supported by the National Science Foundation under Grant No. DMR-1548924 and by the Dean Willard Chair funds. The work from X. Y. was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Release number: LLNL-JRNL-827376.

AUTHOR CONTRIBUTIONS

Y.M. designed the *pysofi* architecture, implemented the *pysofi* package, generated simulation videos for figure 7. X.Y designed the research, supervised the development of *PySOFI* package, developed the MOCA algorithms and collected experimental and simulation data. All authors analyzed the data, discussed the results, and wrote the manuscript.

DATA AVAILABILITY

The data for this project is partially available on the project repository, and all organized on figshare. The usage of the example datasets are described belowL Block1.tif to Block20.tif are live cell imaging data [11] using Hela cells labeled with Dronpa-C12 fused with β-actin. All the rest of the data sets are simulation data sets used and described in the example Jupyter Notebooks.

REFERENCES

[1] Thomas Dertinger et al. “Fast, background-free, 3D super-resolution optical fluctuation imaging (SOFI)”. In: *Proceedings of the National Academy of Sciences* 106.52 (2009), pp. 22287–22292.

[2] Kristin Grußmayer et al. “Self-Blinking Dyes Unlock High-Order and Multiplane Super-Resolution Optical Fluctuation Imaging”. In: *ACS nano* 14.7 (2020), pp. 9156–9165.

[3] S Duwé, W Vandenberg, and P Dedecker. “Live-cell monochromatic dual-label sub-diffraction microscopy by mt-pcSOFI”. In: *Chemical Communications* 53.53 (2017), pp. 7242–7245.

[4] Benjamien Moeyaert and Peter Dedecker. “PcSOFI as a smart label-based superresolution microscopy technique”. In: *Photoswitching Proteins*. Springer, 2014, pp. 261–276.

[5] Sam Duwé, Benjamien Moeyaert, and Peter Dedecker. “Diffraction-Unlimited Fluorescence Microscopy of Living Biological Samples Using pcSOFI”. In: *Current protocols in chemical biology* 7.1 (2015), pp. 27–41.

[6] Peter Dedecker, Gary CH Mo, and Jin Zhang. “Widely Accessible Method for Superresolution Fluorescence Imaging of Living Systems”. In: *Biophysical Journal* 104.2 (2013), 535a.

[7] Stefan Geissbuehler et al. “Mapping molecular statistics with balanced super-resolution optical fluctuation imaging (bSOFI)”. In: *Optical Nanoscopy* 1.1 (2012), pp. 1–7.

[8] Thomas Dertinger et al. “Achieving increased resolution and more pixels with Superresolution Optical Fluctuation Imaging (SOFI)”. In: *Optics express* 18.18 (2010), pp. 18875–18885.

[9] Simon C Stein et al. “Fourier interpolation stochastic optical fluctuation imaging”. In: *Optics express* 23.12 (2015), pp. 16154–16163.

[10] Xiyu Yi. “Super resolution of Optical Fluctuation Imaging 2.0 (SOFI-2.0): Towards fast super resolved imaging of live cells”. PhD thesis. UCLA, 2017.

[11] Xiyu Yi et al. “Moments reconstruction and local dynamic range compression of high order Superresolution Optical Fluctuation Imaging”. In: *Biomedical optics express* 10.5 (2019), pp. 2430–2445.

[12] S Vlasenko et al. “Optimal correlation order in superresolution optical fluctuation microscopy”. In: *Physical Review A* 102.6 (2020), p. 063507.

[13] Dario Cevoli et al. “Design of experiments for the optimization of SOFI super-resolution microscopy imaging”. In: *Biomedical Optics Express* 12.5 (2021), pp. 2617–2630.

[14] Baoju Wang et al. “Active-modulated, random-illumination, super-resolution optical fluctuation imaging”. In: *Nanoscale* 12.32 (2020), pp. 16864–16874.

[15] Adrien Descloux et al. “Combined multi-plane phase retrieval and super-resolution optical fluctuation imaging for 4D cell microscopy”. In: *Nature Photonics* 12.3 (2018), pp. 165–172.

[16] Eliel Hojman et al. “Photoacoustic imaging beyond the acoustic diffraction-limit with dynamic speckle illumination and sparse joint support recovery”. In: *Optics express* 25.5 (2017), pp. 4875–4886.

[17] MinKwan Kim et al. “Superresolution imaging with optical fluctuation using speckle patterns illumination”. In: *Scientific reports* 5.1 (2015), pp. 1–10.

[18] Ori Katz and Noam Shekel. “Using fiber-bending generated speckles for improved working distance and”. In: *Optics Letters* 4.C1 (2020), p. C2.

[19] Lydia Kisley et al. “Characterization of porous materials by fluorescence correlation spectroscopy super-resolution optical fluctuation imaging”. In: *ACS nano* 9.9 (2015), pp. 9158–9166.

[20] Stefan Geissbuehler et al. “Live-cell multiplane three-dimensional super-resolution optical fluctuation imaging”. In: *Nature communications* 5.1 (2014), pp. 1–7.

[21] Ashley M Rozario et al. “‘Live and Large’: Super-Resolution Optical Fluctuation Imaging (SOFI) and Expansion Microscopy (ExM) of Microtubule Remodelling by Rabies Virus P Protein”. In: *Australian Journal of Chemistry* 73.8 (2020), pp. 686–692.

[22] Adrien C Descloux et al. “Experimental Combination of Super-Resolution Optical Fluctuation Imaging with Structured Illumination Microscopy for Large Fields-of-View”. In: *Acs Photonics* 8.8 (2021), pp. 2440–2449.

[23] Aleksandra Sroda et al. “SOFISM: Super-resolution optical fluctuation image scanning microscopy”. In: *Optica* 7.10 (2020), pp. 1308–1316.

[24] Thomas Dertinger et al. “Advances in superresolution optical fluctuation imaging (SOFI)”. In: *Quarterly reviews of biophysics* 46.2 (2013), pp. 210–221.

[25] Sangyeon Cho et al. “Simple super-resolution live-cell imaging based on diffusion-assisted Förster resonance energy transfer”. In: *Scientific reports* 3.1 (2013), pp. 1–7.

[26] Xiyu Yi and Shimon Weiss. “Cusp-artifacts in high order superresolution optical fluctuation imaging”. In: *Biomedical optics express* 11.2 (2020), pp. 554–570.

[27] Maurice George Kendall et al. “The advanced theory of statistics.” In: *The advanced theory of statistics.* 2nd Ed (1946).

[28] Peter Dedecker et al. “Localizer: fast, accurate, open-source, and modular software package for superresolution microscopy”. In: *Journal of biomedical optics* 17.12 (2012), p. 126008.

[29] Nils Gustafsson. “QuickSOFI: High-Performance Superresolution Optical Fluctuation Imaging (SOFI) Plugin for ImageJ Using GPU Computing”. In: *University College London, UK* (2014), pp. 1–18.

[30] Marcel Leutenegger et al. “SOFI Package”. In: *GitHub: https://github.com/kgrussmayer/sofipackage* (2021).

[31] Harry Nyquist. “Certain topics in telegraph transmission theory”. In: *Transactions of the American Institute of Electrical Engineers* 47.2 (1928), pp. 617–644.

[32] Claude E Shannon. “A mathematical theory of communication”. In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.