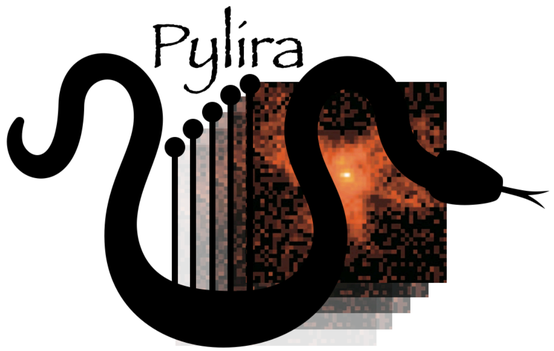
Pylira: deconvolution of images in the presence of Poisson noise

July 13th, 2022

Scipy 2022, Austin TX



Axel Donath

for the CHASC Astro-Statistics Collaboration

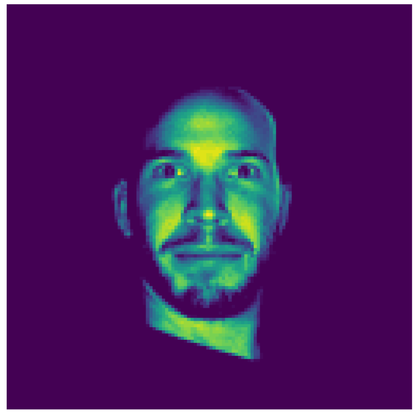
About me

- **Post-Doc @ Center for Astrophysics** | Harvard Smithsonian
- **Gamma-ray & X-ray astronomer**, open source scientific software developer, self-taught “statistician”
- Member of the **CHASC Astro-Statistics Collaboration**
  - https://hea-www.harvard.edu/astrostat/
  - https://github.com/astrostat/
- Also one of the **lead developers of Gammapy** (https://gammapy.org) a Python package for analysis of astronomical gamma-ray data and “Science Tool” library for the Cherenkov Telescope Array (https://www.cta-observatory.org)
- My GitHub profile: https://github.com/adonath

Introduction

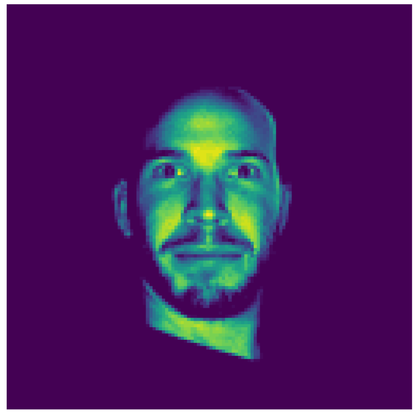
The “low counts” imaging process

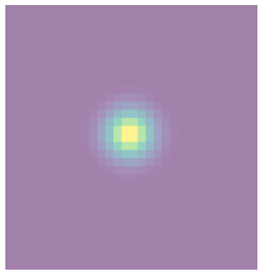
Ground Truth (GT)



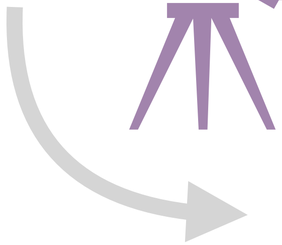
The “low counts” imaging process

Ground Truth (GT)



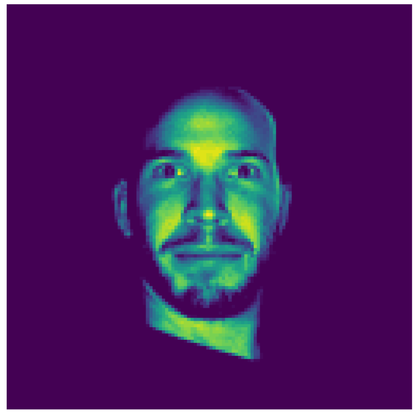


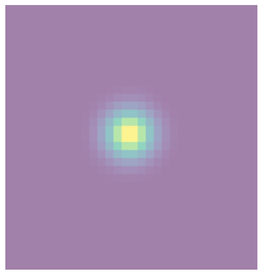
Point Spread Function
(PSF)



The “low counts” imaging process

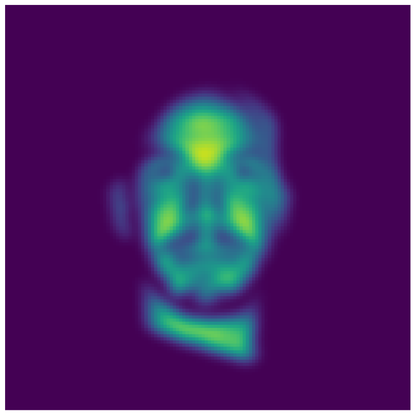
Ground Truth (GT)



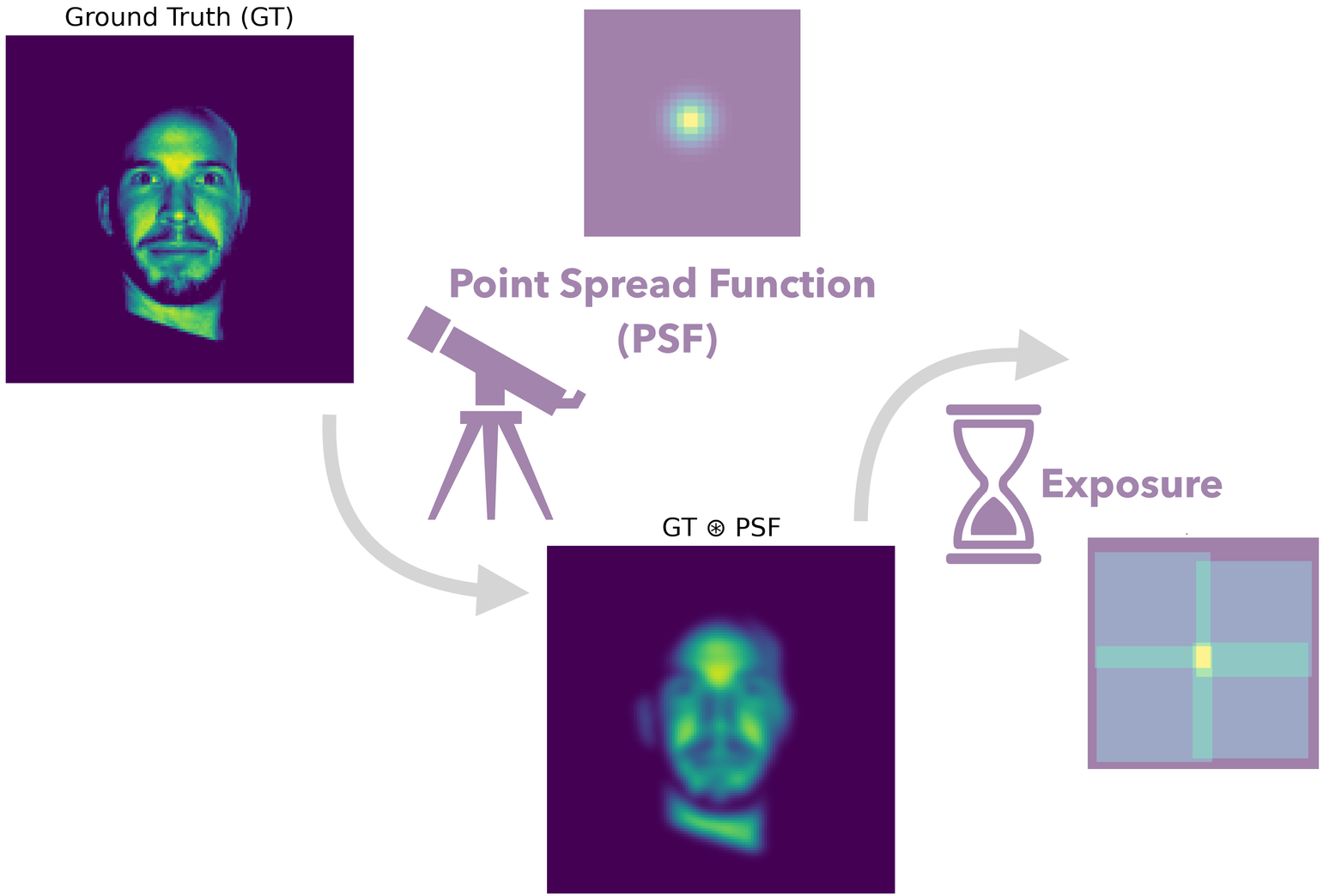


Point Spread Function (PSF)

