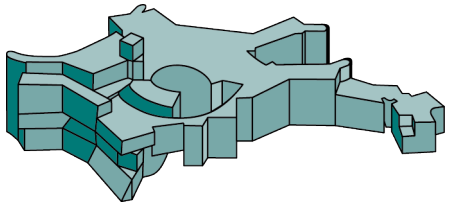
The NIFTy way of Bayesian signal inference

Marco Selig, Michael R. Bell, Henrik Junklewitz, Niels Oppermann,
Martin Reinecke, Maksim Greiner, Carlos Pachajoa, Torsten A. Enßlin

Max Planck Institute for Astrophysics



The NIFTy way of Bayesian signal inference

Marco Selig, Michael R. Bell, Henrik Junklewitz, Niels Oppermann, Martin Reinecke, Maksim Greiner, Carlos Pachajoa, Torsten A. Enßlin

References: arXiv:1301.4499 and arXiv:1210.6866

NIFTy project homepage: http://www.mpa-garching.mpg.de/ift/nifty/

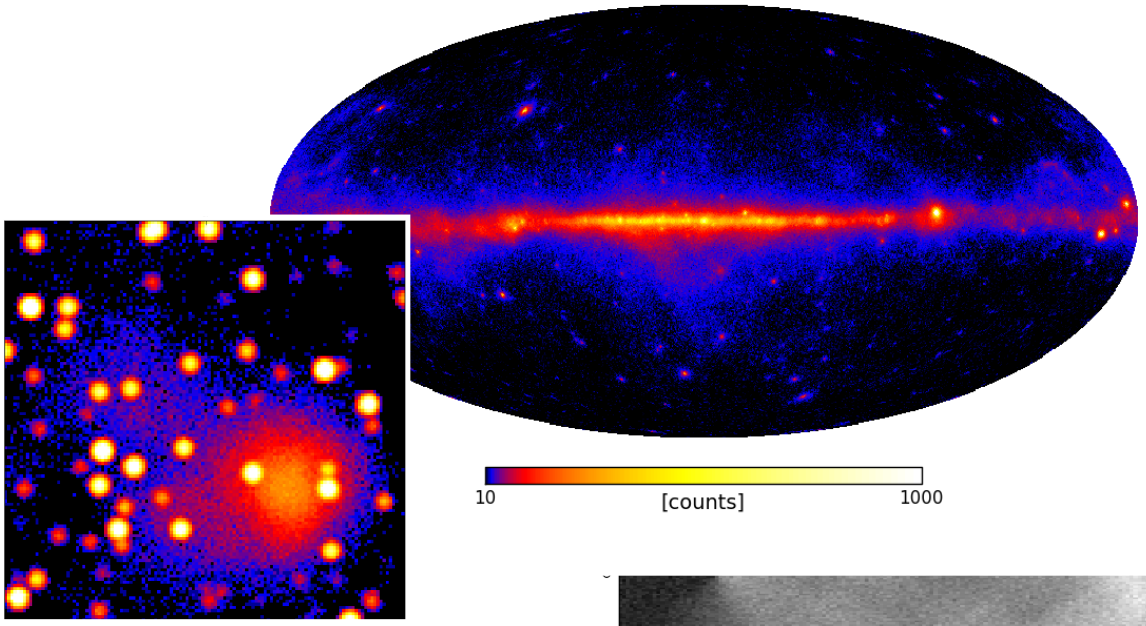
Outline

1. **IFT** – **I**nformation **F**ield **T**heory
2. **NIFTy** – **N**umerical **I**nformation **F**ield **T**heor**y**
3. Applications
4. Summary

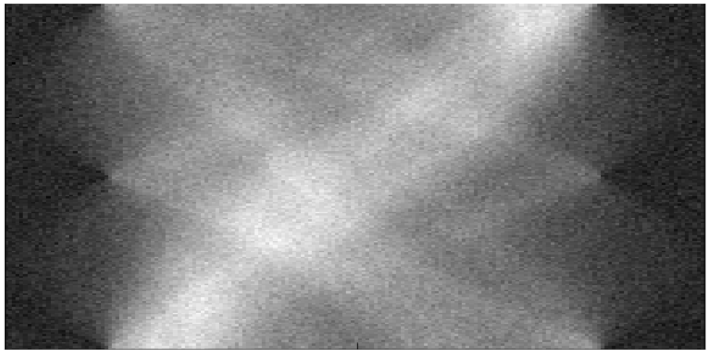
Theory

What's the problem?

- features in the Galactic diffuse γ-ray emission



- separation of diffuse and point-like components



- medical and Galactic tomography

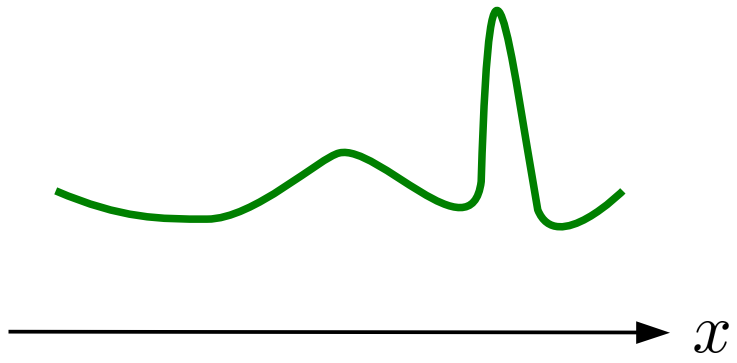
Data ← observation — inference → Signal estimate

- data vector
- finite set of numbers

$$\boldsymbol{d} = (d_1, d_2, d_3, \dots)^\dagger$$

- signal field
- infinite number of degrees of freedom

$$\boldsymbol{s} = s(x) \quad , \quad x \in \Omega$$



Bayes' Theorem ...

$$P(\boldsymbol{s}|\boldsymbol{d}) = \frac{P(\boldsymbol{d}|\boldsymbol{s})P(\boldsymbol{s})}{P(\boldsymbol{d})}$$

Bayes' Theorem ...

$$P(\boldsymbol{s}|\boldsymbol{d}) = \frac{P(\boldsymbol{d}|\boldsymbol{s})P(\boldsymbol{s})}{P(\boldsymbol{d})} = \frac{\exp\left(-H(\boldsymbol{d},\boldsymbol{s})\right)}{Z(\boldsymbol{d})}$$

$$H(\boldsymbol{d},\boldsymbol{s}) = -\log\left(P(\boldsymbol{d},\boldsymbol{s})\right)$$

… Information Field Theory

Wiener filter

- data model
  - linear response
  - additive Gaussian noise

  $$\boldsymbol{d} = \boldsymbol{R}\boldsymbol{s} + \boldsymbol{n}$$

- *a priori* assumptions
  - signal $\boldsymbol{s}$ $\leftarrow$ multidimensional Gaussian prior

- information Hamiltonian

$$\begin{aligned} H(\boldsymbol{d}, \boldsymbol{s}) &= \tfrac{1}{2} \left(\boldsymbol{d} - \boldsymbol{R}\boldsymbol{s}\right)^\dagger \boldsymbol{N}^{-1} \left(\boldsymbol{d} - \boldsymbol{R}\boldsymbol{s}\right) + \tfrac{1}{2} \boldsymbol{s}^\dagger \boldsymbol{S}^{-1} \boldsymbol{s} + [\text{const.}] \\ &= H_0 + \tfrac{1}{2} \boldsymbol{s}^\dagger \left(\boldsymbol{S}^{-1} + \boldsymbol{R}^\dagger \boldsymbol{N}^{-1} \boldsymbol{R}\right) \boldsymbol{s} + \boldsymbol{s}^\dagger \left(\boldsymbol{R}^\dagger \boldsymbol{N}^{-1} \boldsymbol{d}\right) \end{aligned}$$

Wiener filter

- *a posteriori* solution

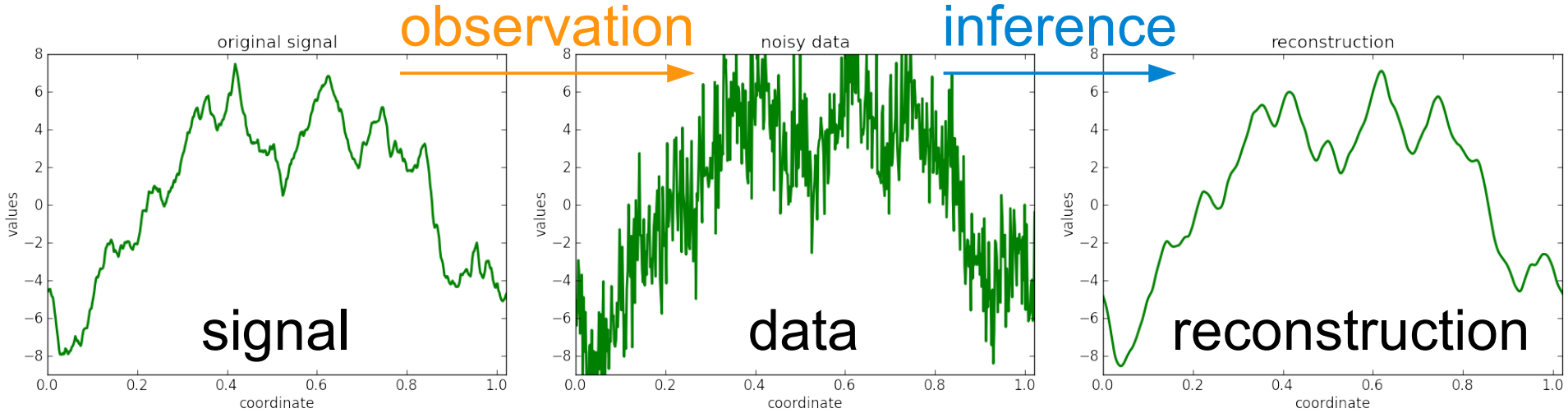
$$\langle \boldsymbol{s} \rangle_{(\boldsymbol{s}|\boldsymbol{d})} = \boldsymbol{m} = \underbrace{\left(\boldsymbol{S}^{-1} + \boldsymbol{R}^\dagger \boldsymbol{N}^{-1} \boldsymbol{R}\right)^{-1}}_{\boldsymbol{D}} \underbrace{\left(\boldsymbol{R}^\dagger \boldsymbol{N}^{-1} \boldsymbol{d}\right)}_{\boldsymbol{j}}$$

$$\left\langle (\boldsymbol{s} - \boldsymbol{m})(\boldsymbol{s} - \boldsymbol{m})^\dagger \right\rangle_{(\boldsymbol{s}|\boldsymbol{d})} = \boldsymbol{D}$$