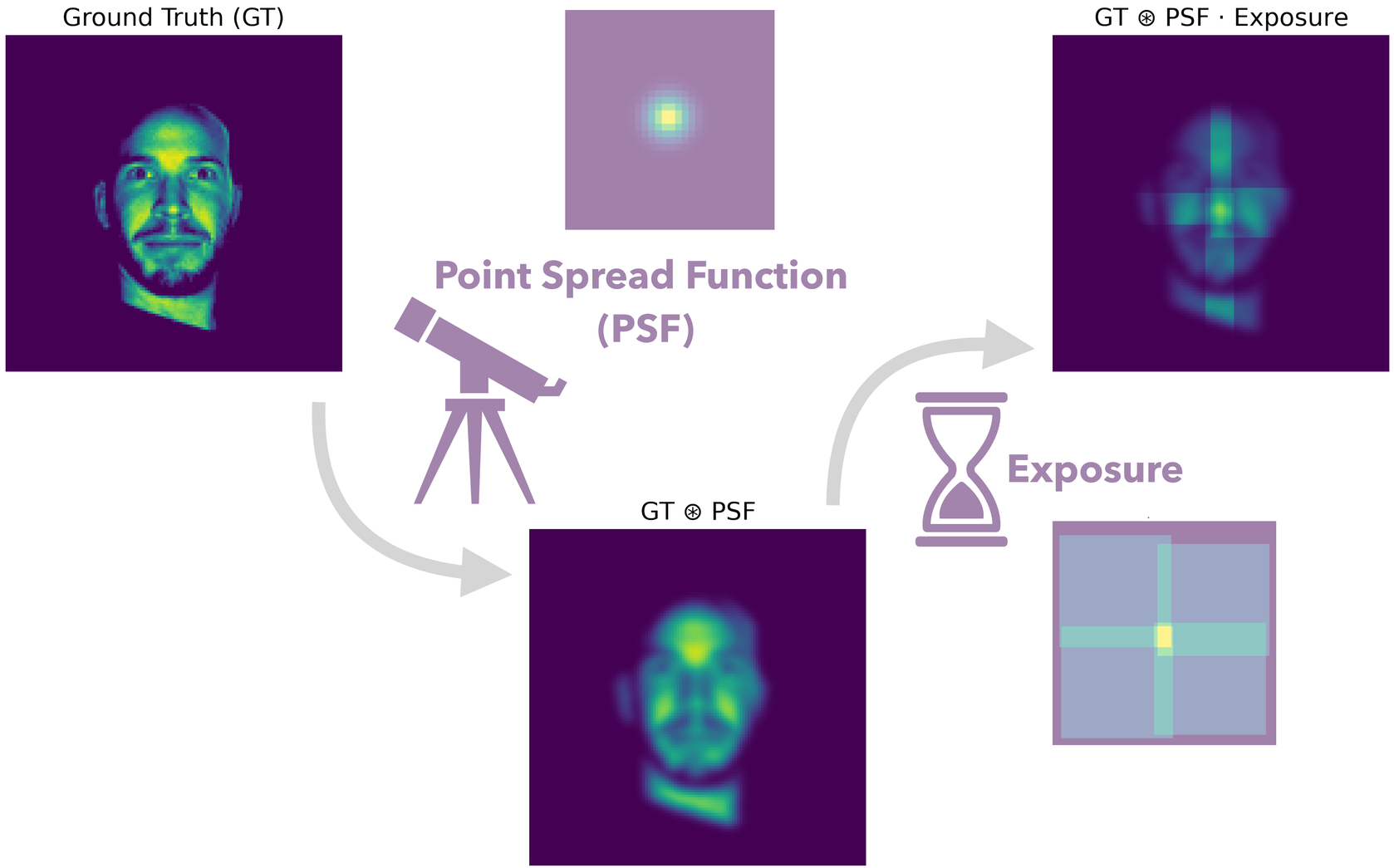
GT ⊛ PSF



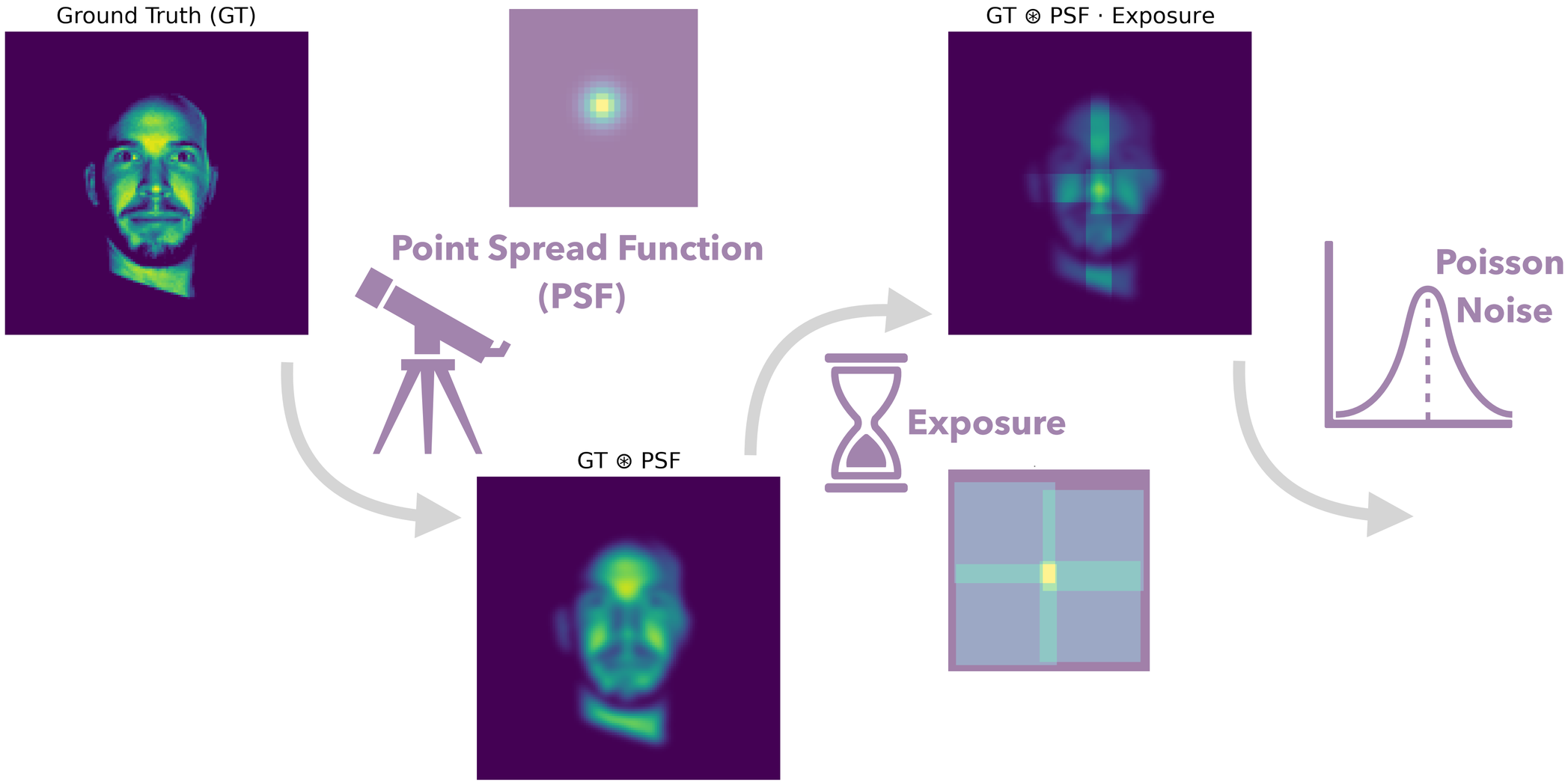
The “low counts” imaging process



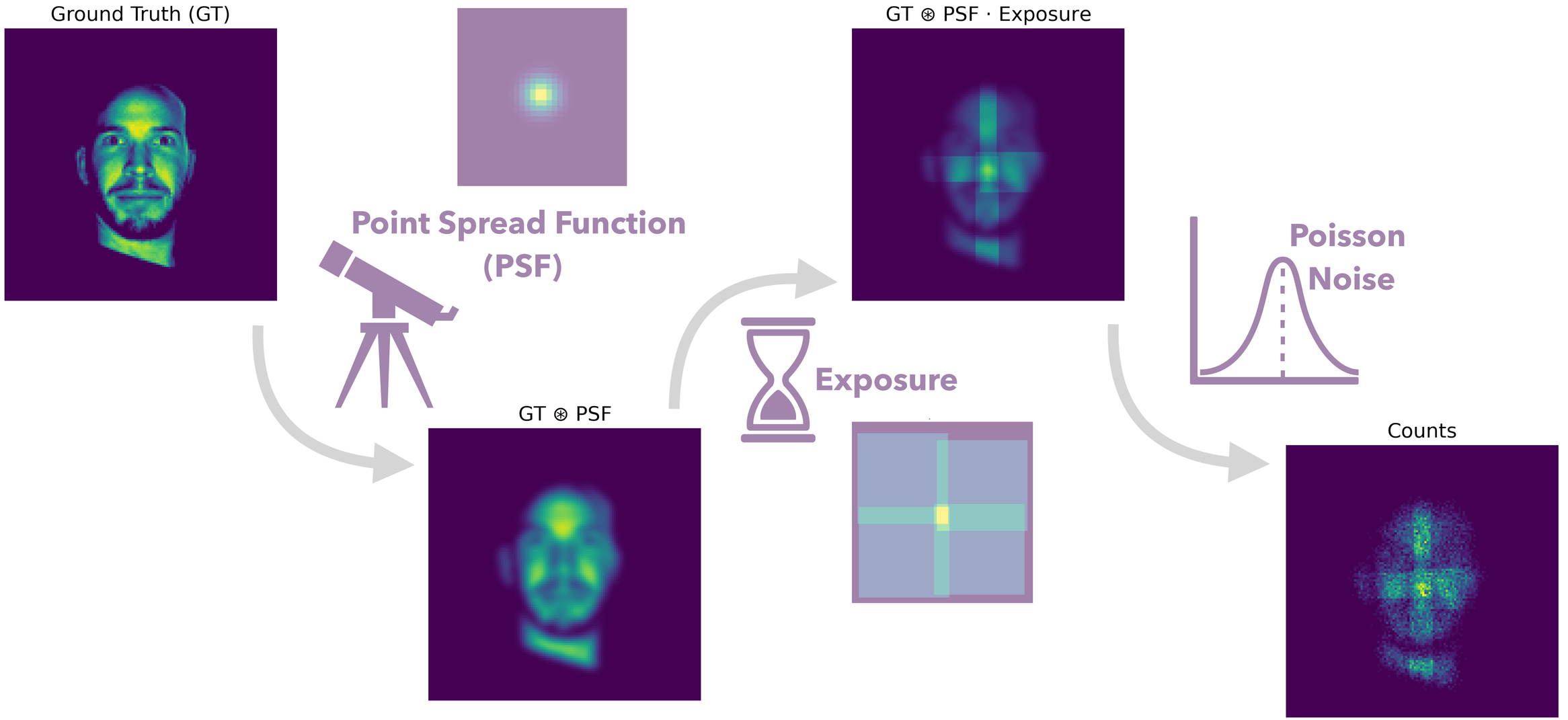
The “low counts” imaging process



The “low counts” imaging process

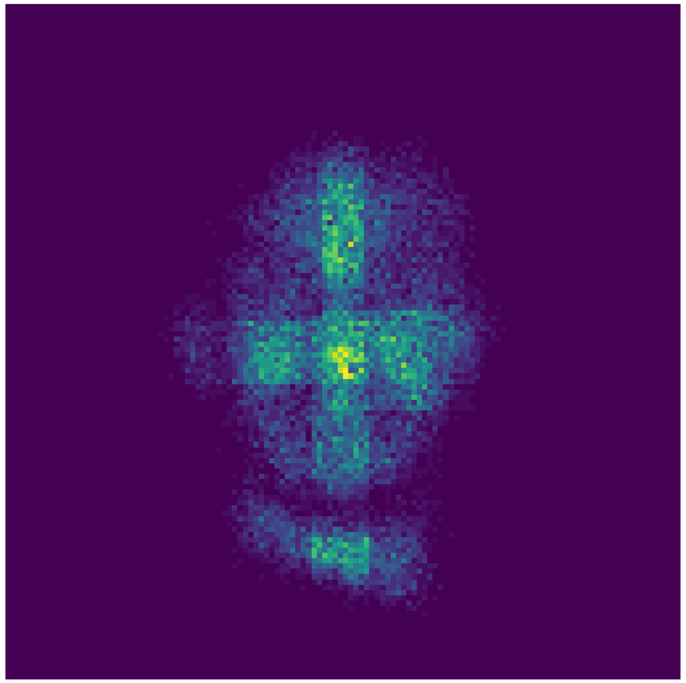


The “low counts” imaging process



The problem of “unblind” deconvolution

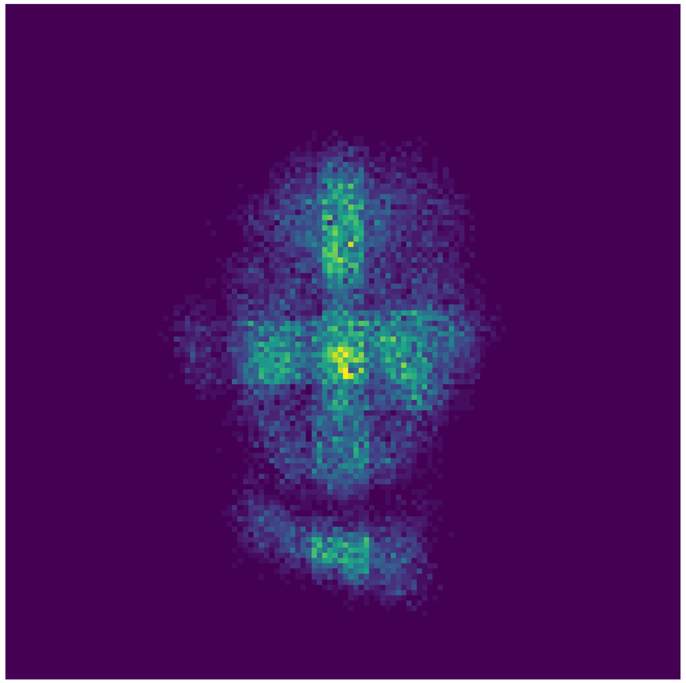
Counts



Ground Truth (GT)

The problem of “unblind” deconvolution

Counts



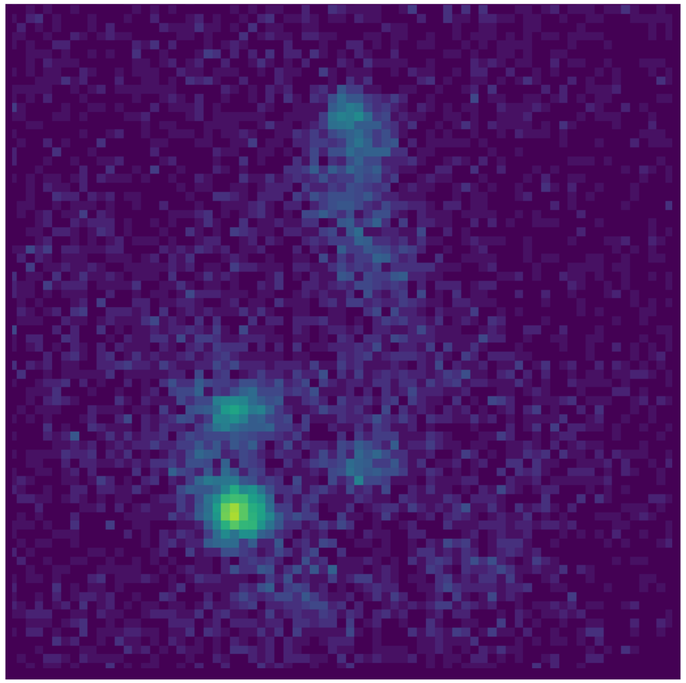
?

Ground Truth (GT)

CAUTION:
Selfies in X-rays or γ-rays not recommended...

The problem of “unblind” deconvolution

Counts



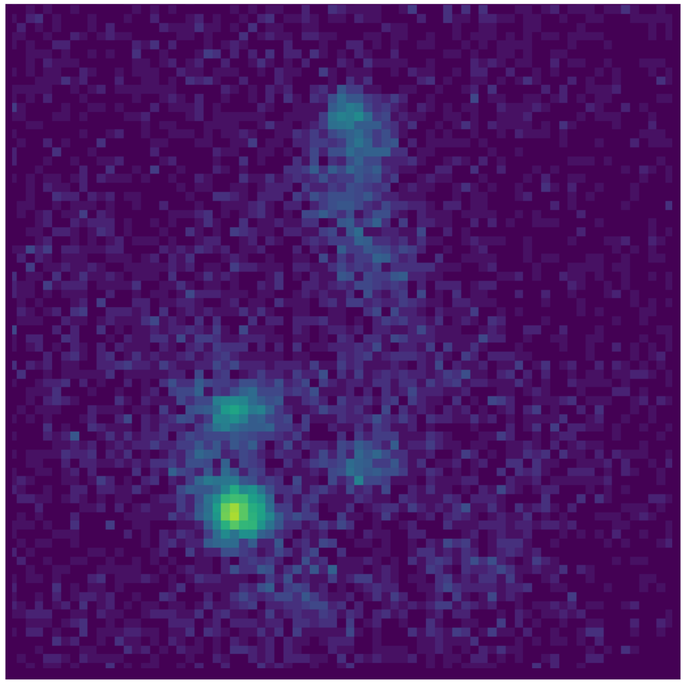
Low counts astronomical images…

?

Reconstruction

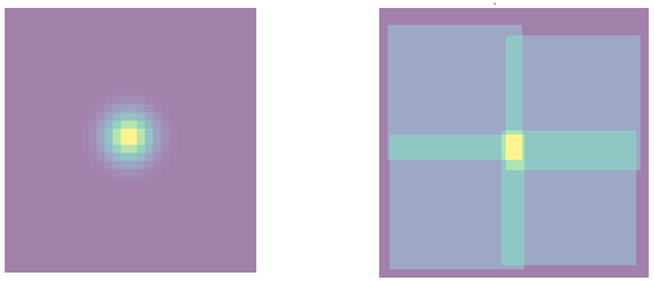
The problem of “unblind” deconvolution

Counts



Low counts astronomical images…

?

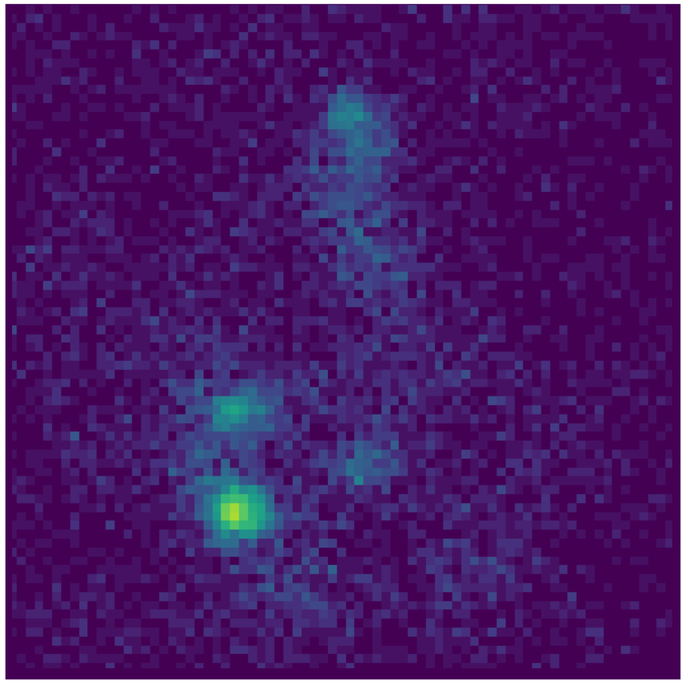


“Unblind”: PSF and exposure are known or can be simulated

Reconstruction

The problem of “unblind” deconvolution

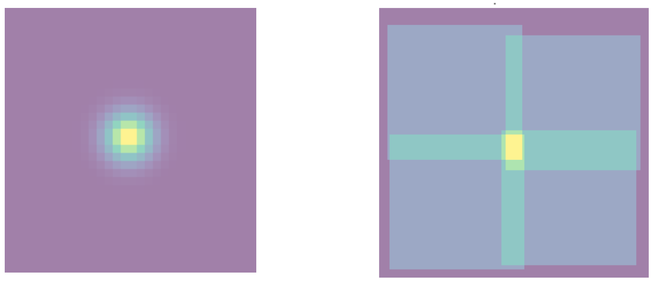
Counts



Low counts astronomical images…

$$\mathscr{L}\left(\mathbf{d}\,|\,\lambda\right) = \prod_{i}^{N} \frac{\lambda_i^{d_i} \mathrm{e}^{-d_i}}{d_i!} \qquad \lambda = \mathbf{x} \circledast \mathrm{PSF}$$

$$\lambda_i = \sum_{k} \left(e_i \cdot (x_i \bullet b_i)\right) p_{i-k}$$



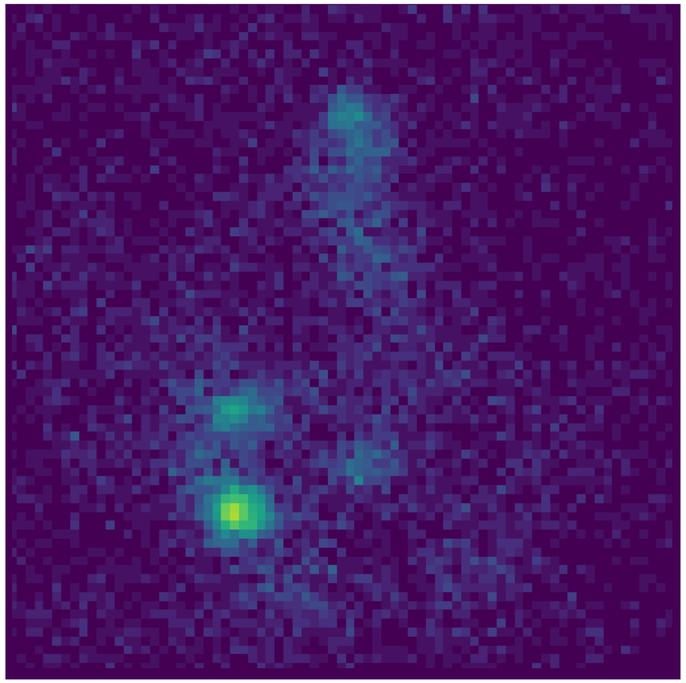
“Unblind”: PSF and exposure are known or can be simulated

Reconstruction

Needs to be reconstructed using statistical methods

The problem of “unblind” deconvolution

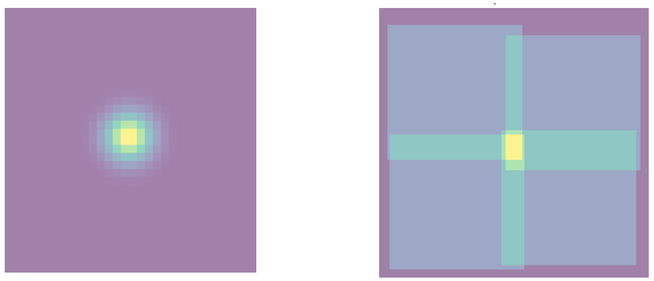
Counts



Low counts astronomical images…

$$\mathscr{L}\left(\mathbf{d}\,|\,\lambda\right) = \prod_{i}^{N} \frac{\lambda_i^{d_i} \mathrm{e}^{-d_i}}{d_i!} \qquad \lambda = \mathbf{x} \circledast \mathrm{PSF}$$

$$\lambda_i = \sum_{k} \left(e_i \cdot (x_i \bullet b_i)\right) p_{i-k}$$



“Unblind”: PSF and exposure are known or can be simulated

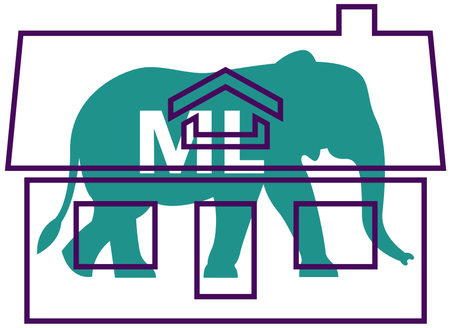
Reconstruction

Needs to be reconstructed using statistical methods

“Ill-posed inference problem”

Why not just use ML?