Wiener filter

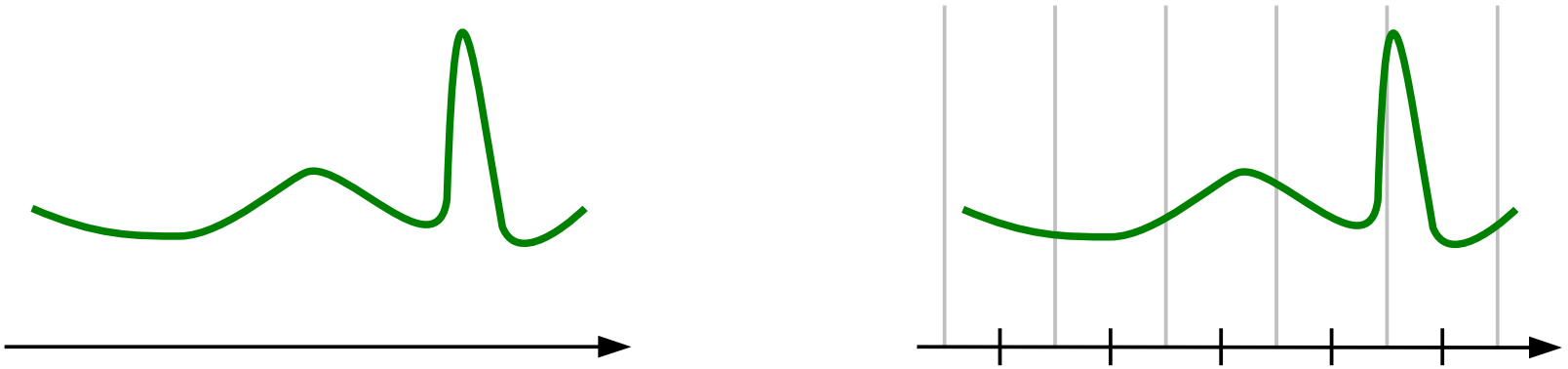
- *a posteriori* solution

$$\langle \boldsymbol{s} \rangle_{(\boldsymbol{s}|\boldsymbol{d})} = \boldsymbol{m} = \underbrace{\left(\boldsymbol{S}^{-1} + \boldsymbol{R}^\dagger \boldsymbol{N}^{-1} \boldsymbol{R}\right)^{-1}}_{\boldsymbol{D}} \underbrace{\left(\boldsymbol{R}^\dagger \boldsymbol{N}^{-1} \boldsymbol{d}\right)}_{\boldsymbol{j}}$$



Discretizing continuous fields

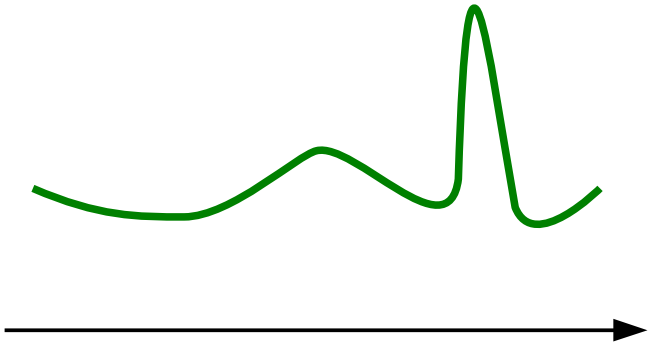
$$\boldsymbol{\varphi} = \varphi(x) \quad , \quad x \in \Omega \qquad\qquad \boldsymbol{\varphi} \mapsto \varphi_q$$

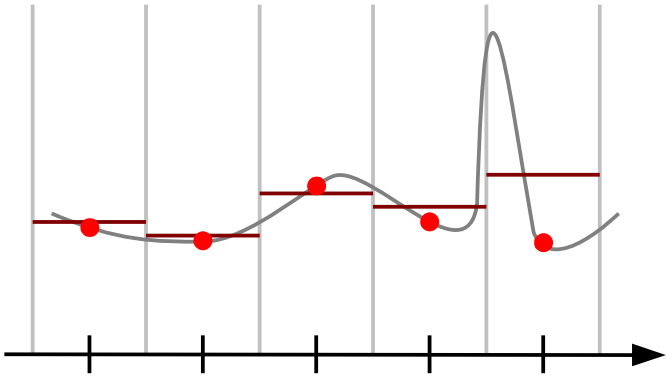


Discretizing continuous fields

$$\boldsymbol{\varphi} = \varphi(x) \quad , \quad x \in \Omega$$

$$\boldsymbol{\varphi} \mapsto \varphi_q \equiv \begin{cases} \left\langle \varphi(x) \right\rangle_{\Omega_q} \\ \varphi\left(\left\langle x \right\rangle_{\Omega_q}\right) \end{cases}$$



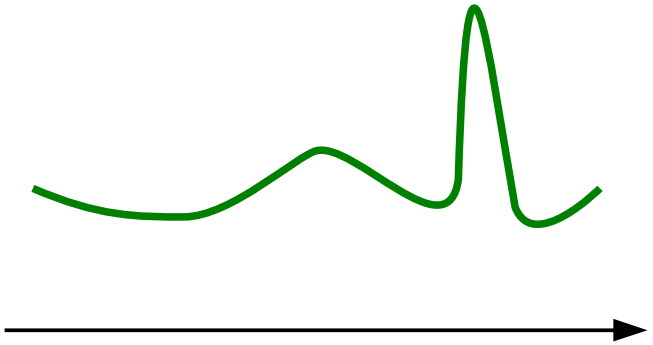


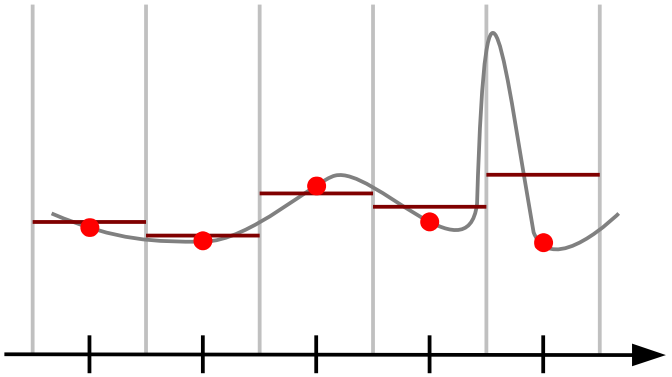
Discretizing continuous fields

$$\boldsymbol{\varphi} = \varphi(x) \quad , \quad x \in \Omega$$

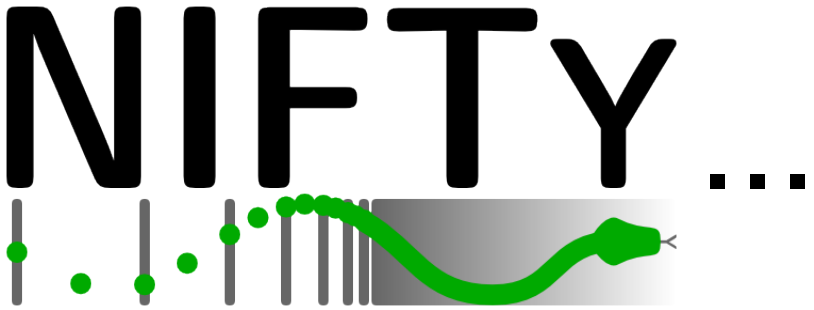
$$\boldsymbol{\varphi} \mapsto \varphi_q \equiv \begin{cases} \left\langle \varphi(x) \right\rangle_{\Omega_q} \\ \varphi\left(\left\langle x \right\rangle_{\Omega_q}\right) \end{cases}$$

$$\boldsymbol{\varphi}^\dagger \boldsymbol{\psi} = \int_\Omega \mathrm{d}x \; \varphi^*(x)\, \psi(x) \approx \sum_q V_q \; \varphi_q^* \psi_q$$



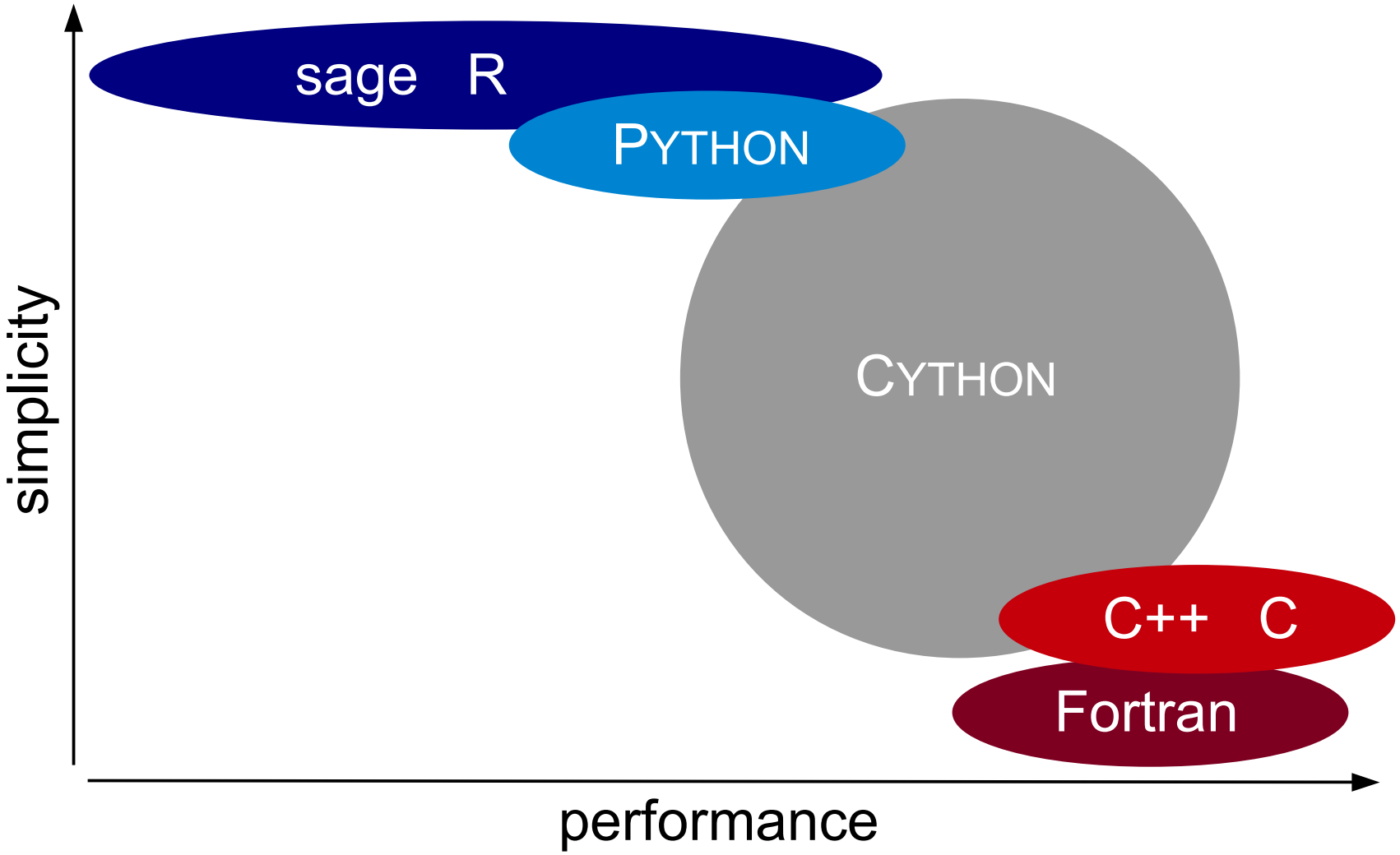



NIFTy



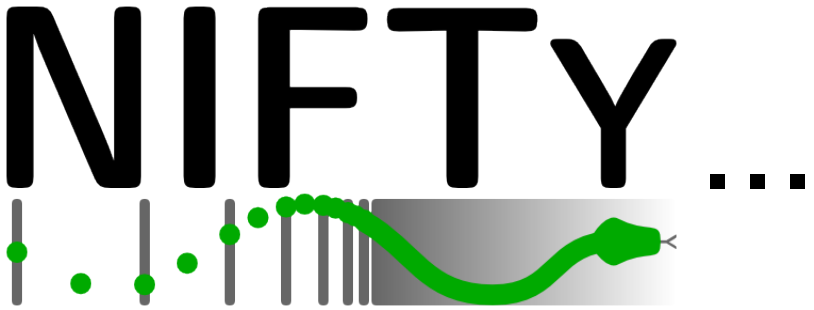
Selig et al. (2013)