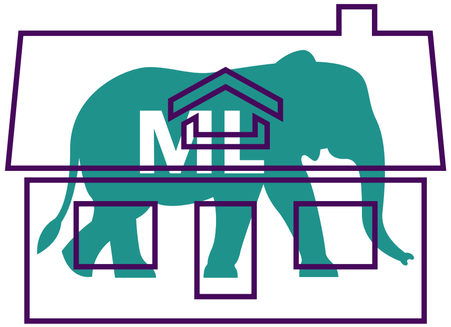
- **Lack of training data:** astronomical images from X-ray and gamma-ray observatories are expensive.
- **There is no ground truth:** we don't know what the sky "really" looks like. We only have the data that is observed and distorted through our instruments.
- **Standard deep CNN architectures are not suited** for the deconvolution task. They can learn a "pseudo inverse PSF kernel" in the spatial domain, but this requires large kernels and adapted architectures [Xu et al. 2014].
- For Poisson data one is always in the low signal to noise regime. So **specifying "Type I" errors (false detections) is important**. Ideally one can keep track of the full likelihood.
- Needs to learn information that is available a priory, e.g. Poisson statistics of the data.
- However **"transfer learning" and simulated training data could maybe work**, or hybrid approaches…



The "MLephant in the room" question…

Why not just use ML?

- **Lack of training data:** astronomical images from X-ray and gamma-ray observatories are expensive.
- **There is no ground truth:** we don't know what the sky "really" looks like. We only have the data that is observed and distorted through our instruments.
- **Standard deep CNN architectures are not suited** for the deconvolution task. They can learn a "pseudo inverse PSF kernel" in the spatial domain, but this requires large kernels and adapted architectures [Xu et al. 2014].
- For Poisson data one is always in the low signal to noise regime. So **specifying "Type I" errors (false detections) is important**. Ideally one can keep track of the full likelihood.
- Needs to learn information that is available a priory, e.g. Poisson statistics of the data.
- However **"transfer learning" and simulated training data could maybe work**, or hybrid approaches…



The "MLephant in the room" question…

Methods

Some math…

For the counts image assume per pixel **Poisson** likelihood:

$$\mathscr{L}\left(\mathbf{d}\,|\,\lambda\right) = \prod_{i}^{N} \frac{\lambda_i^{d_i} \mathrm{e}^{-d_i}}{d_i!}$$

Some math…

For the counts image assume per pixel **Poisson** likelihood:

$$\mathscr{L}\left(\mathbf{d}\,|\,\lambda\right) = \prod_i^N \frac{\lambda_i^{d_i} \mathrm{e}^{-d_i}}{d_i!}$$

As usual: take the **negative log-likelihood**, in Astronomy often call "Cash" statistics

$$\mathscr{C}\left(\mathbf{d}\,|\,\lambda\right) = \sum_i^N \left(\lambda_i - d_i \log \lambda_i\right)$$

Some math…

For the counts image assume per pixel **Poisson** likelihood:

$$\mathscr{L}\left(\mathbf{d}\,|\,\lambda\right) = \prod_i^N \frac{\lambda_i^{d_i} \mathrm{e}^{-d_i}}{d_i!}$$

As usual: take the **negative log-likelihood**, in Astronomy often call "Cash" statistics

$$\mathscr{C}\left(\mathbf{d}\,|\,\lambda\right) = \sum_i^N \left(\lambda_i - d_i \log \lambda_i\right)$$

λ are the "model counts"

$$\lambda = \mathbf{x} \circledast \mathrm{PSF}$$

x is the reconstructed image we are looking for. Consider each pixel x_i as independent parameter in the model…

Some math…

For the counts image assume per pixel **Poisson** likelihood:

$$\mathscr{L}\left(\mathbf{d}\,|\,\lambda\right) = \prod_i^N \frac{\lambda_i^{d_i} \mathrm{e}^{-d_i}}{d_i!}$$

As usual: take the **negative log-likelihood**, in Astronomy often call "Cash" statistics

$$\mathscr{C}\left(\mathbf{d}\,|\,\lambda\right) = \sum_i^N \left(\lambda_i - d_i \log \lambda_i\right)$$

E.g. could be solved by **"Gradient Descent"**…

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \cdot \frac{\partial \mathscr{C}\left(\mathbf{d}\,|\,\mathbf{x}\right)}{\partial x_i}$$

λ are the "model counts"

$$\lambda = \mathbf{x} \circledast \mathrm{PSF}$$

x is the reconstructed image we are looking for. Consider each pixel x_i as independent parameter in the model…

Some math…

For the counts image assume per pixel **Poisson** likelihood:

$$\mathscr{L}\left(\mathbf{d}\,|\,\lambda\right) = \prod_i^N \frac{\lambda_i^{d_i} \mathrm{e}^{-d_i}}{d_i!}$$

As usual: take the **negative log-likelihood**, in Astronomy often call "Cash" statistics

$$\mathscr{C}\left(\mathbf{d}\,|\,\lambda\right) = \sum_i^N \left(\lambda_i - d_i \log \lambda_i\right)$$

λ are the "model counts"

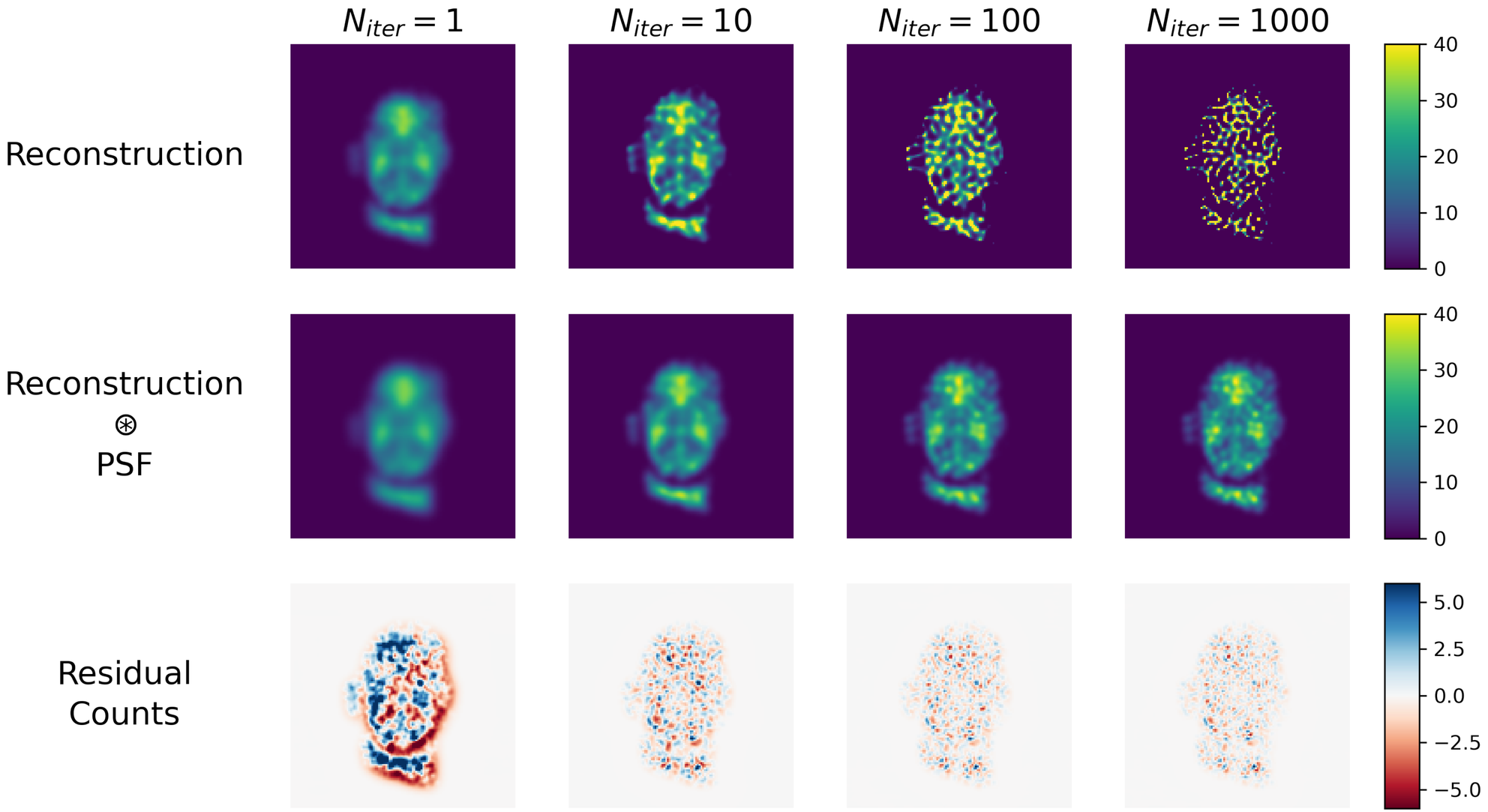
$$\lambda = \mathbf{x} \circledast \mathrm{PSF}$$

x is the reconstructed image we are looking for. Consider each pixel x_i as independent parameter in the model…

…or using **Expectation Maximisation (EM)**
Proposed by [Richardson 1972] & [Lucy 1974]

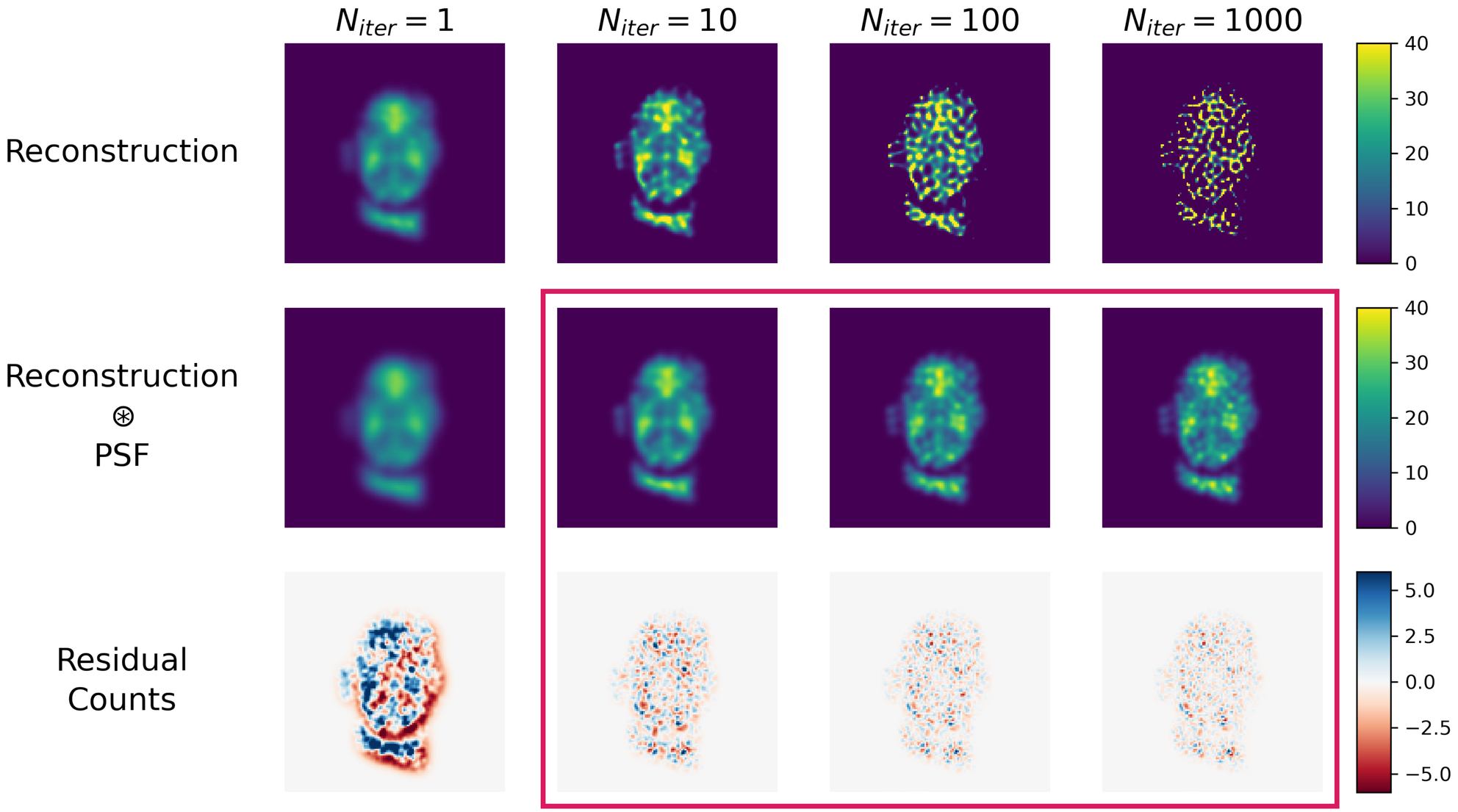
$$\mathbf{x}_{n+1} = \mathbf{x}_n \frac{\mathbf{d}}{\mathbf{x}_n \circledast \mathrm{PSF}} \circledast \mathrm{PSF}^{\mathrm{T}}$$

RL reconstruction quality



Uses [skimage.restoration.richardson_lucy]

RL reconstruction quality



All show good residuals and model counts. But reconstructions very different…

Possible solutions

Pragmatic: early stopping, limit number of iterations until one visually “likes” the result. Often used in literature, good enough for showing nice images…

Possible solutions

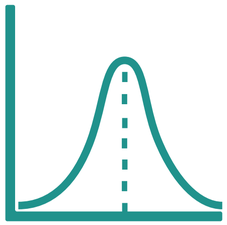
Pragmatic: early stopping, limit number of iterations until one visually “likes” the result. Often used in literature, good enough for showing nice images…

“**Engineered”:** adding additional regularization terms to the likelihood function, such [total variation], [Thikonov regularization] etc. often require additional “hand tuned” parameters…

Possible solutions

Pragmatic: early stopping, limit number of iterations until one visually "likes" the result. Often used in literature, good enough for showing nice images…

"**Engineered":** adding additional regularization terms to the likelihood function, such [total variation], [Thikonov regularization] etc. often require additional "hand tuned" parameters



"**Statistically closed":** include prior information or assumptions and change to a fully Bayesian treatment of the problem…

LIRA method

(L)ow counts (I)mage (R)econstruction and (A)analysis

Objective function **extended by a log-prior term,** only depending on λ :

$$\mathscr{L}\left(\lambda \,|\, \mathbf{d}\right) = \mathscr{C}\left(\mathbf{d} \,|\, \lambda\right) + \mathscr{P}(\lambda)$$

Log-Posterior Log-Likelihood

LIRA method

(L)ow counts (I)mage (R)reconstruction and (A)analysis

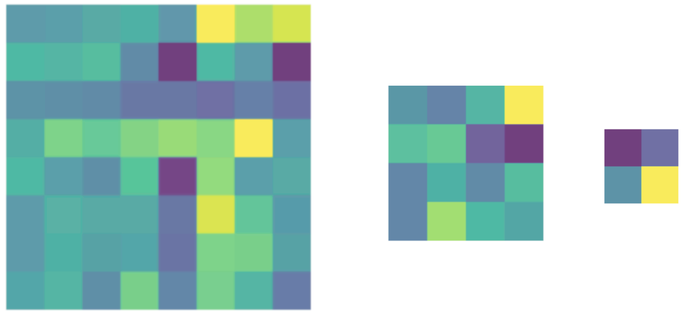
Objective function **extended by a log-prior term,** only depending on λ :

$$\mathscr{L}\left(\lambda \,|\, \mathbf{d}\right) = \mathscr{C}\left(\mathbf{d} \,|\, \lambda\right) + \mathscr{P}(\lambda)$$

Log-Posterior

Log-Likelihood

Log-Prior



A special “multi-scale”prior to achieve **smoothness on multiple scales.**
Initial idea and implementation by [Esch et al. 2004].
More detailed explanation follows…

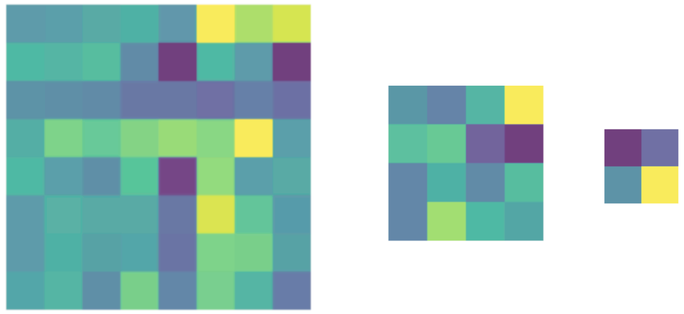
LIRA method

(L)ow counts (I)mage (R)econstruction and (A)analysis

Objective function **extended by a log-prior term,** only depending on λ :

$$\mathscr{L}\left(\lambda \,|\, \mathbf{d}\right) = \mathscr{C}\left(\mathbf{d} \,|\, \lambda\right) + \mathscr{P}(\lambda)$$

Log-Posterior Log-Likelihood **Log-Prior**



A special "multi-scale"prior to achieve **smoothness on multiple scales.**
Initial idea and implementation by [Esch et al. 2004].
More detailed explanation follows…

Model counts extend by **exposure e** and optional **baseline ("background") component b** by [Connors et al. 2011]

$$\lambda = [(\mathbf{x} + \mathbf{b}) \cdot \mathbf{e}] \circledast \mathrm{PSF}$$

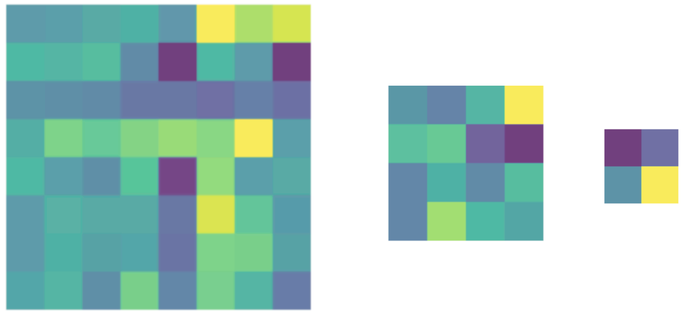
LIRA method

(L)ow counts (I)mage (R)econstruction and (A)analysis

Objective function **extended by a log-prior term,** only depending on λ :

$$\mathscr{L}\left(\lambda \,|\, \mathbf{d}\right) = \mathscr{C}\left(\mathbf{d} \,|\, \lambda\right) + \mathscr{P}(\lambda)$$

Log-Posterior — Log-Likelihood — **Log-Prior**



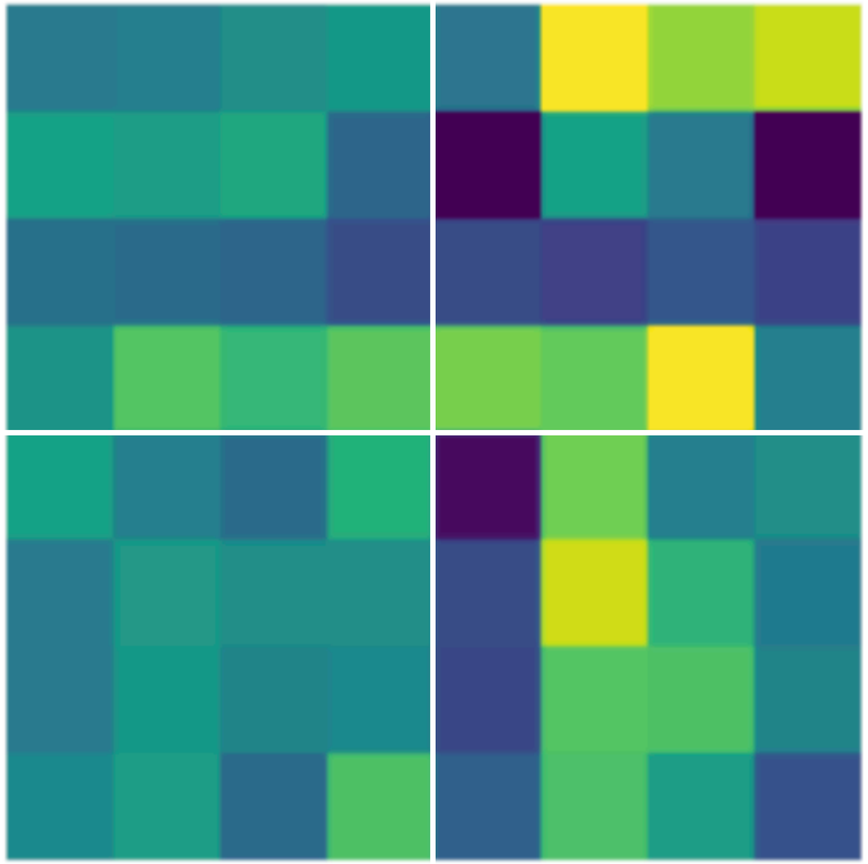
A special "multi-scale"prior to achieve **smoothness on multiple scales.**
Initial idea and implementation by [Esch et al. 2004].
More detailed explanation follows…

Model counts extend by **exposure e** and optional **baseline ("background") component b** by [Connors et al. 2011]

$$\lambda = [(\mathbf{x} + \mathbf{b}) \cdot \mathbf{e}] \circledast \mathrm{PSF}$$

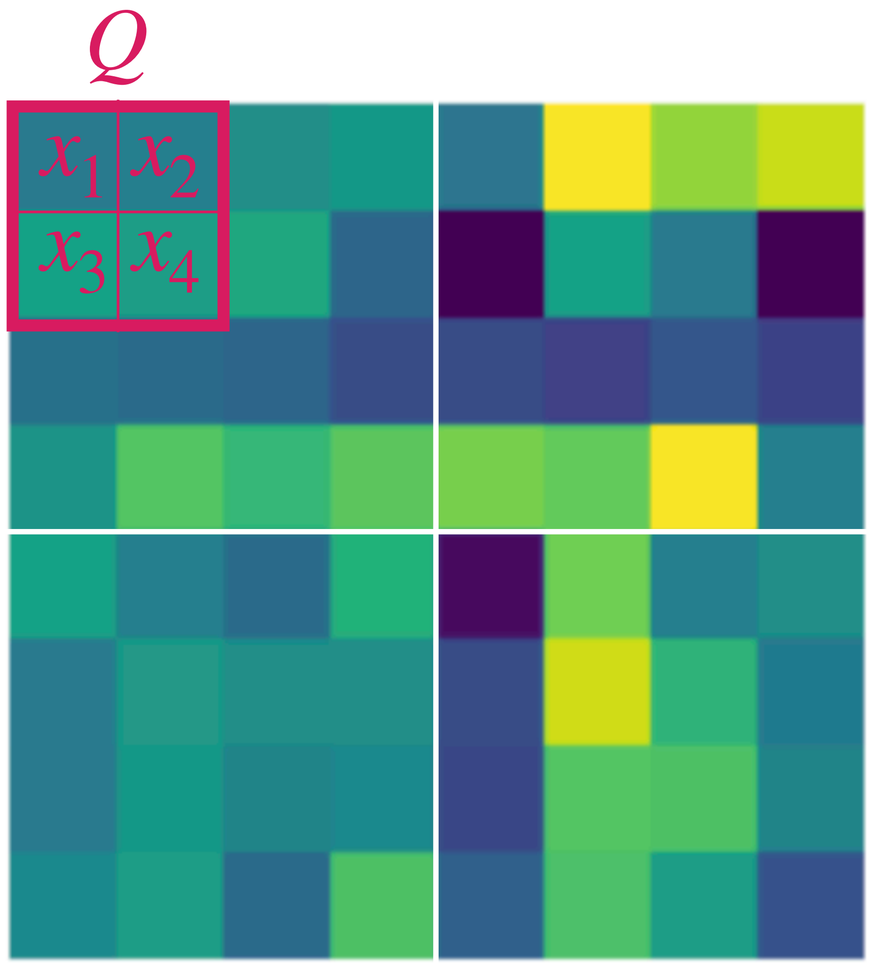
Change from EM to **MCMC sampling** from the log-posterior. Effectively **sampling a series of images**…