- is a versatile PYTHON library incorporating CYTHON, C++, and C libraries


sage R
PYTHON
CYTHON
C++ C
Fortran
simplicity
performance

PYTHON is simple

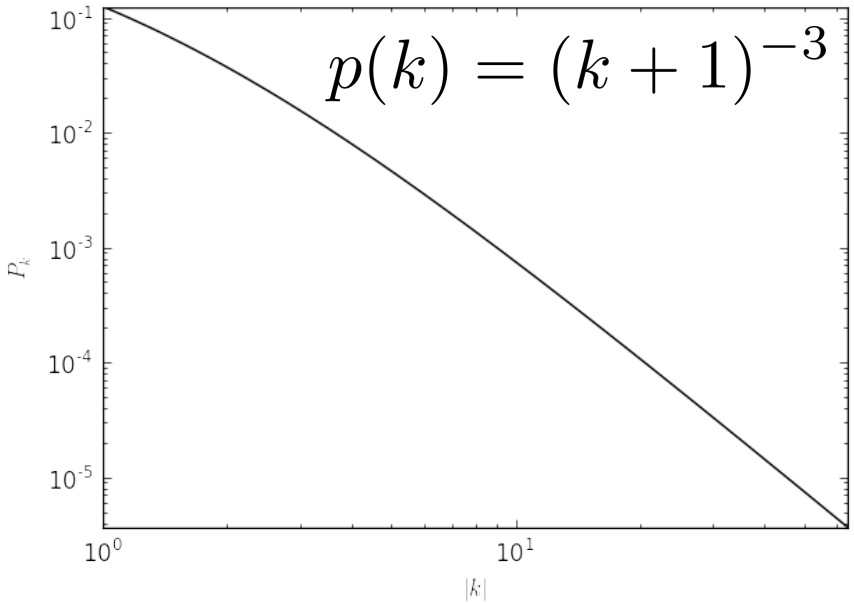
```
In [1]: 1+2
Out[1]: 3

In [2]: func = lambda x: x**2

In [3]: func(3)
Out[3]: 9

In [4]: import numpy

In [5]: func(numpy.array([1, 2, 3]))
Out[5]: array([1, 4, 9])

In [6]:
```



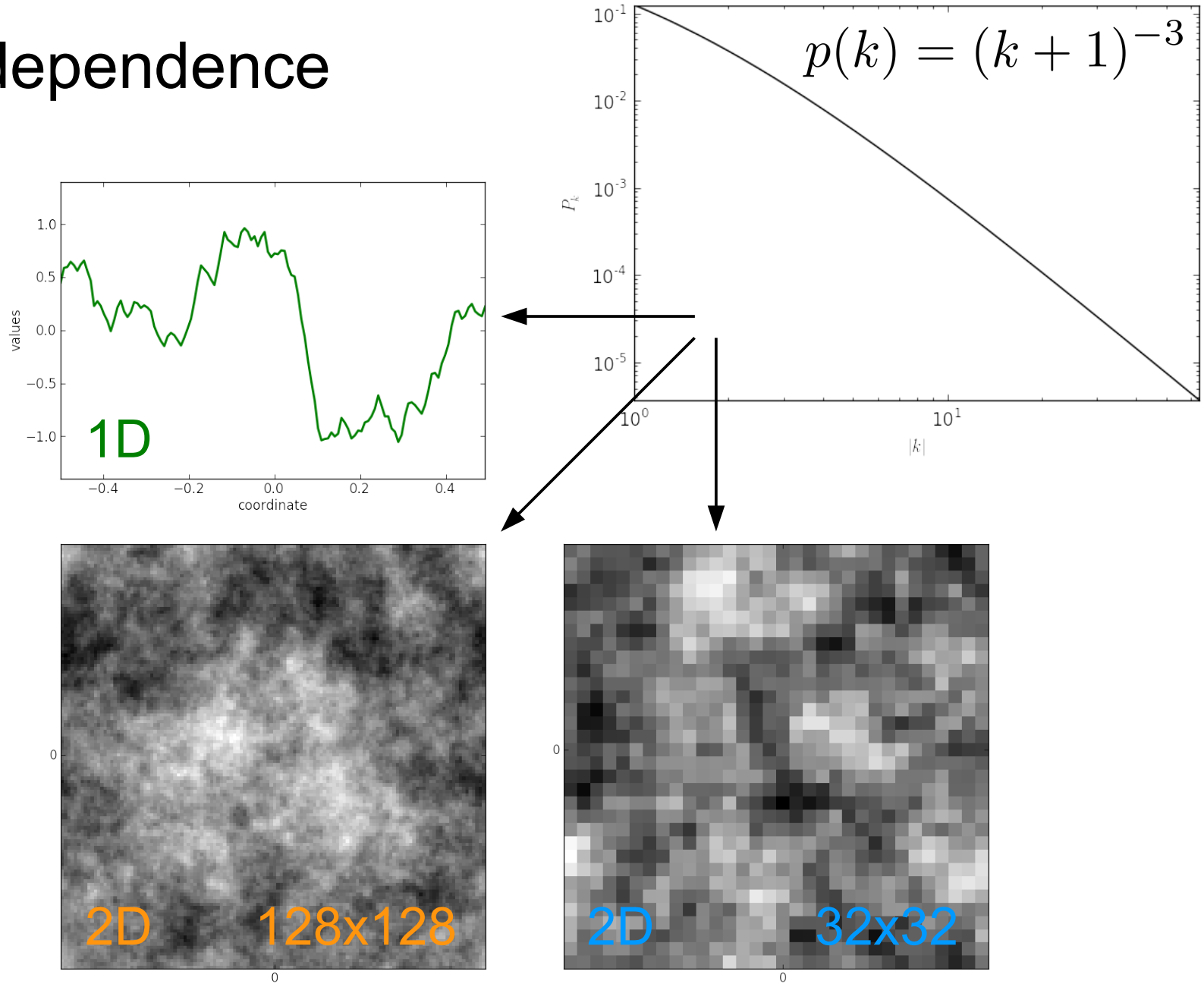
Selig et al. (2013)

- is a versatile PYTHON library incorporating CYTHON, C++, and C libraries
- operates regardless of the underlying spatial grid and its resolution

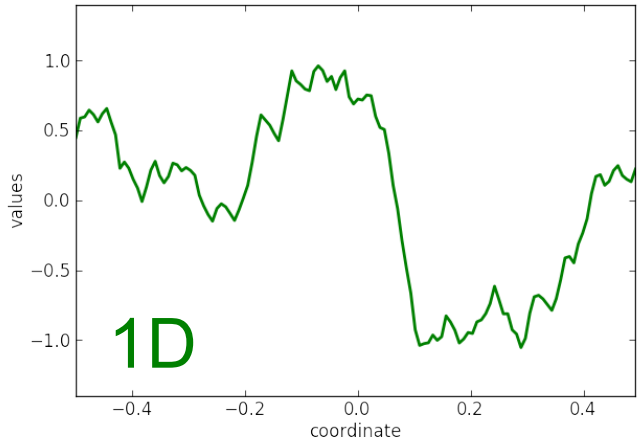
Grid independence

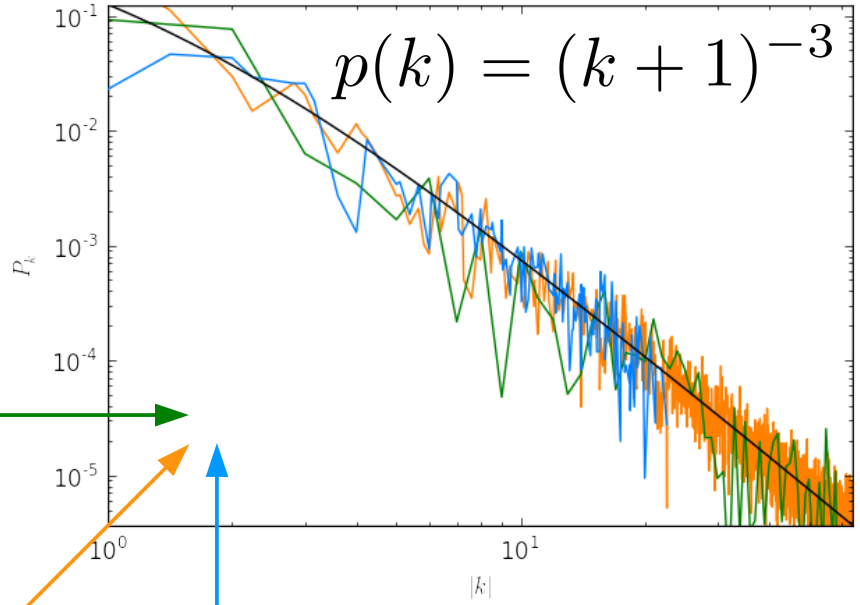


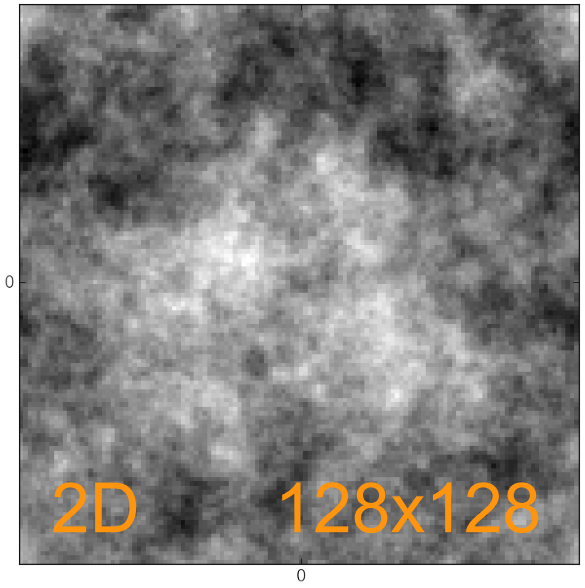
Grid independence

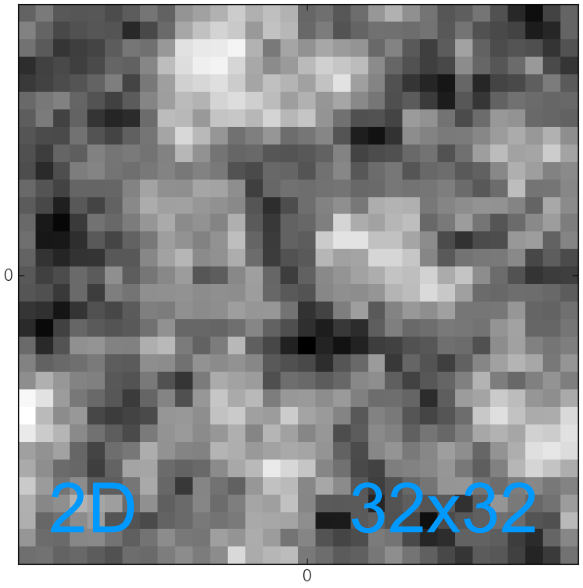


Grid independence









NIFTy ...

Selig et al. (2013)

- is a versatile PYTHON library incorporating CYTHON, C++, and C libraries
- operates regardless of the underlying spatial grid and its resolution
- abstracts spaces, fields, and operators into an object-orientated framework

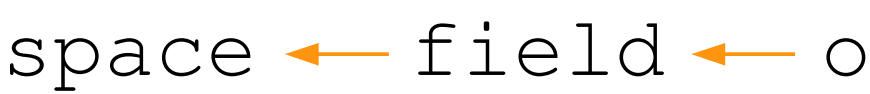
Selig et al. (2013)

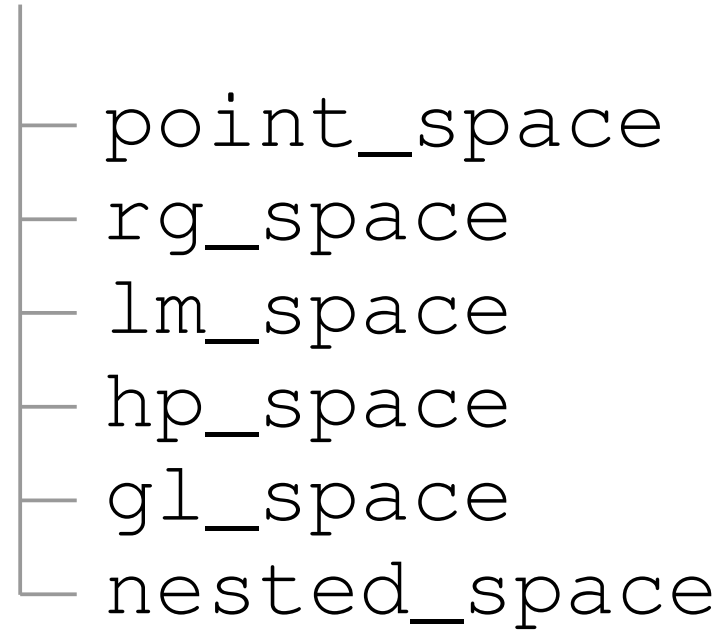
NIFTy classes

- space
  - parameters
- field
  - domain space
  - field values
- operator
  - domain space
  - target space
  - instance methods applying to fields

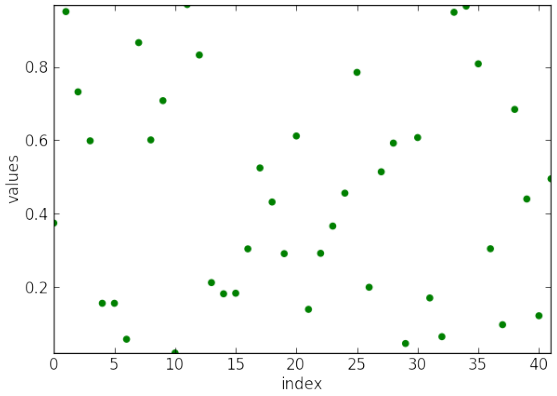
Selig et al. (2013)

NIFTy classes





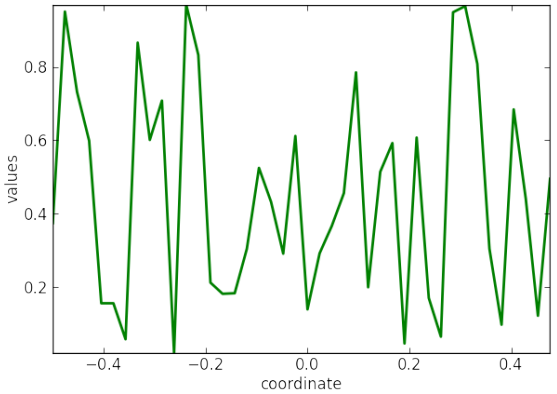
unstructured list of points

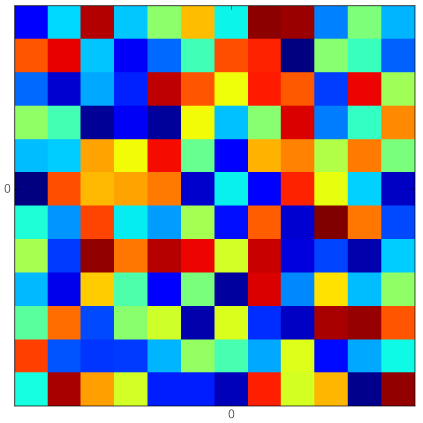


Selig et al. (2013)

NIFTy classes

- point_space — unstructured list of points
- rg_space — n-dimensional regular grid
- lm_space
- hp_space
- gl_space
- nested_space



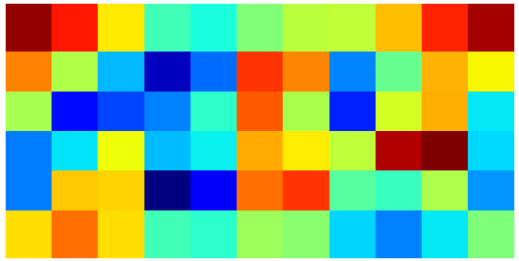


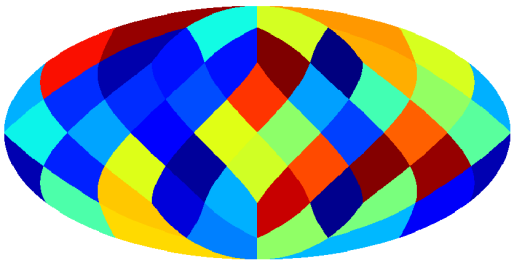
Selig et al. (2013)

NIFTy classes

```
object
space ← field ← operator
```

- point_space — unstructured list of points
- rg_space — n-dimensional regular grid
- lm_space — spherical harmonics
- hp_space — Gauss-Legendre grid on the sphere
- gl_space — HEALPix grid on the sphere
- nested_space

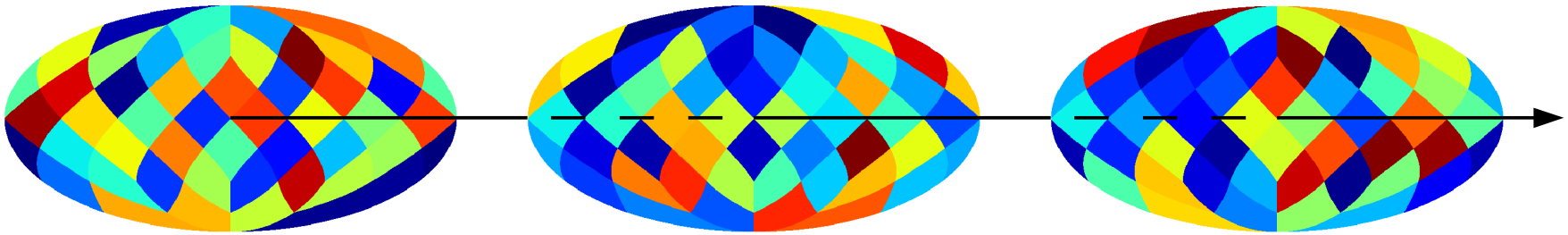




Selig et al. (2013)

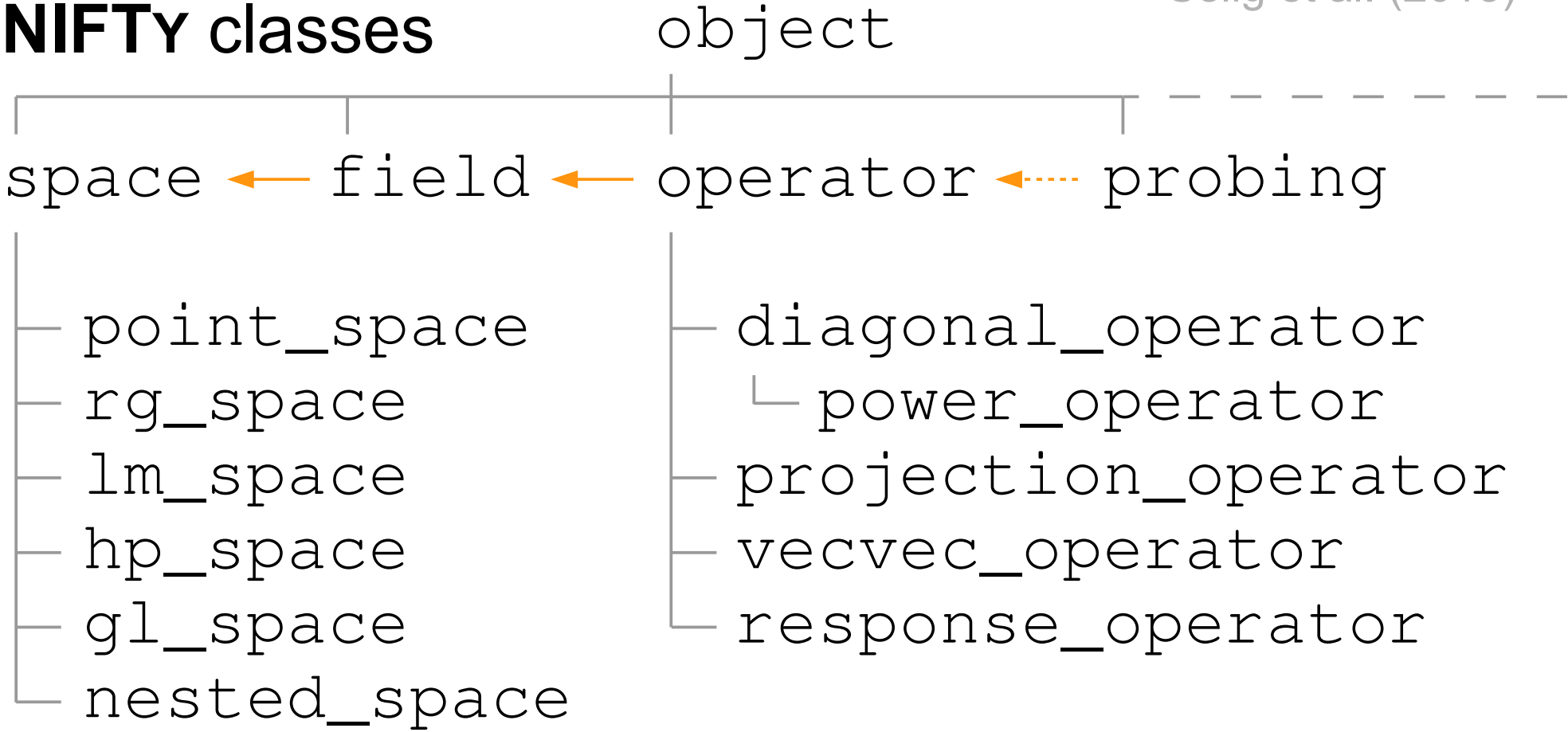
NIFTy classes

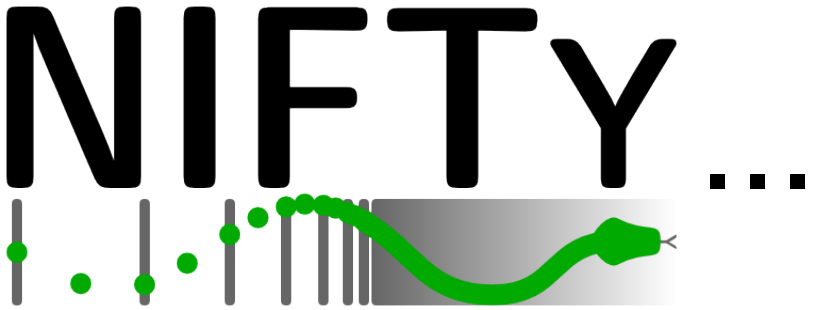
- `point_space` — unstructured list of points
- `rg_space` — n-dimensional regular grid
- `lm_space` — spherical harmonics
- `hp_space` — Gauss-Legendre grid on the sphere
- `gl_space` — HEALPix grid on the sphere
- `nested_space` — (arbitrary product of grids)



Selig et al. (2013)

NIFTy classes





Selig et al. (2013)

- is a versatile PYTHON library incorporating CYTHON, C++, and C libraries
- operates regardless of the underlying spatial grid and its resolution
- abstracts spaces, fields, and operators into an object-orientated framework
- allows the user the abstract formulation and programming of signal inference algorithms

Selig et al. (2013)