Multiscale decomposition



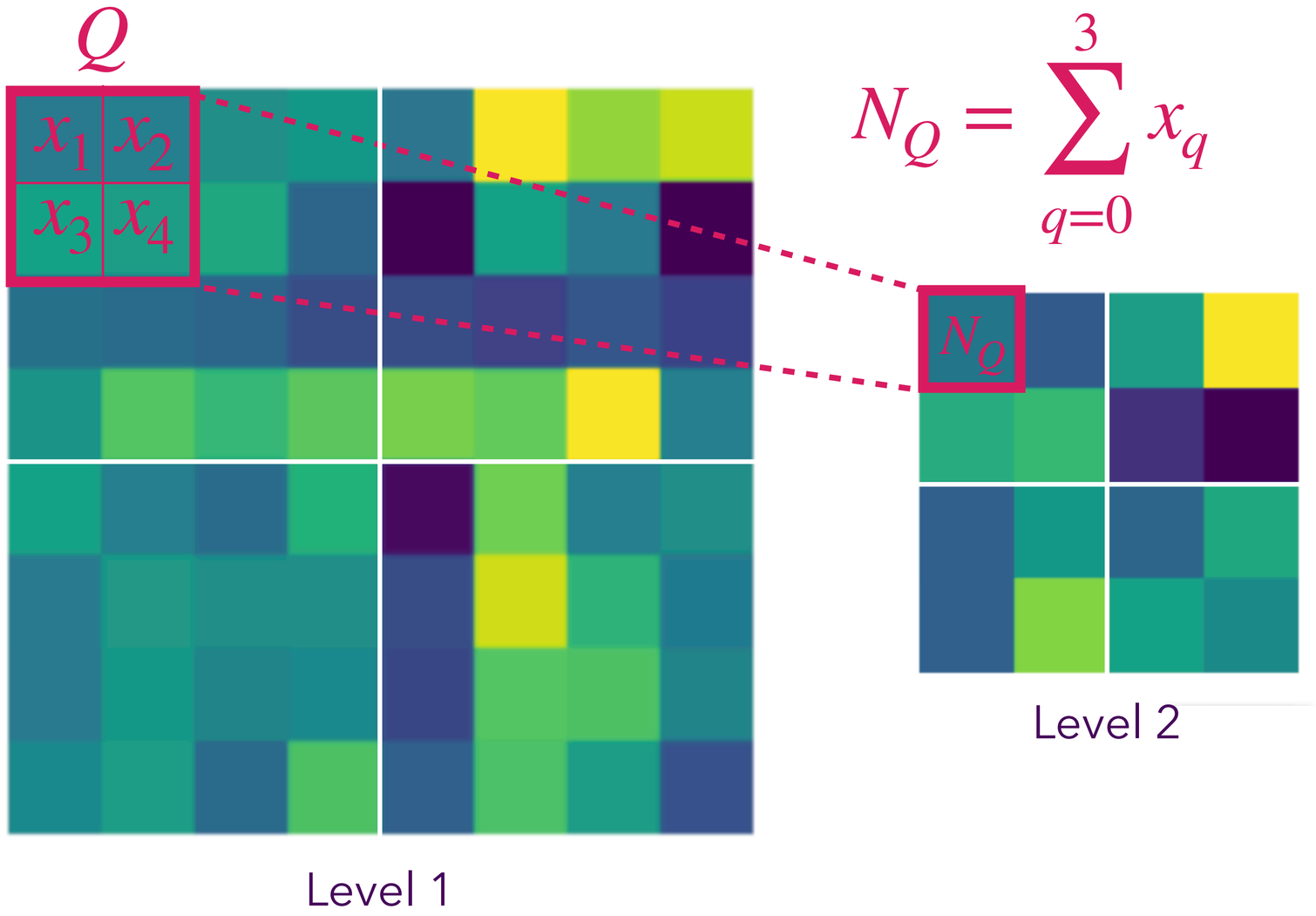
Level 1

Multiscale decomposition



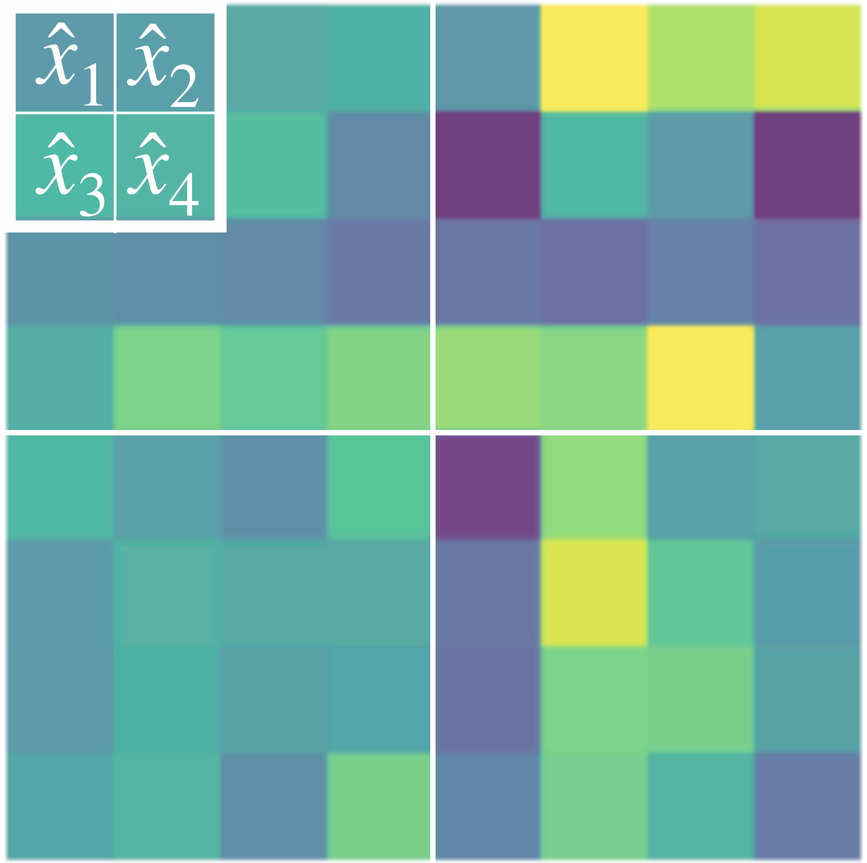
Level 1

Multiscale decomposition



Multiscale decomposition

$$\hat{x}_q = x_q / N_Q$$



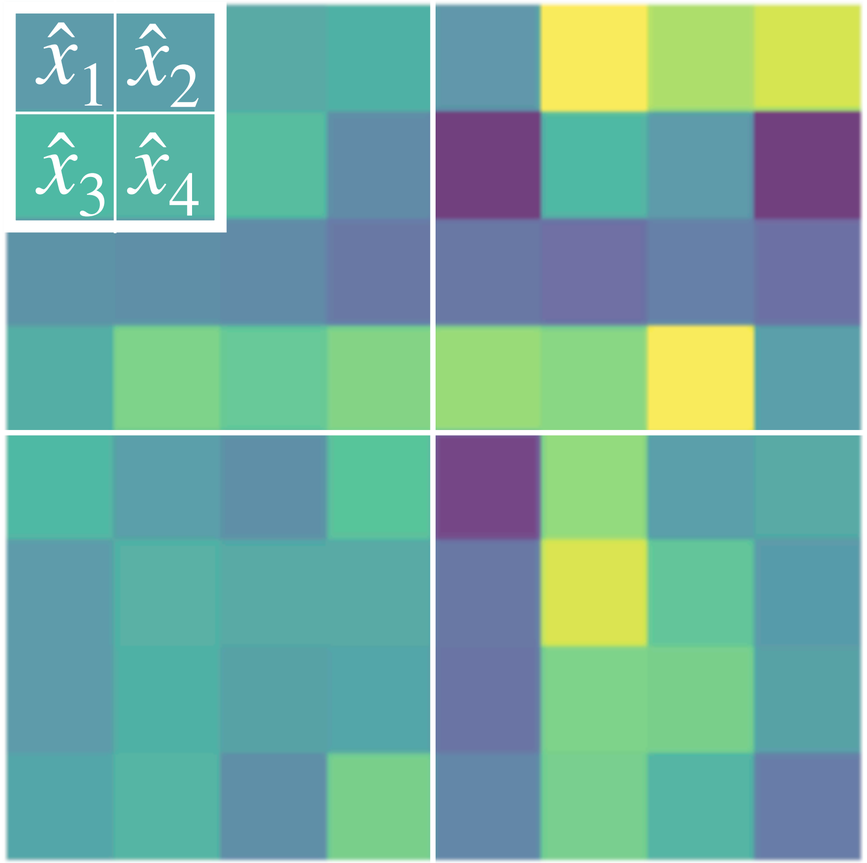
Level 1

$$N_Q = \sum_{q=0}^{3} x_q$$

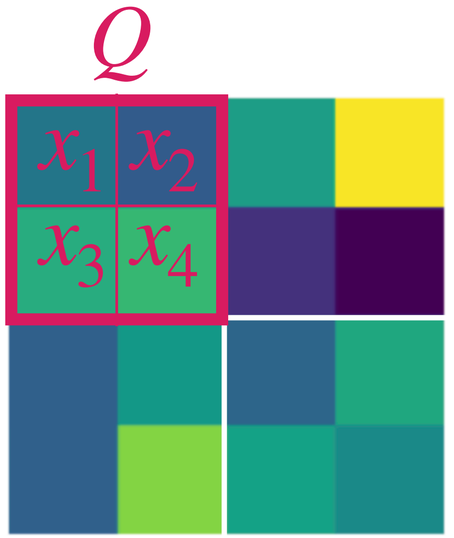
Level 2

Multiscale decomposition

$$\hat{x}_q = x_q / N_Q$$



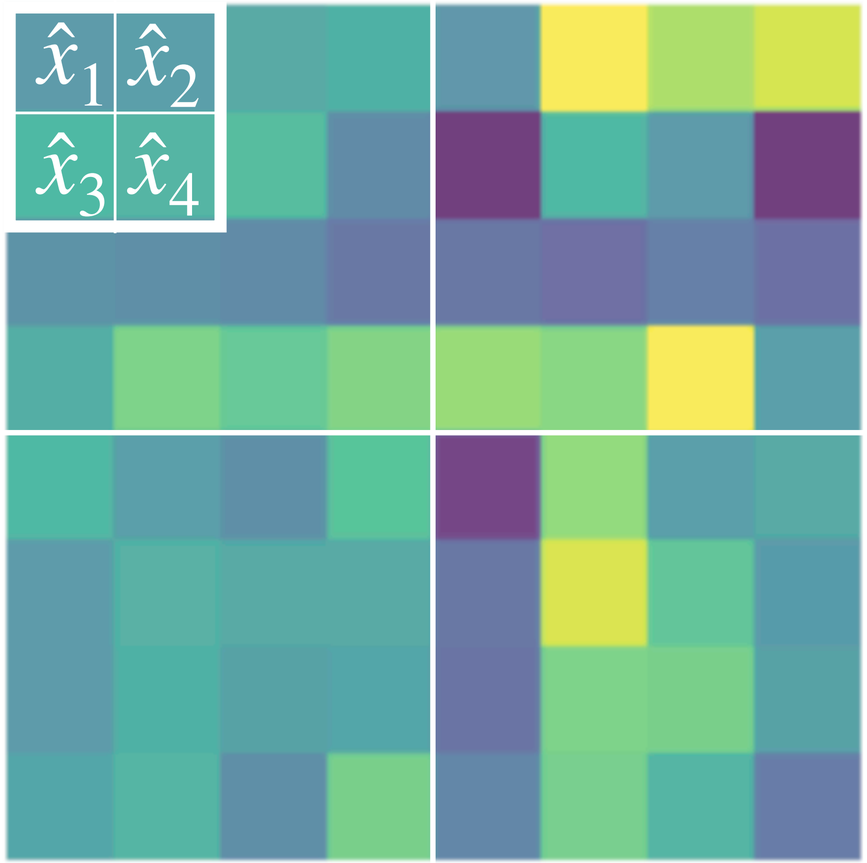
Level 1



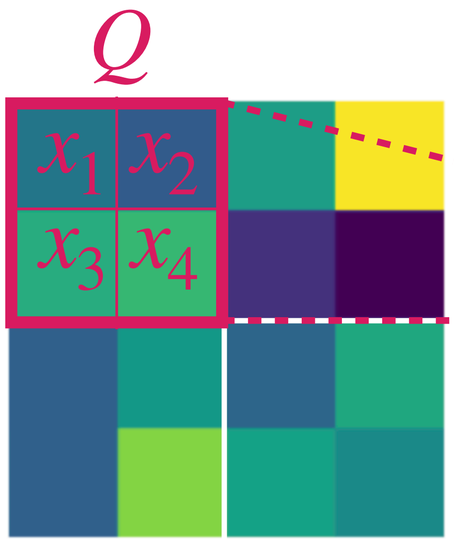
Level 2

Multiscale decomposition

$$\hat{x}_q = x_q / N_Q$$



Level 1



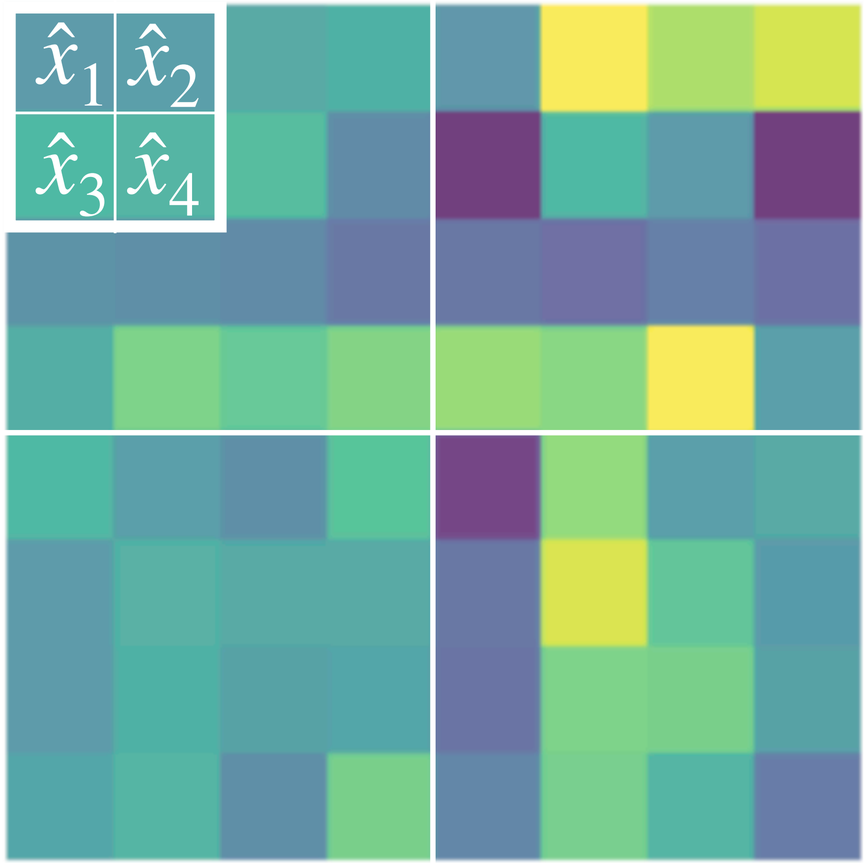
Level 2

$$N_Q = \sum_{a=0}^{3} x_q$$

Level k

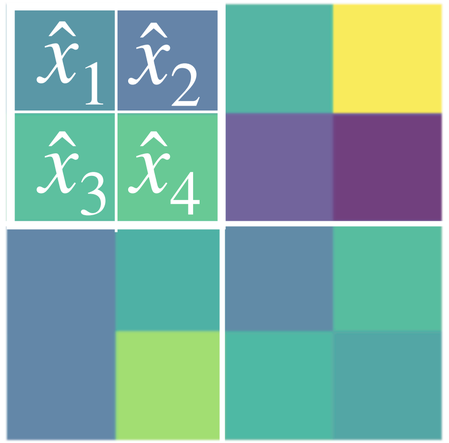
Multiscale decomposition

$$\hat{x}_q = x_q / N_Q$$



Level 1

$$\hat{x}_q = x_q / N_Q$$



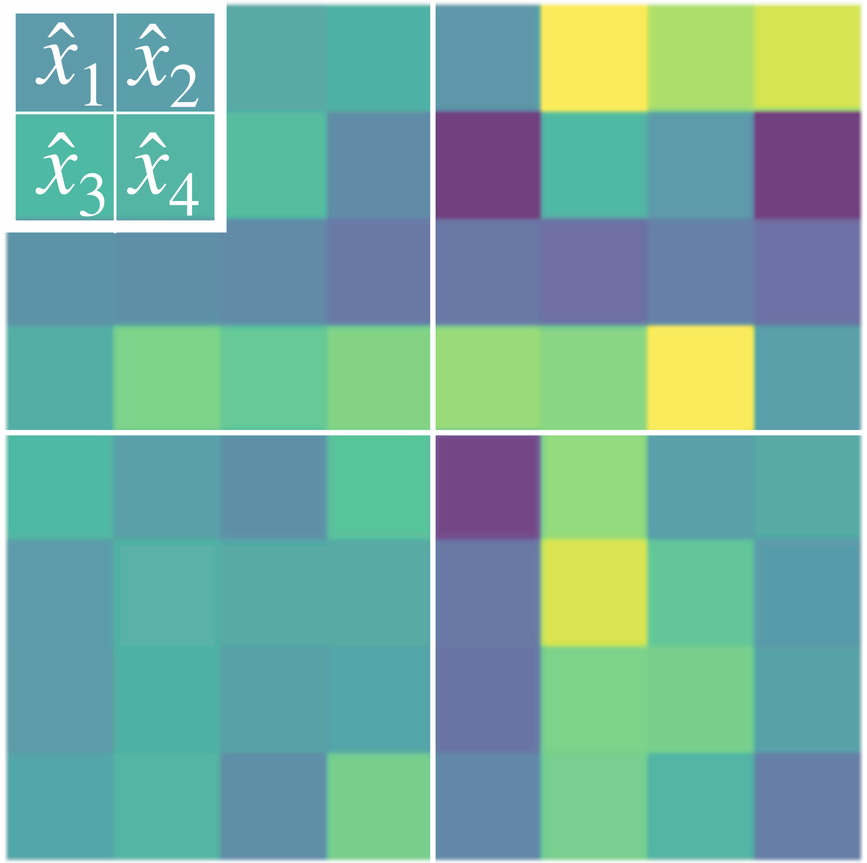
Level 2

$$N_Q = \sum_{a=0}^{3} x_q$$

Level k

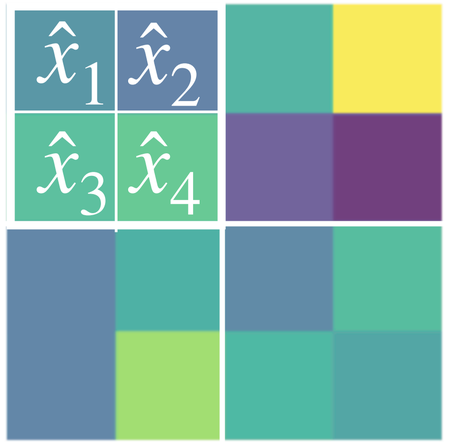
Multiscale decomposition

$$\hat{x}_q = x_q / N_Q$$