Wiener filtering

```
from nifty import *
from scipy.sparse.linalg import LinearOperator as lo
from scipy.sparse.linalg import cg

class propagator(operator):                                          # define propagator class

    _matvec = (lambda self, x: self.inverse_times(x).val.flatten())

    def _multiply(self, x):
        # some numerical invertion technique; here, conjugate gradient
        A = lo(shape=tuple(self.dim()), matvec=self._matvec)
        b = x.val.flatten()
        x_, info = cg(A, b, M=None)
        return x_

    def _inverse_multiply(self, x):
        S, N, R = self.para
        return S.inverse_times(x) + R.adjoint_times(N.inverse_times(R.times(x)))

# some signal space; e.g., a one-dimensional regular grid
s_space = rg_space(512, zerocenter=False, dist=0.002)                # define signal space
# or      rg_space([256, 256])
# or      hp_space(128)
k_space = s_space.get_codomain()                                     # get conjugate space
kindex, rho = k_space.get_power_index(irreducible=True)
# some power spectrum
power = [42 / (kk + 1) ** 3 for kk in kindex]
S = power_operator(k_space, spec=power)                              # define signal covariance
s = S.get_random_field(domain=s_space)                               # generate signal
R = response_operator(s_space, sigma=0.0, mask=1.0, assign=None)     # define response
d_space = R.target                                                   # get data space
# some noise variance; e.g., 1
N = diagonal_operator(d_space, diag=1, bare=True)                    # define noise covariance
n = N.get_random_field(domain=d_space)                               # generate noise
d = R(s) + n                                                         # compute data
j = R.adjoint_times(N.inverse_times(d))                              # define source
D = propagator(s_space, sym=True, imp=True, para=[S,N,R])            # define propagator
m = D(j)                                                             # reconstruct map
s.plot(title="signal")                                               # plot signal
d.cast_domain(s_space)
d.plot(title="data", vmin=s.val.min(), vmax=s.val.max())             # plot data
m.plot(title="reconstructed map", vmin=s.val.min(), vmax=s.val.max()) # plot map
```

Selig et al. (2013)

Wiener filtering

$$\boldsymbol{d} = \boldsymbol{R}\boldsymbol{s} + \boldsymbol{n}$$

$$\boldsymbol{m} = \underbrace{\left(\boldsymbol{S}^{-1} + \boldsymbol{R}^\dagger \boldsymbol{N}^{-1} \boldsymbol{R}\right)^{-1}}_{\boldsymbol{D}} \underbrace{\left(\boldsymbol{R}^\dagger \boldsymbol{N}^{-1} \boldsymbol{d}\right)}_{\boldsymbol{j}}$$

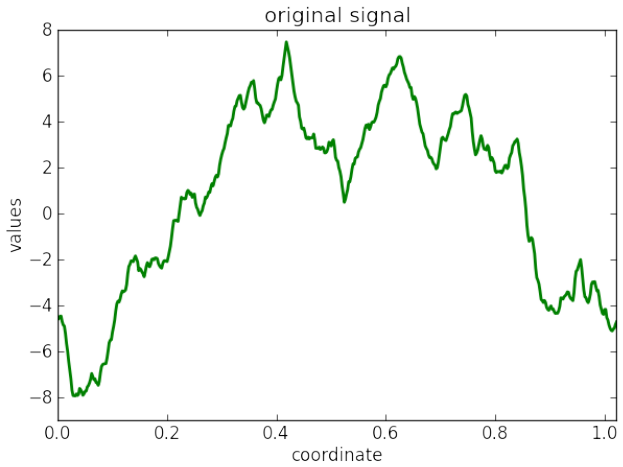
```
# some signal space; e.g., a one-dimensional regular grid
s_space = rg_space(512, zerocenter=False, dist=0.002)                # define signal space
# or      rg_space([256, 256])
# or      hp_space(128)
k_space = s_space.get_codomain()                                     # get conjugate space
kindex, rho = k_space.get_power_index(irreducible=True)
# some power spectrum
power = [42 / (kk + 1) ** 3 for kk in kindex]
S = power_operator(k_space, spec=power)                              # define signal covariance
s = S.get_random_field(domain=s_space)                               # generate signal
R = response_operator(s_space, sigma=0.0, mask=1.0, assign=None)     # define response
d_space = R.targ
# some noise var
N = diagonal_ope
n = N.get_random
d = R(s) + n
j = R.adjoint_ti
D = propagator(s_space, sym=True, imp=True, para=[S,N,R])            # define propagator
m = D(j)
s.plot(title="si
d.cast_domain(s_
d.plot(title="da
m.plot(title="reconstructed map", vmin=s.val.min(), vmax=s.val.max()) # plot map
```

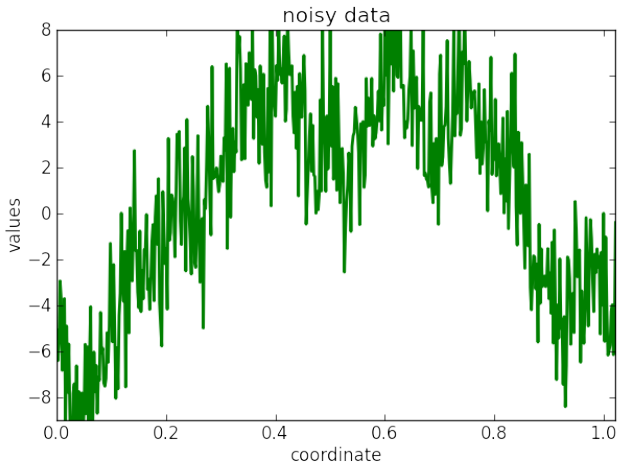
```
d = R(s) + n
j = R.adjoint_times(N.inverse_times(d))
```

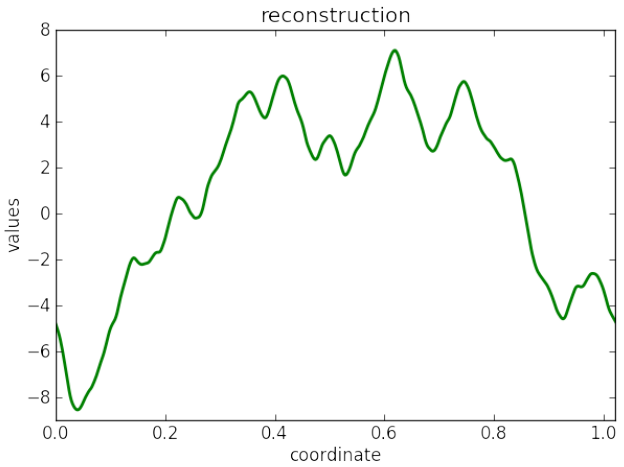
```
m = D(j)
```

Selig et al. (2013)

Wiener filtering





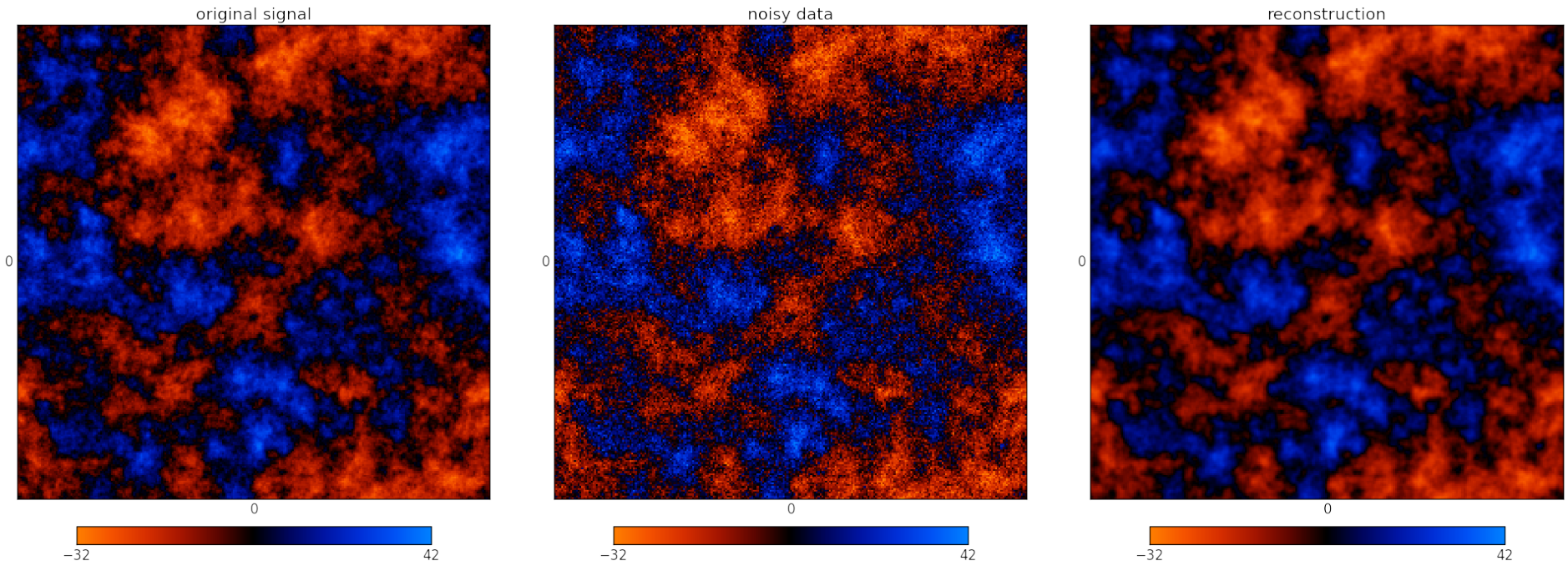



```
# some signal space: e.g. a one-dimensional regular grid
s_space = rg_spa
# or      rg_spa
# or      hp_spa
k_space = s_space.get_codomain()                                  # get conjugate space
kindex, rho = k_space.get_power_index(irreducible=True)
# some power spectrum
power = [42 / (kk + 1) ** 3 for kk in kindex]
S = power_operator(k_space, spec=power)                           # define signal covariance
s = S.get_random_field(domain=s_space)                            # generate signal
R = response_operator(s_space, sigma=0.0, mask=1.0, assign=None)  # define response
d_space = R.targ
# some noise var
N = diagonal_ope
n = N.get_random
d = R(s) + n
j = R.adjoint_ti
D = propagator(s_space, sym=True, imp=True, para=[3,N,R])         # define propagator
m = D(j)
s.plot(title="si
d.cast_domain(s_
d.plot(title="da
m.plot(title="reconstructed map", vmin=s.val.min(), vmax=s.val.max()) # plot map
```

```
s_space = rg_space(512, ...)
```

```
d = R(s) + n
j = R.adjoint_times(N.inverse_times(d))
```

```
m = D(j)
```




```
# some signal space: e.g. a one-dimensional regular grid
s_space = rg_spa
# or      rg_spa
# or      hp_spa
k_space = s_space.get_codomain()                                  # get conjugate space
kindex, rho = k_space.get_power_index(irreducible=True)
# some power spectrum
power = [42 / (kk + 1) ** 3 for kk in kindex]
S = power_operator(k_space, spec=power)                           # define signal covariance
s = S.get_random_field(domain=s_space)                            # generate signal
R = response_operator(s_space, sigma=0.0, mask=1.0, assign=None)  # define response
d_space = R.targ
# some noise var
N = diagonal_ope
n = N.get_random
d = R(s) + n
j = R.adjoint_ti
D = propagator(s_space, sym=True, imp=True, para=[3,N,R])         # define propagator
m = D(j)
s.plot(title="si
d.cast_domain(s_
d.plot(title="da
m.plot(title="reconstructed map", vmin=s.val.min(), vmax=s.val.max()) # plot map
```