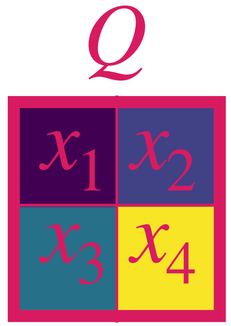


Level 1

$$\hat{x}_q = x_q / N_Q$$



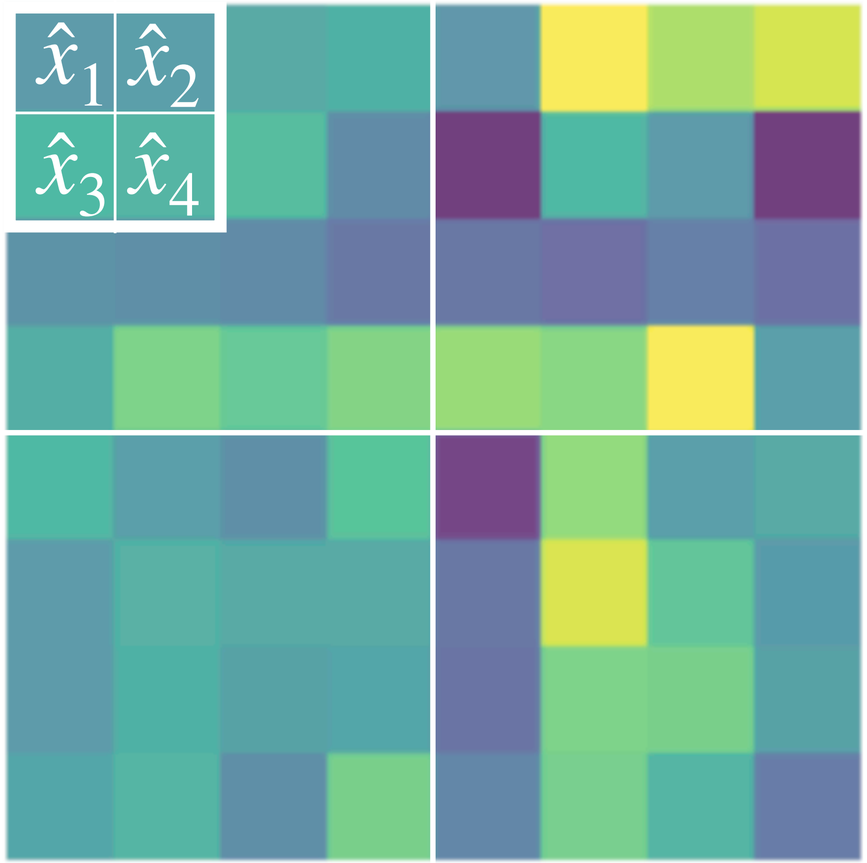
Level 2



Level k

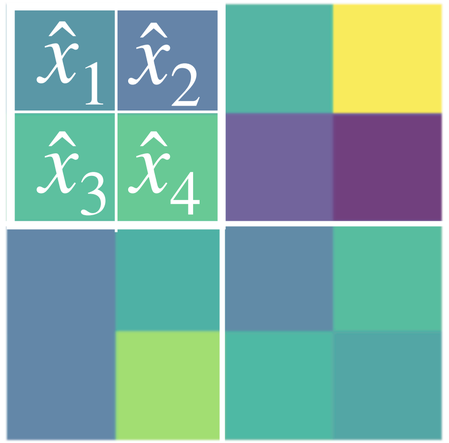
Multiscale decomposition

$$\hat{x}_q = x_q / N_Q$$



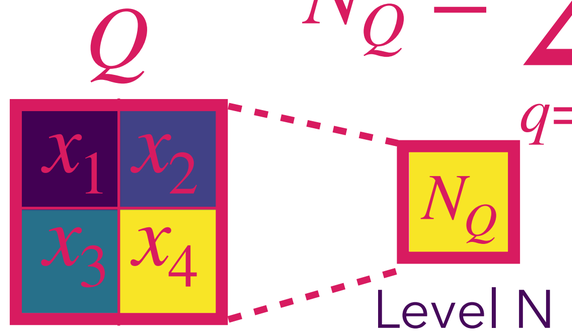
Level 1

$$\hat{x}_q = x_q / N_Q$$



Level 2

$$N_Q = \sum_{q=0}^{3} x_q$$

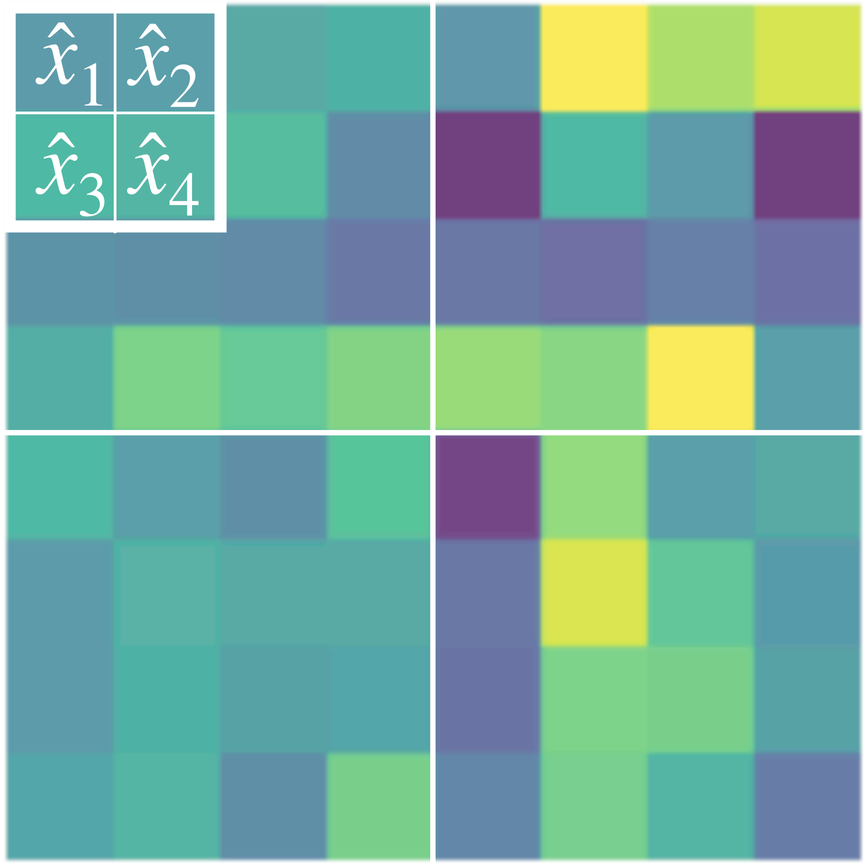


Level k

Level N

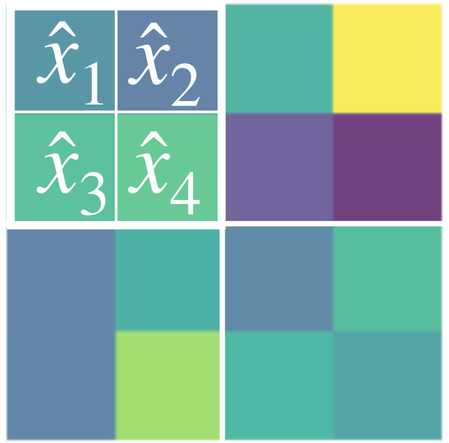
Multiscale decomposition

$$\hat{x}_q = x_q / N_Q$$



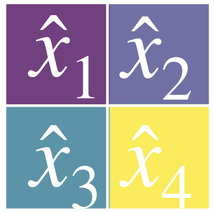
Level 1

$$\hat{x}_q = x_q / N_Q$$



Level 2

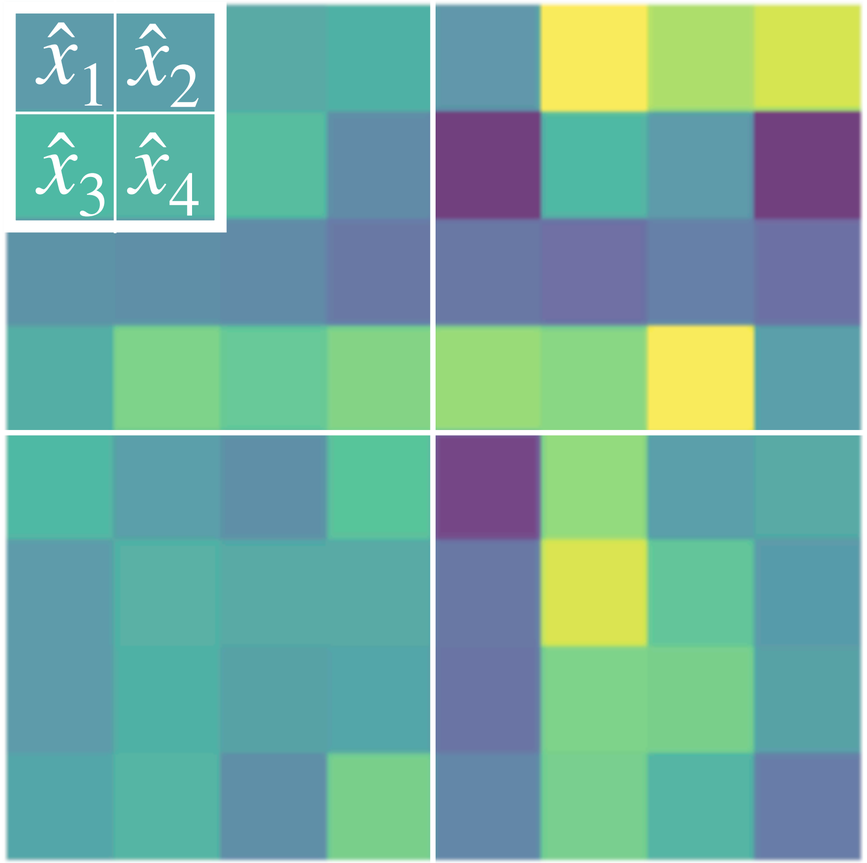
$$\hat{x}_q = x_q / N_Q$$



Level k

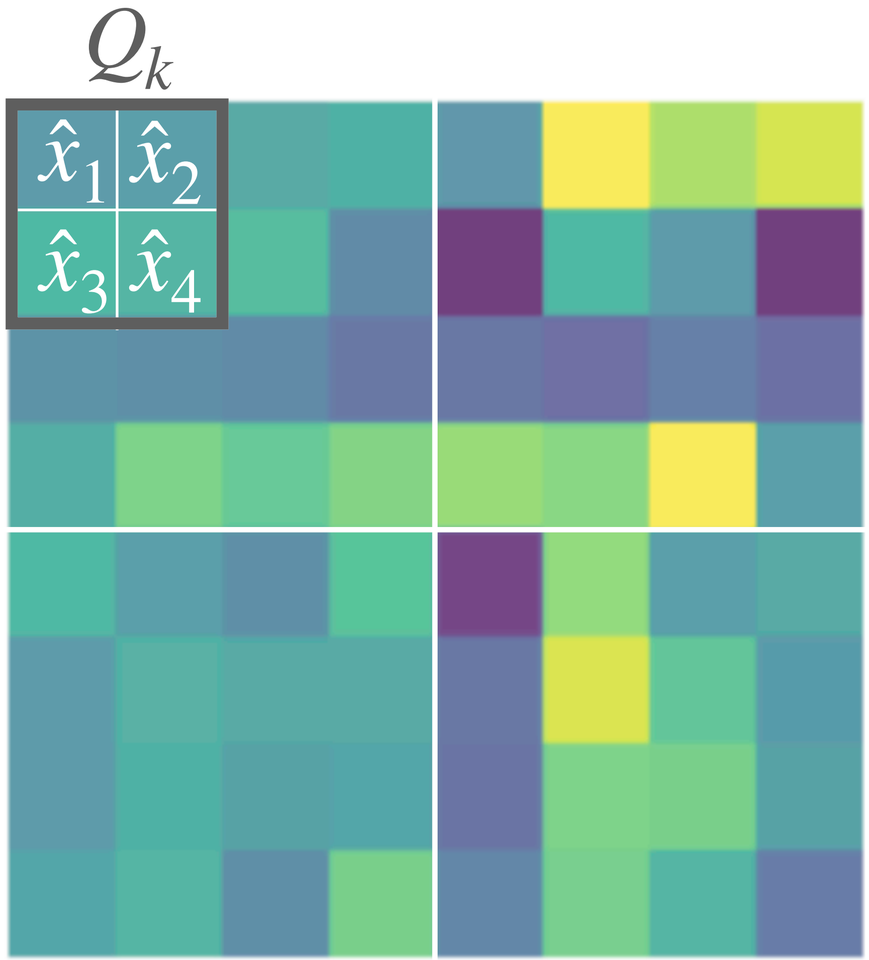
Level N

Multiscale prior



Level k

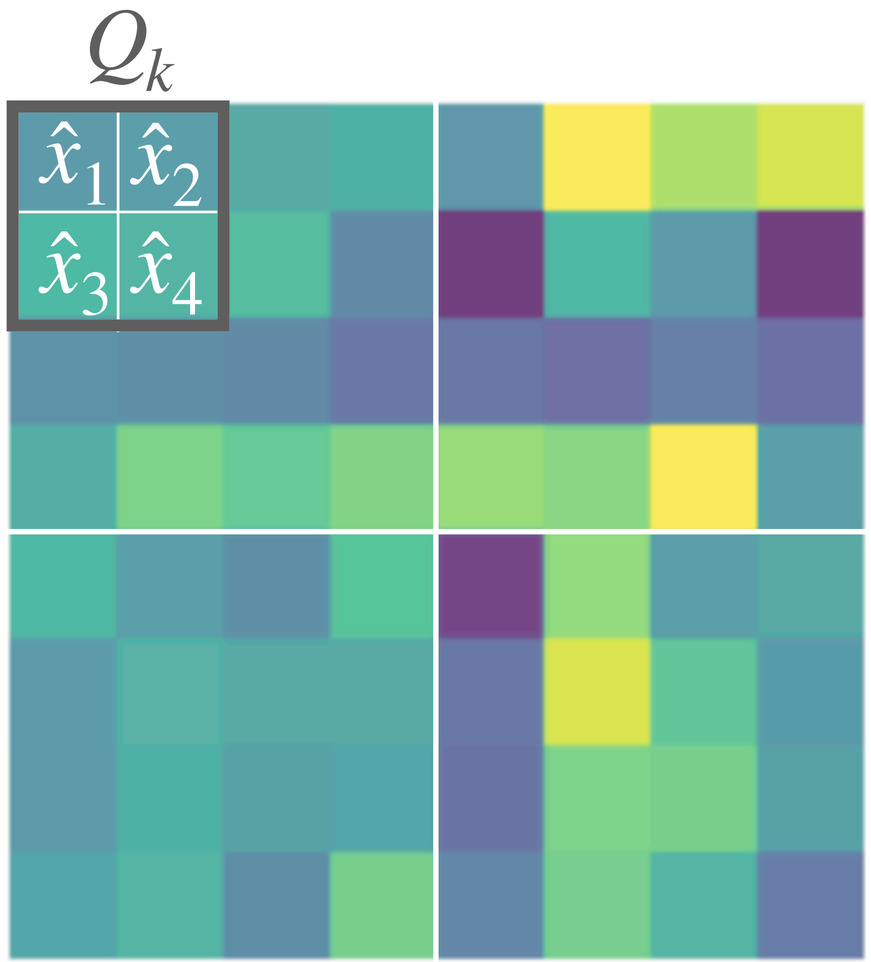
Multiscale prior



The **Dirichlet distribution** is a multivariate distribution, where all **variables sum to unity**. And the larger the parameter α_k the more similar the variables are.

$$Q_k \propto \text{Dirichlet}(\alpha_k, \alpha_k, \alpha_k, \alpha_k)$$

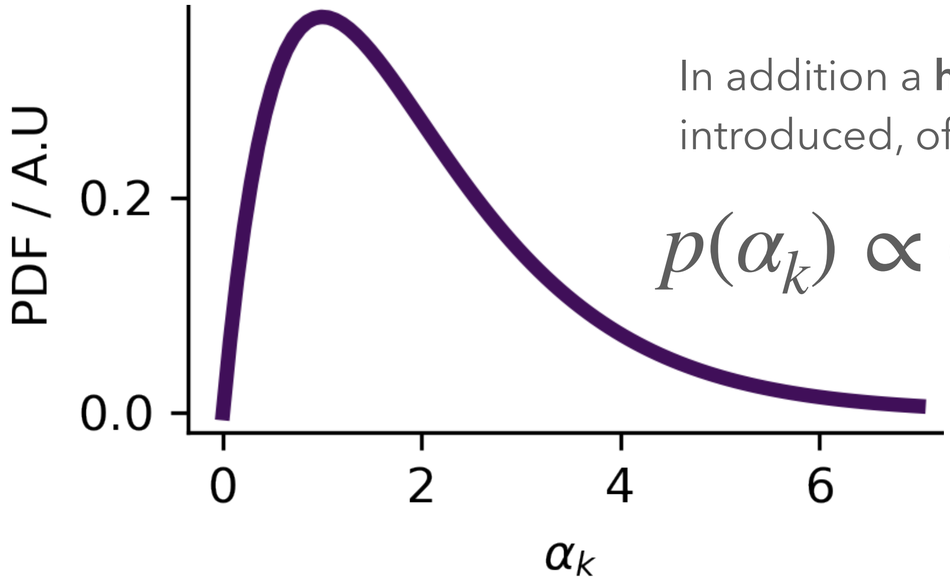
Multiscale prior



Level k

The **Dirichlet distribution** is a multivariate distribution, where all **variables sum to unity**. And the larger the parameter α_k the more similar the variables are.

$$Q_k \propto \text{Dirichlet}(\alpha_k, \alpha_k, \alpha_k, \alpha_k)$$



In addition a **hyper-prior** on α_k is introduced, of the empirical form:

$$p(\alpha_k) \propto \exp(-\delta\alpha^3/3)$$

Pylira Python Package

Pylira Python Package

- The LIRA method was **initially implemented as an R package** by [Connors et al. 2011].
  Original code available at https://github.com/astrostat/LIRA
- Meanwhile Python has become the favored language of choice for astronomical data analysis.
- **Pylira is a Python wrapper** around the initial C implementation implemented via [pybind11] and based on the [Astropy affiliated package template]
- Additional dependencies are Numpy, Scipy, Astropy, Matplotlib, …
- Github: https://github.com/astrostat/pylira
- Docs: https://pylira.readthedocs.io/en/latest/

pybind11

Pylira Python Package

- The LIRA method was **initially implemented as an R package** by [Connors et al. 2011].
  Original code available at https://github.com/astrostat/LIRA
- Meanwhile Python has become the favored language of choice for astronomical data analysis.
- **Pylira is a Python wrapper** around the initial C implementation implemented via [pybind11] and based on the [Astropy affiliated package template]
- Additional dependencies are Numpy, Scipy, Astropy, Matplotlib, …
- Github: https://github.com/astrostat/pylira
- Docs: https://pylira.readthedocs.io/en/latest/

pybind11

NumPy

SciPy

matplotlib

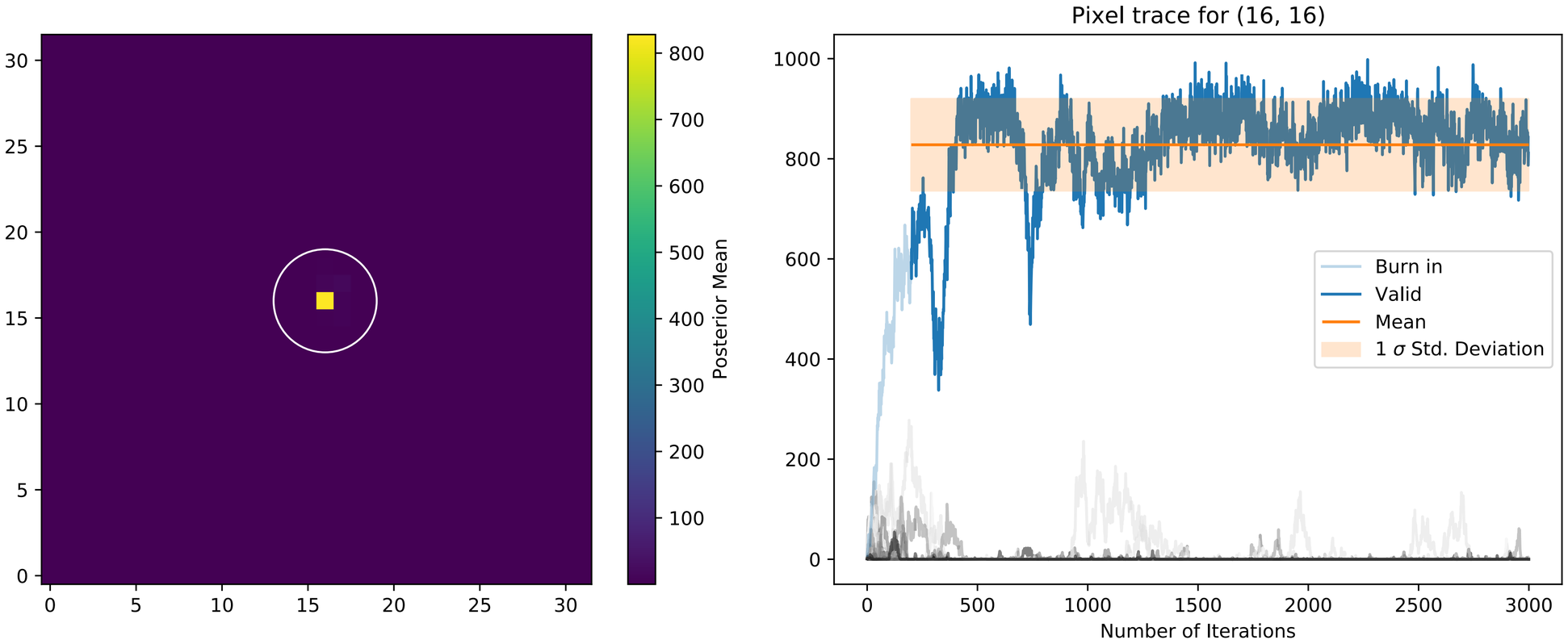
astropy-powered
astropy.org

A short code example...

```
import numpy as np
from pylira import LIRADeconvolver
from pylira.data import point_source_gauss_psf

random_state = np.RandomState(372)

# load built in test dataset
data = point_source_gauss_psf()

data["flux_init"] = random_state.gamma(10, size=(32, 32))

# sklearn inspired API, take config on init
deconvolve = LIRADeconvolver(
    alpha_init=np.ones(5),
    n_iter_max=5_000,
    n_burn_in=500,
    random_state=random_state,
)

# run algorithm
result = deconvolve.run(data=data)

# serialise to FITS
result.write("pylira-result.fits")
```

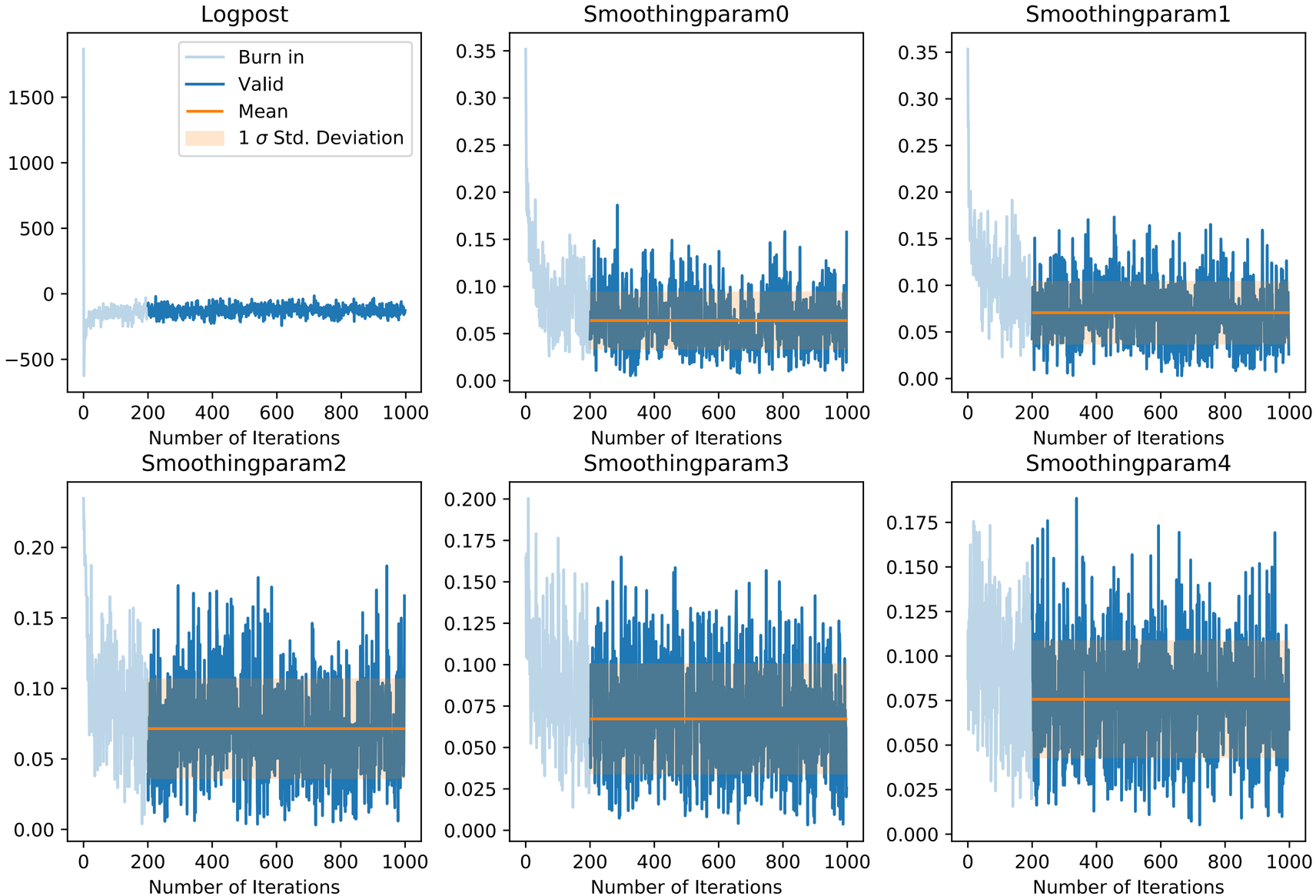
[From Simple Point Source Tutorial]

Diagnostic plot I



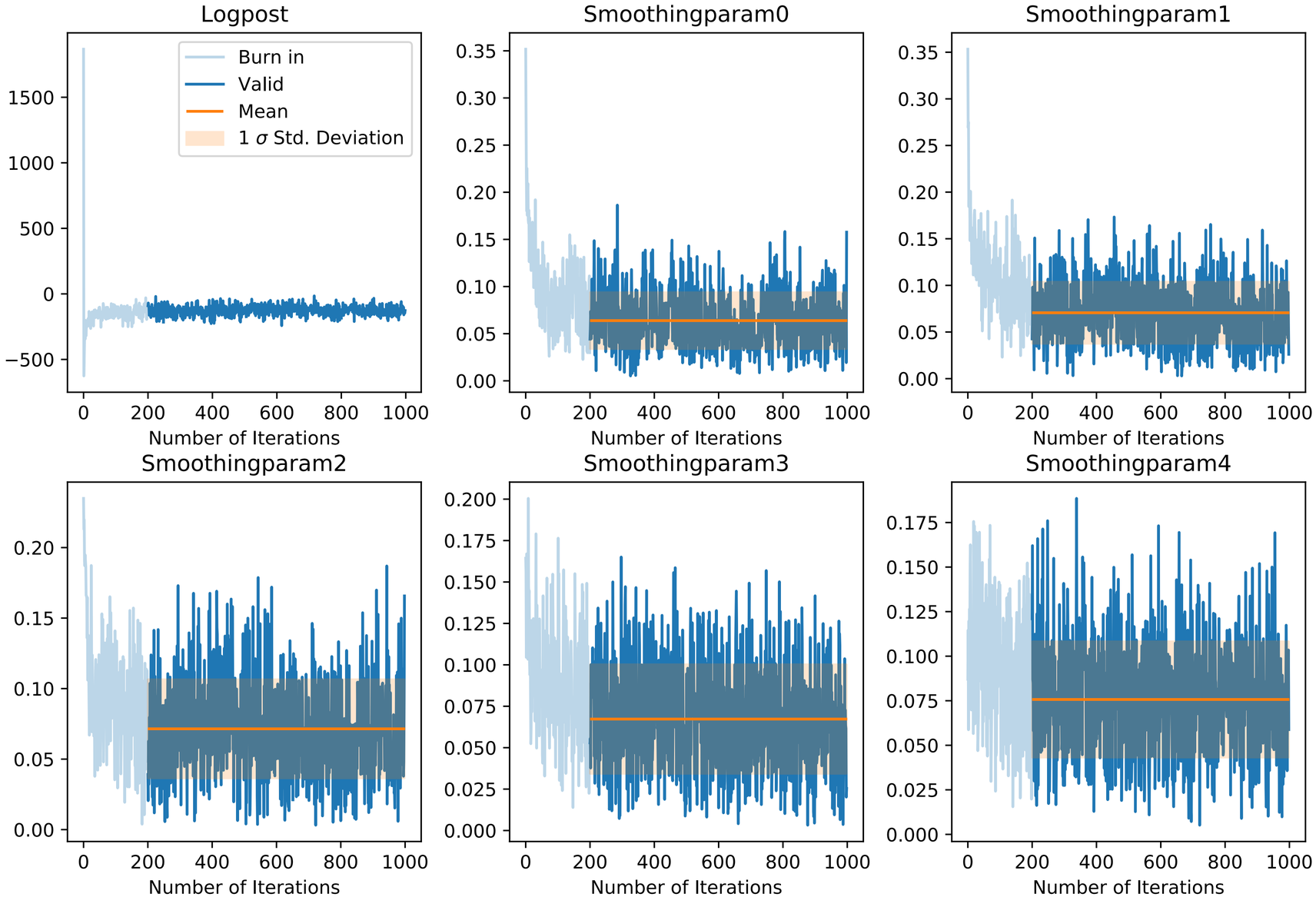

```
from pylira import LIRADeconvolverResult

# read result back from disk
result = LIRADeconvolverResult.read("pylira-result.fits")

# diagnostic plot
result.plot_pixel_traces_region(center_pix=(16, 16), radius_pix=2)
```

Diagnostic plot II




```python
from pylira import LIRADeconvolverResult

# read result back from disk
result = LIRADeconvolverResult.read(
    "pylira-result.fits"
)

# diagnostic plot
result.plot_parameter_traces()
```

Diagnostic plot II

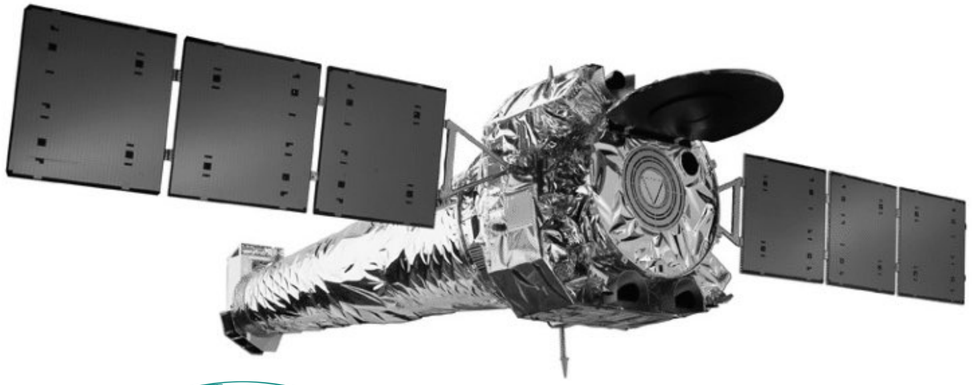



```python
from pylira import LIRADeconvolverResult

# read result back from disk
result = LIRADeconvolverResult.read(
    "pylira-result.fits"
)

# diagnostic plot
result.plot_parameter_traces()
```

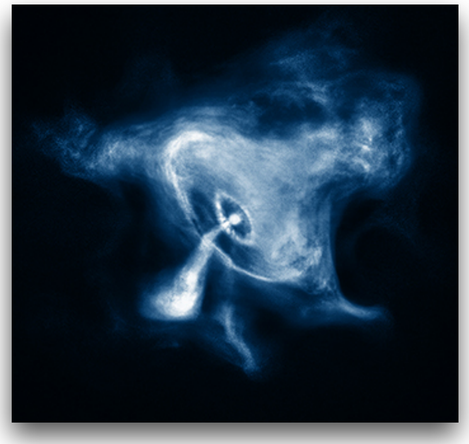
Goal: provide as much additional diagnostic information as possible! It is important to check and trust sampling results!