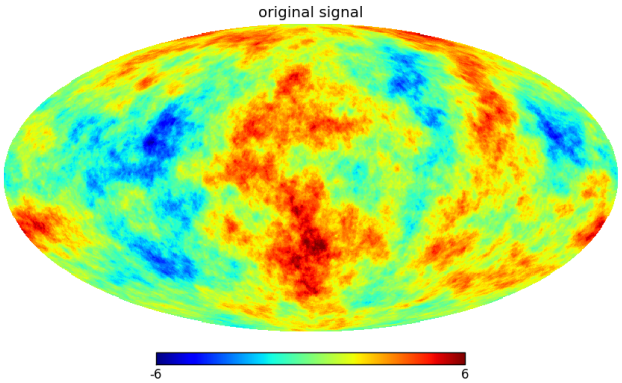
```
s_space = rg_space([256, 256])
```

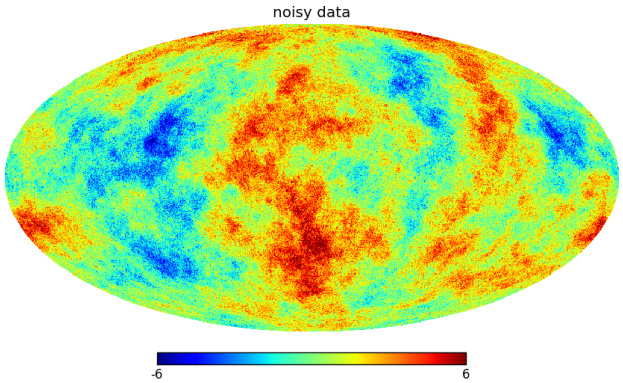
```
d = R(s) + n
j = R.adjoint_times(N.inverse_times(d))
```

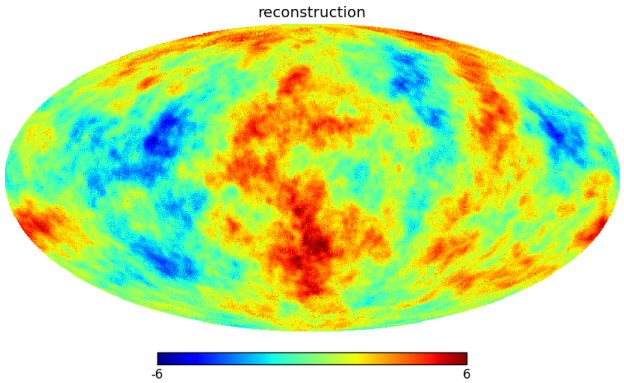
```
m = D(j)
```

Wiener filtering

Selig et al. (2013)








```
# some signal space: e.g. a one-dimensional regular grid
s_space = rg_spa
# or      rg_spa
# or      hp_spa
k_space = s_space.get_codomain()                                  # get conjugate space
kindex, rho = k_space.get_power_index(irreducible=True)
# some power spectrum
power = [42 / (kk + 1) ** 3 for kk in kindex]
S = power_operator(k_space, spec=power)                           # define signal covariance
s = S.get_random_field(domain=s_space)                            # generate signal
R = response_operator(s_space, sigma=0.0, mask=1.0, assign=None)  # define response
d_space = R.targ
# some noise var
N = diagonal_ope
n = N.get_random
d = R(s) + n
j = R.adjoint_ti
D = propagator(s_space, sym=True, imp=True, para=[3,N,R])         # define propagator
m = D(j)
s.plot(title="si
d.cast_domain(s_
d.plot(title="da
m.plot(title="reconstructed map", vmin=s.val.min(), vmax=s.val.max()) # plot map
```

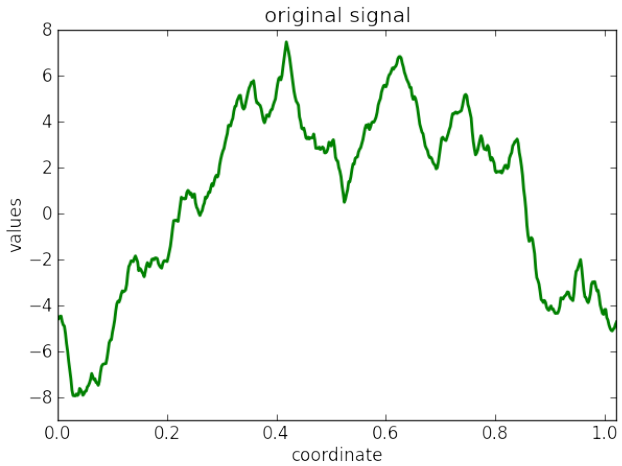
```
s_space = hp_space(128)
```

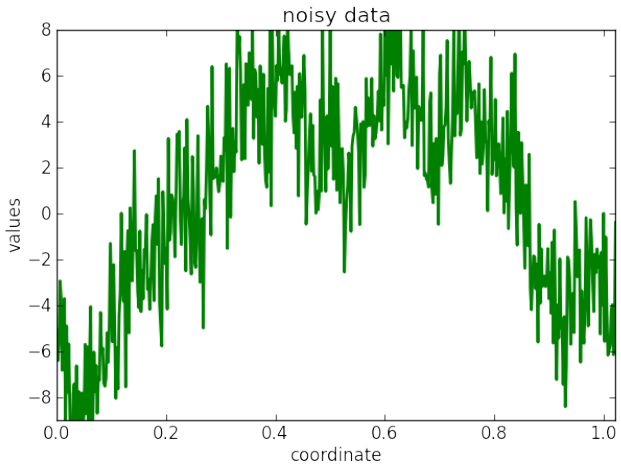
```
d = R(s) + n
j = R.adjoint_times(N.inverse_times(d))
```

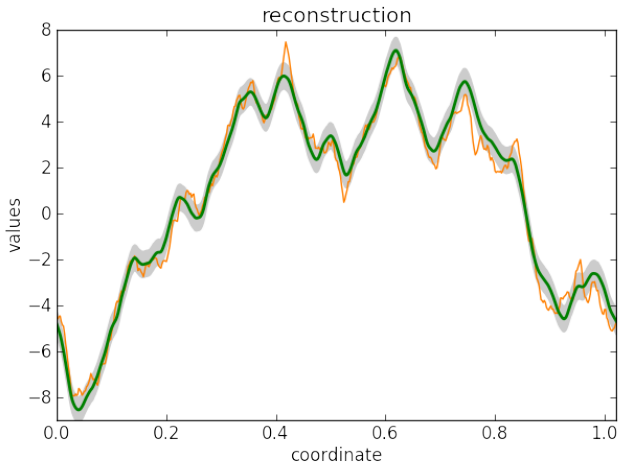
```
m = D(j)
```

Selig et al. (2013)

Wiener filtering and more

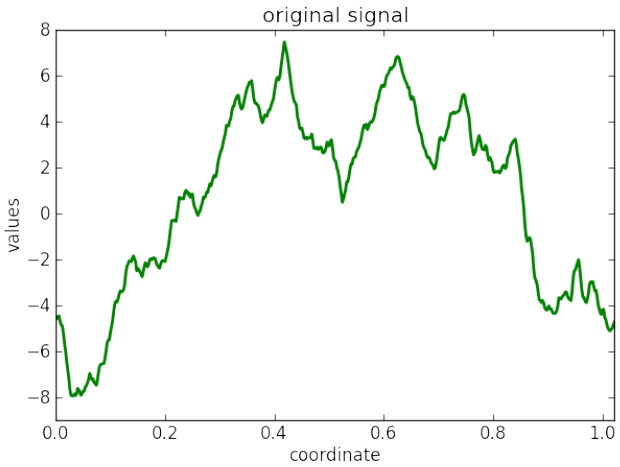


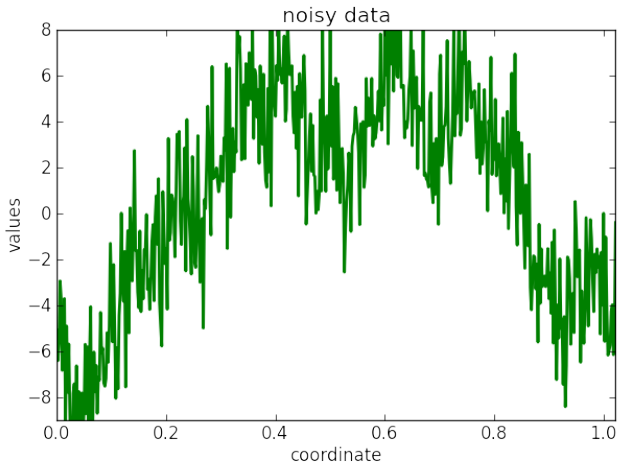


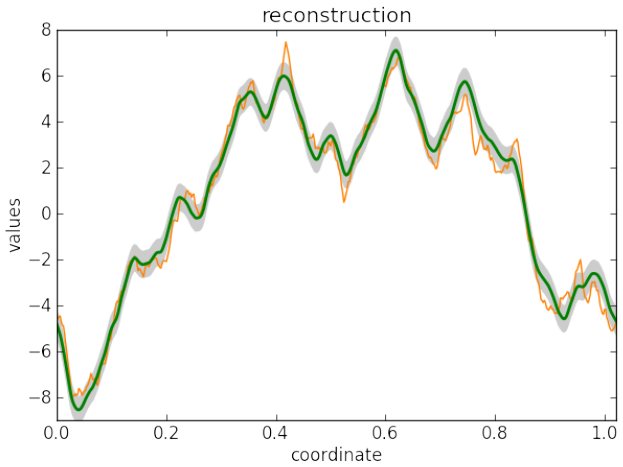


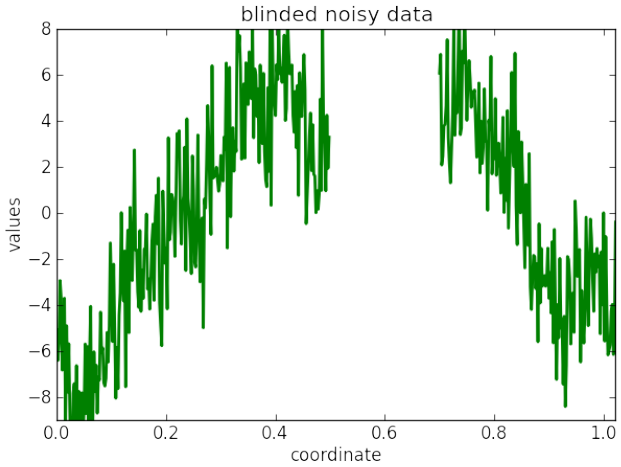
Selig et al. (2013)