Application Example

CHANDRA Observatory



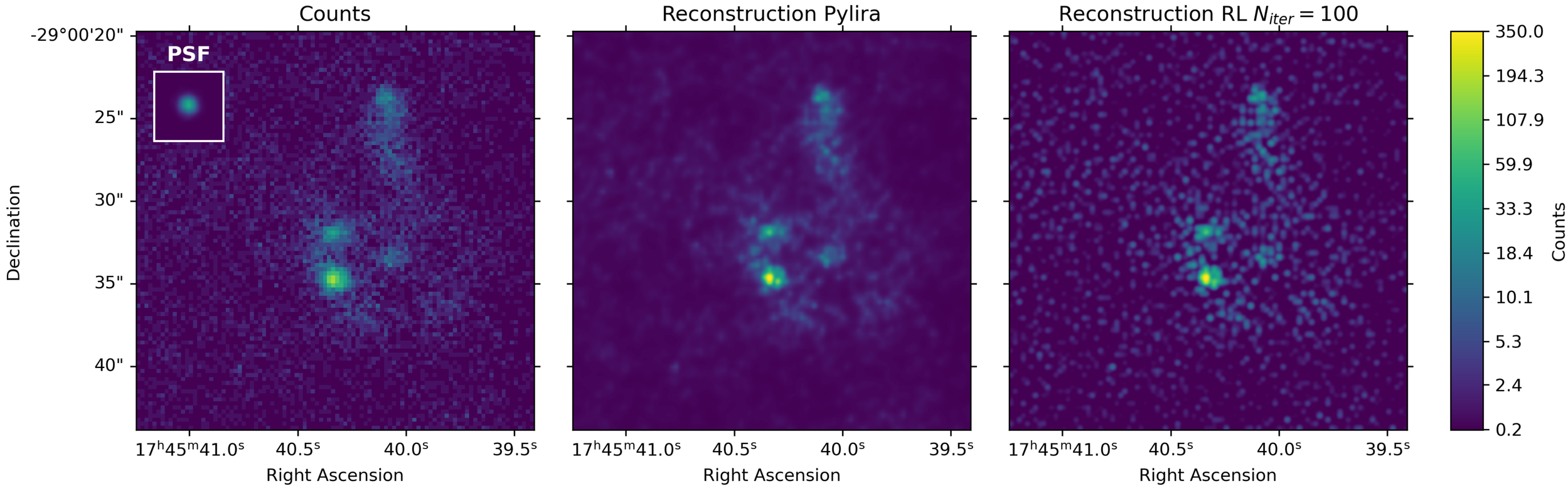
CHANDRA



Crab Nebula seen with CHANDRA…

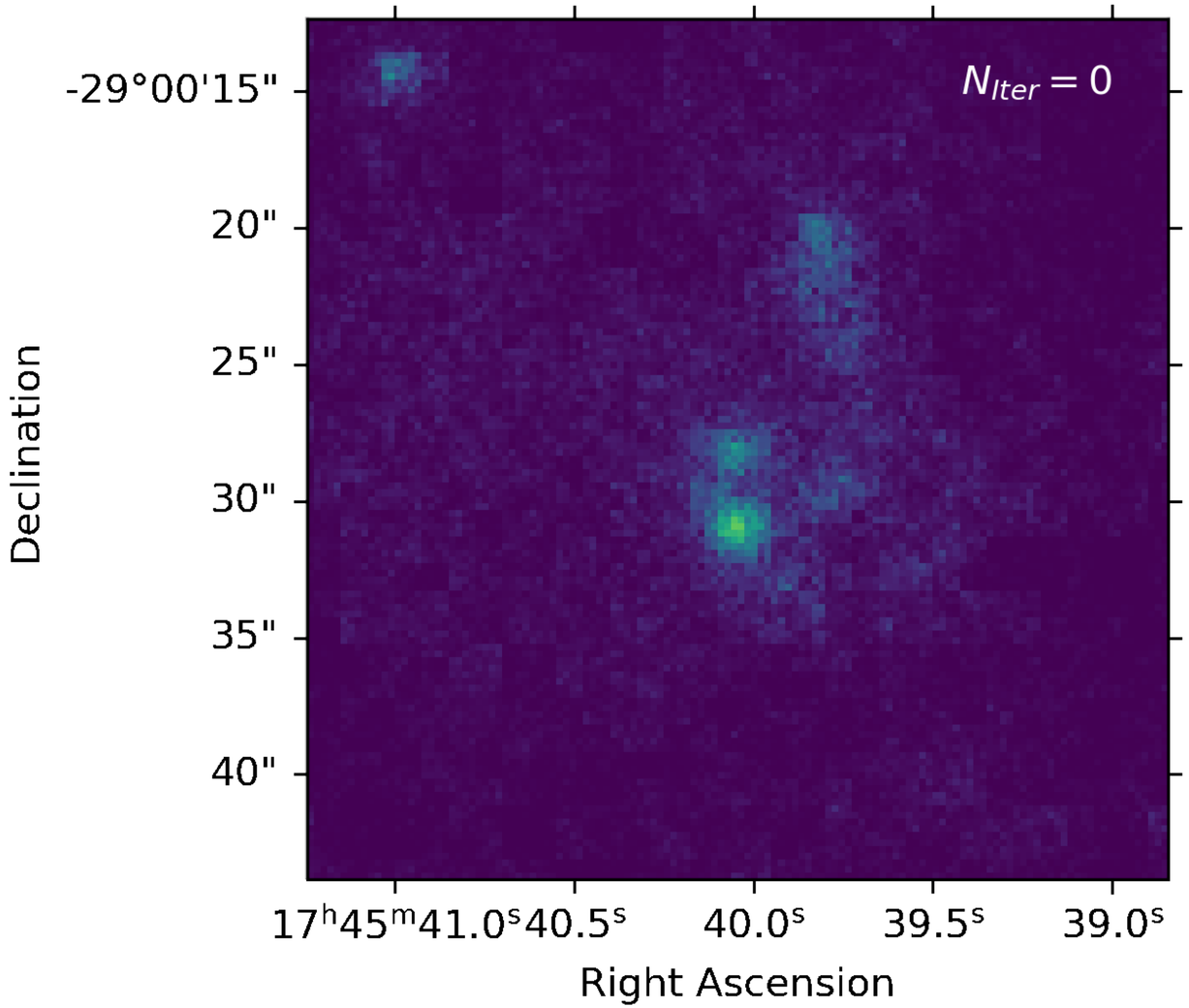
- Chandra is a **space-based X-ray observatory**, which has been in operation since 1999
- It consists of nested cylindrical paraboloid and hyperboloid surfaces, which form an i**maging optical system for X-rays**. In the focal plane, it has multiple instruments for different scientific purposes. This includes a high-resolution camera (HRC) and an Advanced CCD Imaging Spectrometer (ACIS)
- The typical **angular resolution is 0.5 arcsecond** and the covered **energy ranges from 0.1 - 10 keV**.

CHANDRA Galactic Center

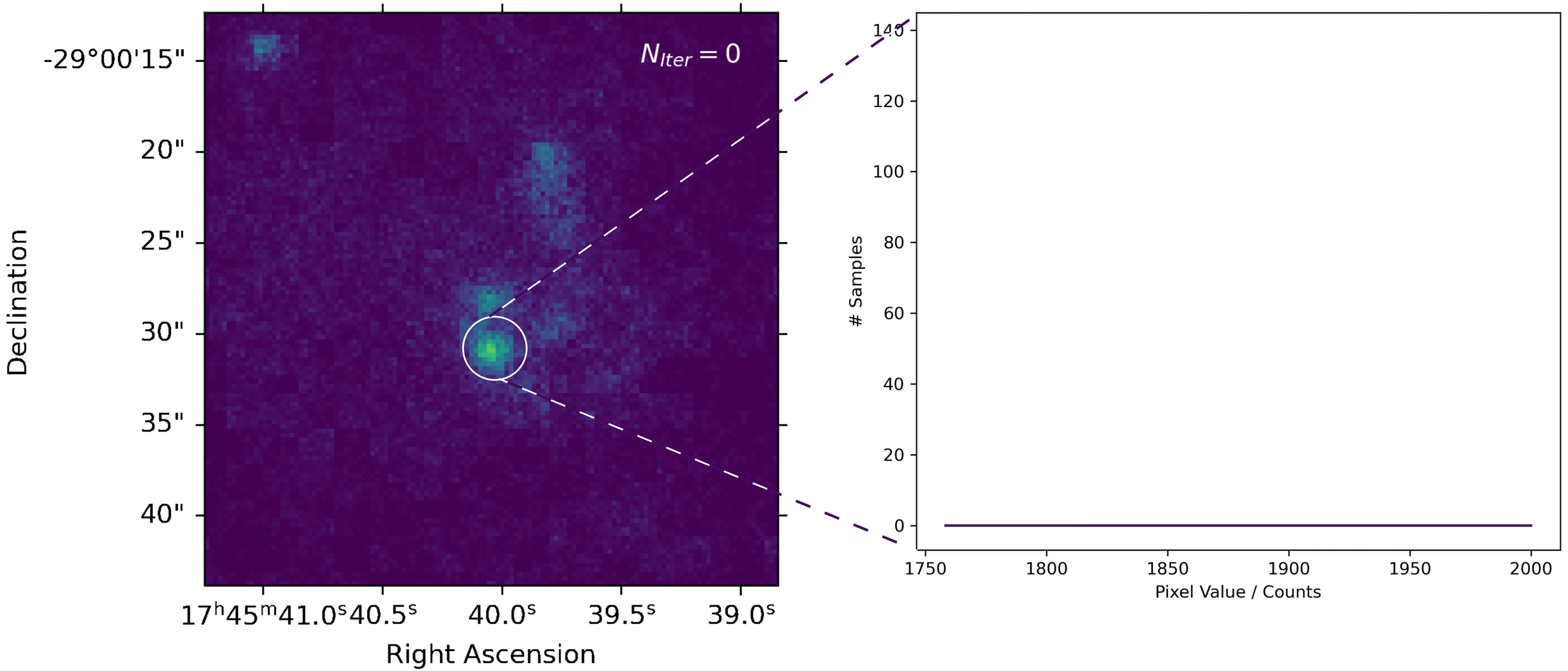


Pylira applied to Chandra ACIS data of the Galactic Center region, using the observation IDs 4684 and 4684.

CHANDRA Galactic Center

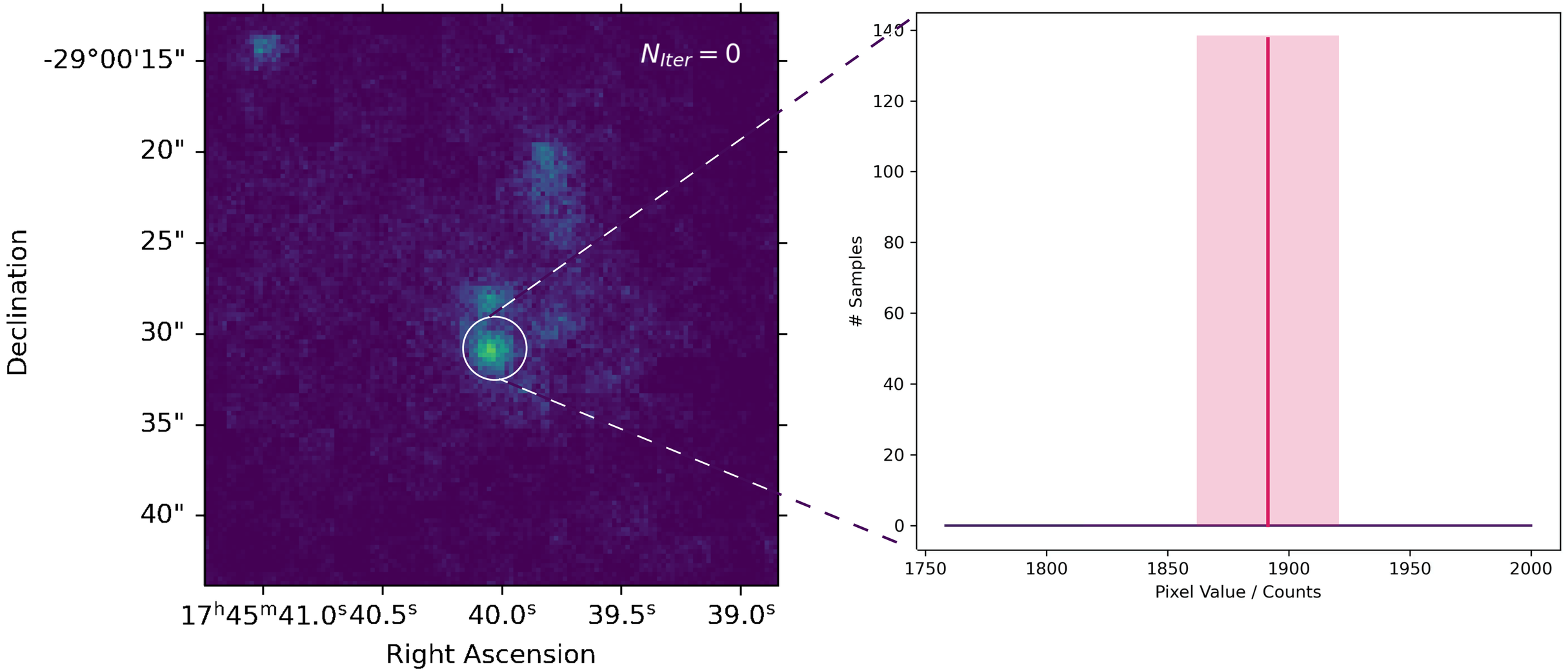


CHANDRA Galactic Center



CHANDRA Galactic Center

Can report mean values and errors…



Summary & Outlook

- Pylira is a **Python package for deconvolution of images** in the presence of Poisson noise. It acts as a wrapper around an original C (RMath) implementation.

- The LIRA method is a nice example for a **“pre ML era” method**, which is statistically fully “traceable”. So it delivers **error estimates and even per-pixel distributions.**

- The current implementation based on the initial C code is very inflexible in terms of prior definitions / modifications. Plans to **extend for more flexible prior definitions**.

- Could also **consider a re-implementation based on more modern technology**, such as [pytorch] / [jax] to take advantage from GPU acceleration

- Possibly combine with ML based methods e.g. by **learning patch based image priors**