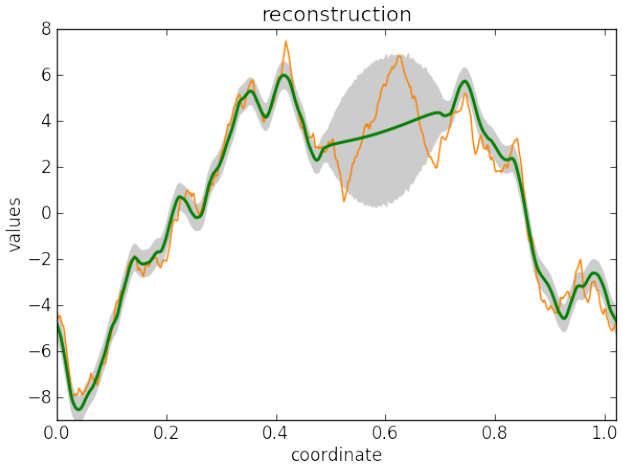
Wiener filtering and more



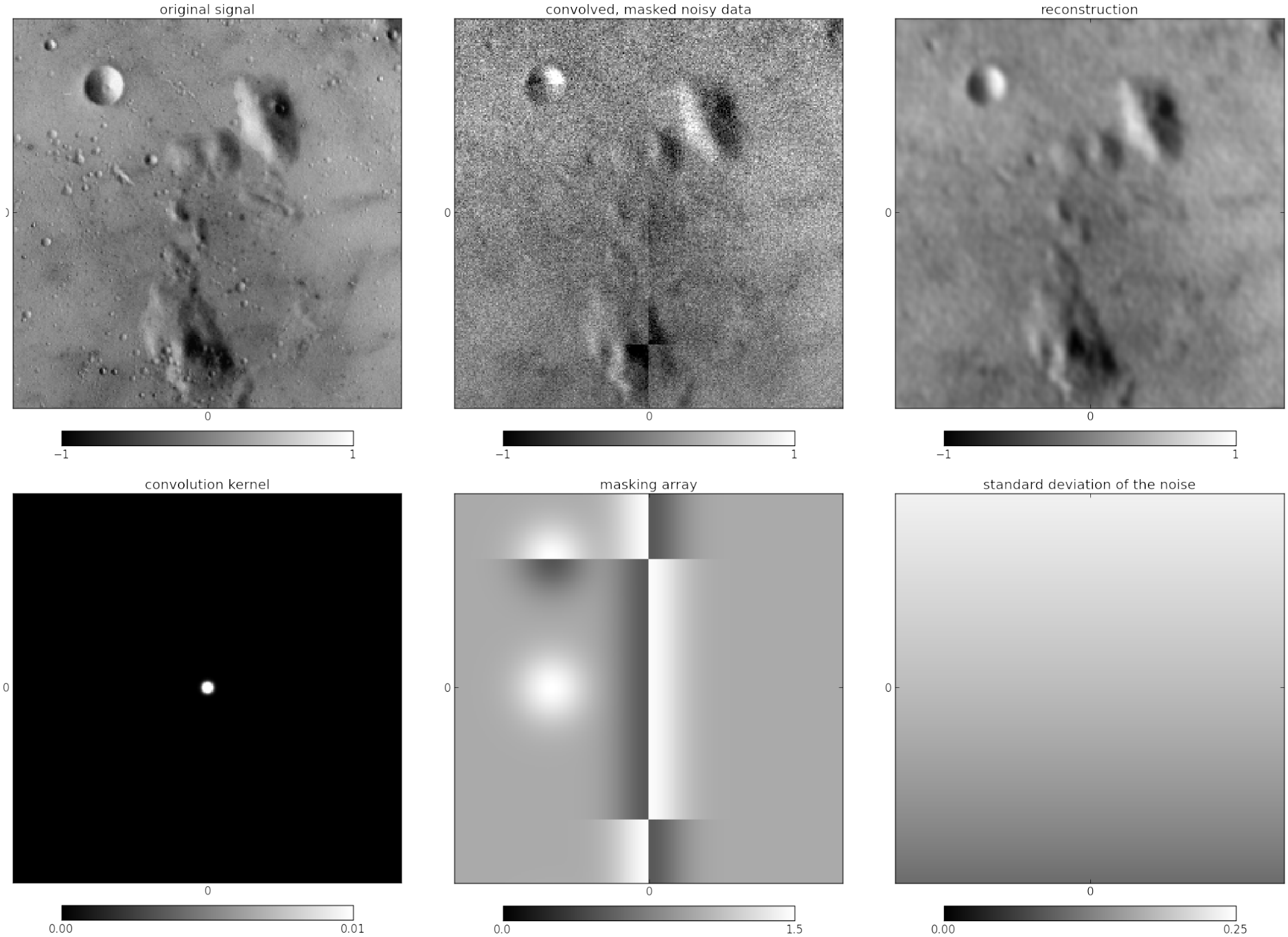


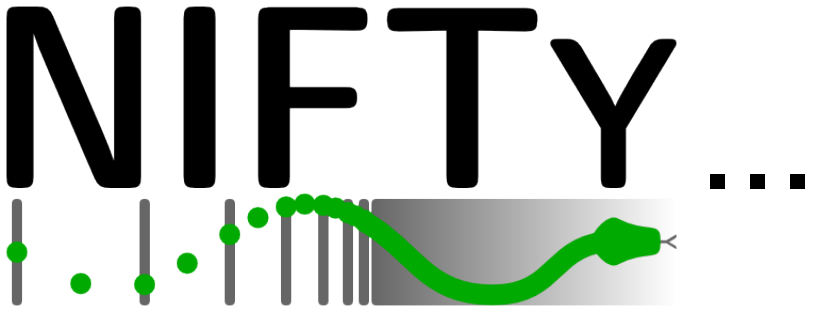






source: USC-SIPI

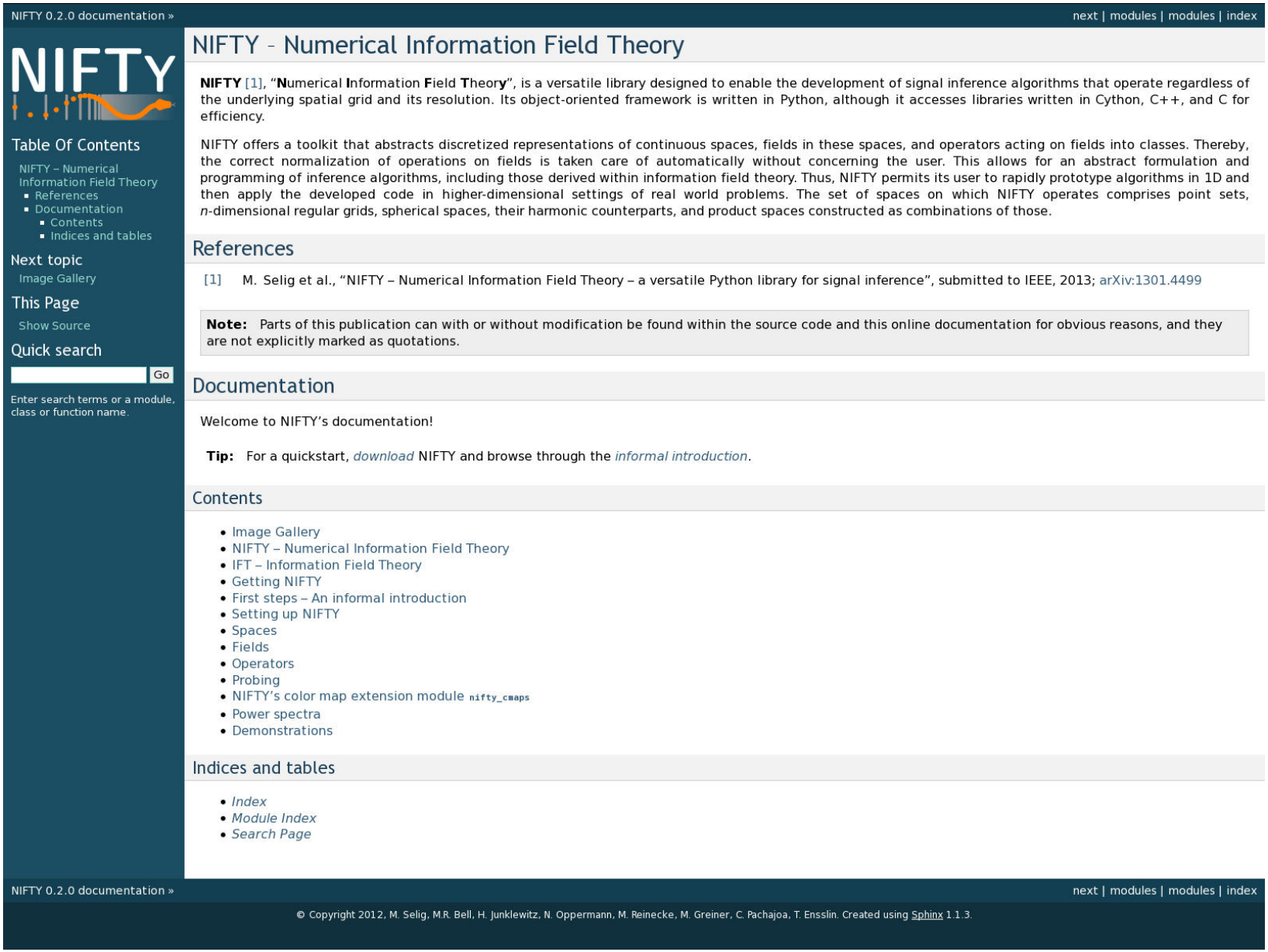




Selig et al. (2013)

- is a versatile PYTHON library incorporating CYTHON, C++, and C libraries
- operates regardless of the underlying spatial grid and its resolution
- abstracts spaces, fields, and operators into an object-orientated framework
- allows the user the abstract formulation and programming of signal inference algorithms
- provides an extensive online documentation

NIFTy project homepage: http://www.mpa-garching.mpg.de/ift/nifty/



NIFTY 0.2.0 documentation » next | modules | modules | index

NIFTy

Table Of Contents

- NIFTY – Numerical Information Field Theory
  - References
  - Documentation
    - Contents
    - Indices and tables

Next topic

Image Gallery

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

NIFTY – Numerical Information Field Theory

NIFTY [1], "**N**umerical **I**nformation **F**ield **T**heor**y**", is a versatile library designed to enable the development of signal inference algorithms that operate regardless of the underlying spatial grid and its resolution. Its object-oriented framework is written in Python, although it accesses libraries written in Cython, C++, and C for efficiency.

NIFTY offers a toolkit that abstracts discretized representations of continuous spaces, fields in these spaces, and operators acting on fields into classes. Thereby, the correct normalization of operations on fields is taken care of automatically without concerning the user. This allows for an abstract formulation and programming of inference algorithms, including those derived within information field theory. Thus, NIFTY permits its user to rapidly prototype algorithms in 1D and then apply the developed code in higher-dimensional settings of real world problems. The set of spaces on which NIFTY operates comprises point sets, *n*-dimensional regular grids, spherical spaces, their harmonic counterparts, and product spaces constructed as combinations of those.

References

[1] M. Selig et al., "NIFTY – Numerical Information Field Theory – a versatile Python library for signal inference", submitted to IEEE, 2013; arXiv:1301.4499

Note: Parts of this publication can with or without modification be found within the source code and this online documentation for obvious reasons, and they are not explicitly marked as quotations.

Documentation

Welcome to NIFTY's documentation!

Tip: For a quickstart, *download* NIFTY and browse through the *informal introduction*.

Contents

- Image Gallery
- NIFTY – Numerical Information Field Theory
- IFT – Information Field Theory
- Getting NIFTY
- First steps – An informal introduction
- Setting up NIFTY
- Spaces
- Fields
- Operators
- Probing
- NIFTY's color map extension module `nifty_cmaps`
- Power spectra
- Demonstrations

Indices and tables

- *Index*
- *Module Index*
- *Search Page*

NIFTY 0.2.0 documentation » next | modules | modules | index

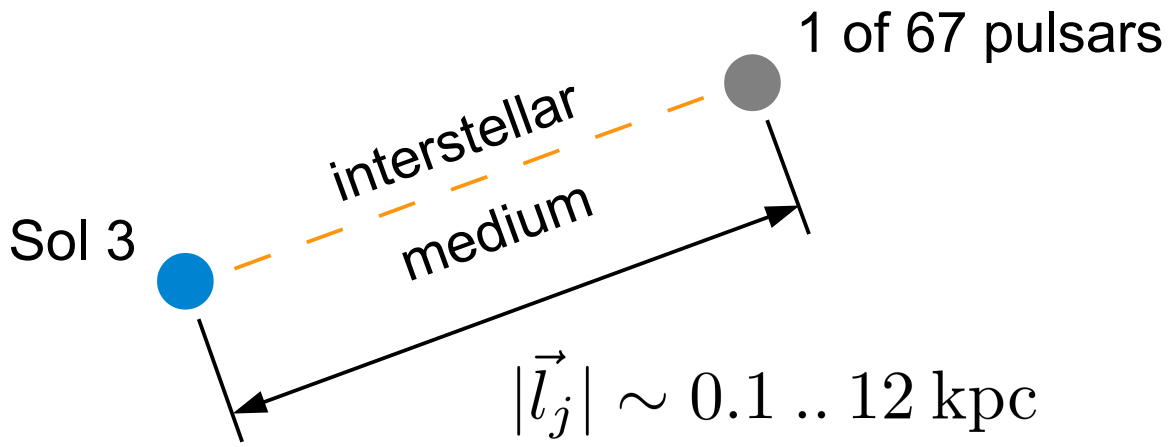
Applications

The Galactic free electron density

Greiner et al. (in prep.)

- data model
  - dispersion measures from different lines of sight
  - additive Gaussian noise

$$\langle \boldsymbol{d} \rangle_{(\boldsymbol{d}|\boldsymbol{\rho})} = \left(\int_{\text{observer}}^{\text{pulsar}} \mathrm{d}\vec{l}_j \; \rho(\vec{x}) \right)_j = \boldsymbol{R}\boldsymbol{\rho}$$



The Galactic free electron density

Greiner et al. (in prep.)

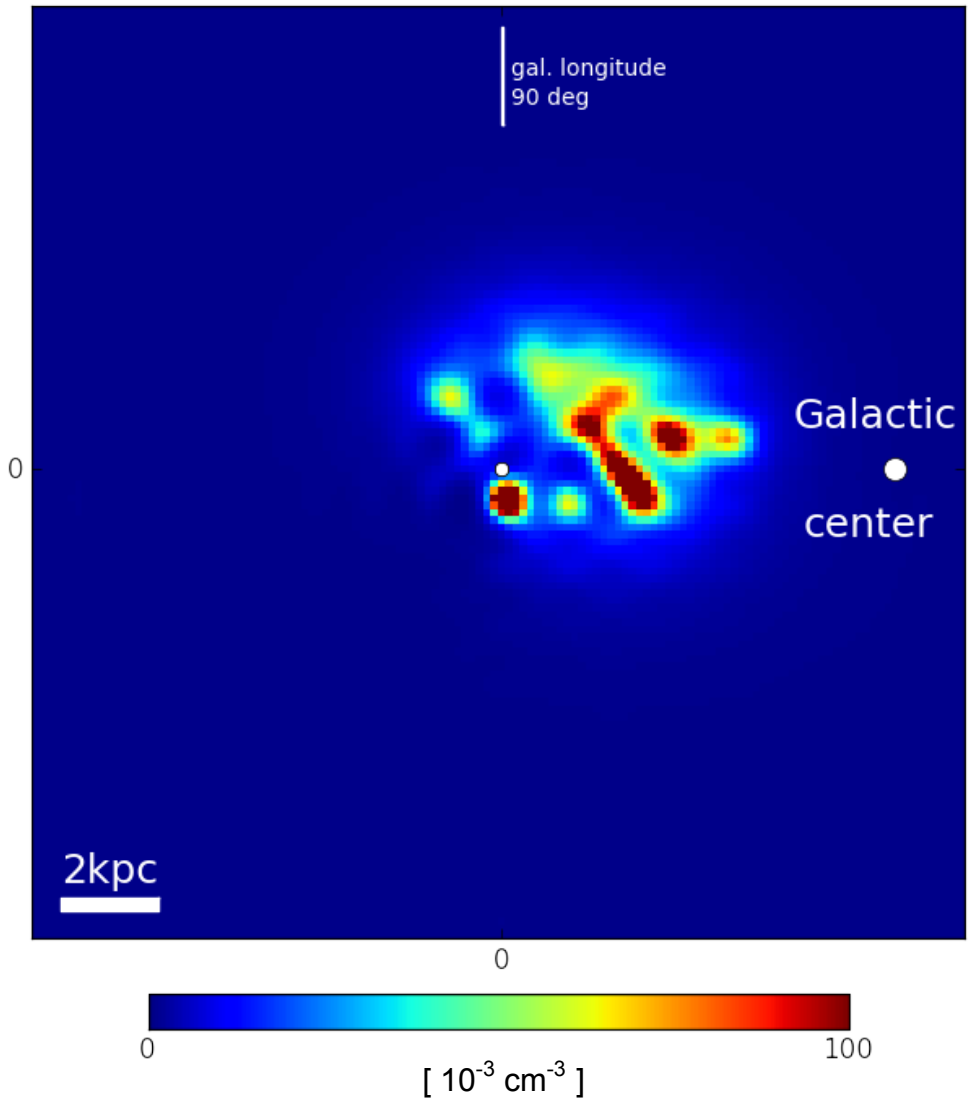
- data model
  - dispersion measures from different lines of sight
  - additive Gaussian noise

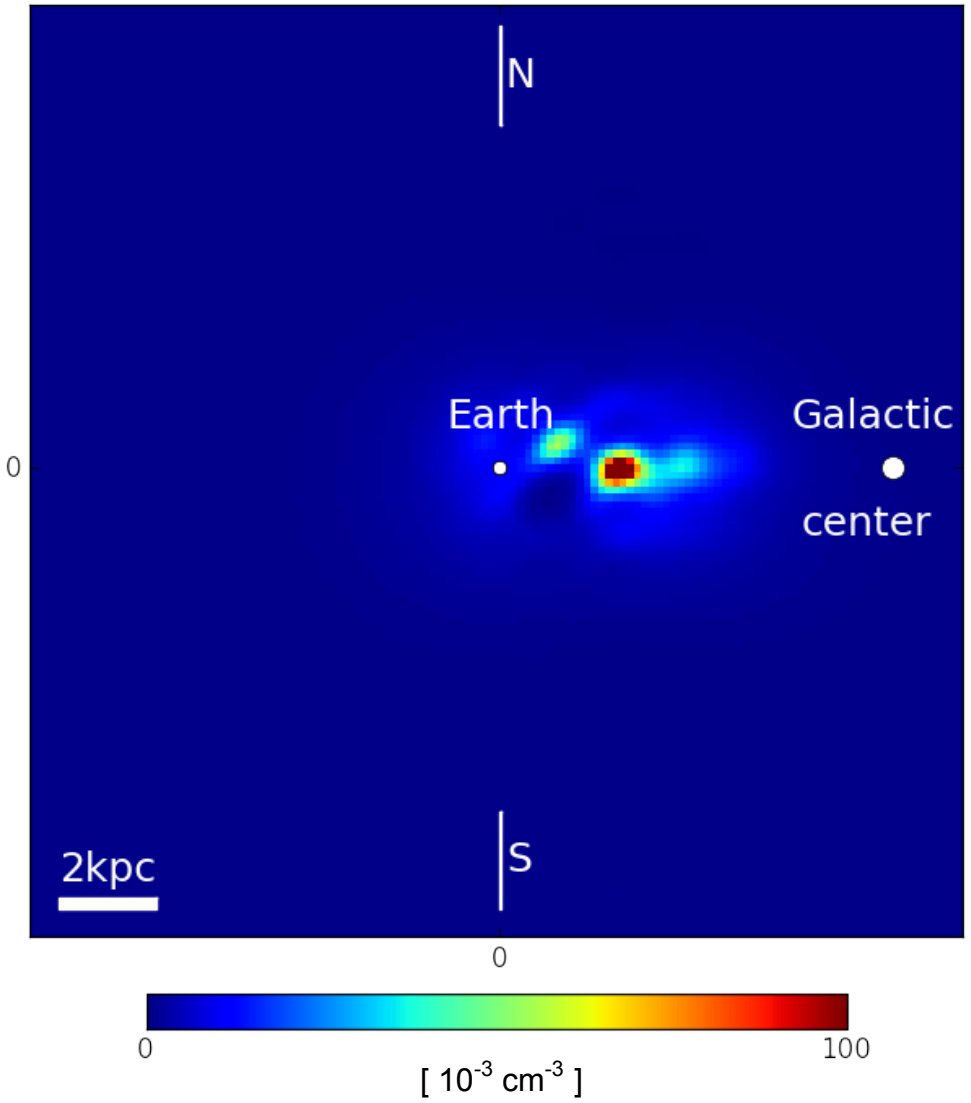
$$\langle \boldsymbol{d} \rangle_{(\boldsymbol{d}|\boldsymbol{\rho})} = \left(\int_{\text{observer}}^{\text{pulsar}} \mathrm{d}\vec{l}_j \; \rho(\vec{x}) \right)_j = \boldsymbol{R}\boldsymbol{\rho}$$

- *a priori* assumptions
  - electron density $\boldsymbol{\rho}$ $\leftarrow$ log-normal prior
  - unknown correlations

The Galactic free electron density

Greiner et al. (in prep.)

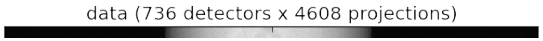


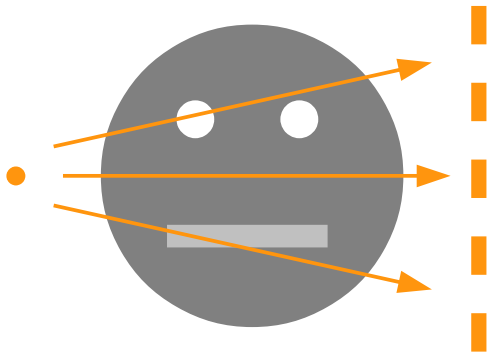


Computer tomography

- data model
  - absorption along the line of sight
  - Poissonian noise

$$\langle \boldsymbol{d} \rangle_{(\boldsymbol{d}|\boldsymbol{\rho})} = \lambda_0 \exp\left(-a \int_{\text{source}}^{\text{detector}} \mathrm{d}\vec{l}_j \; \rho(\vec{x}) \right)_j = \lambda_0 \exp\left(-a\boldsymbol{R}\boldsymbol{\rho}\right)$$

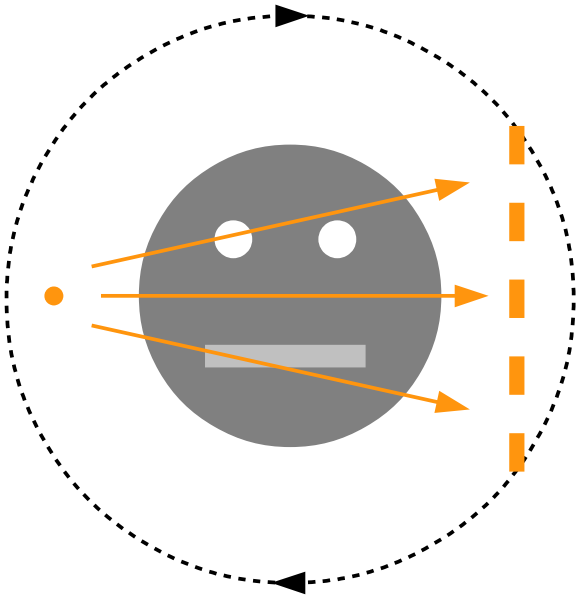


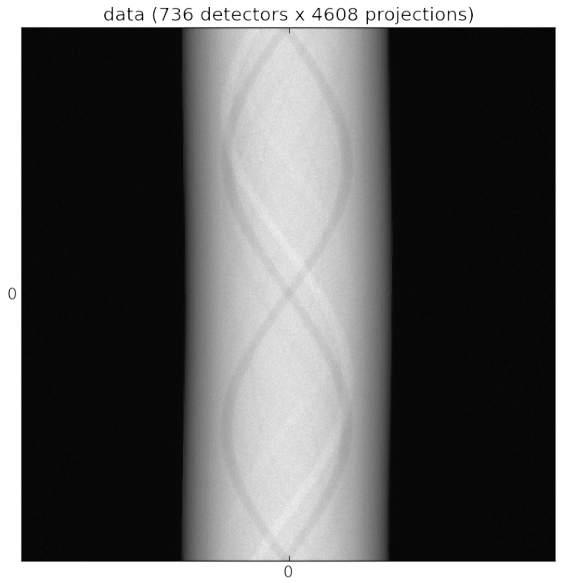


Computer tomography

- data model
  - absorption along the line of sight
  - Poissonian noise

$$\langle \boldsymbol{d} \rangle_{(\boldsymbol{d}|\boldsymbol{\rho})} = \lambda_0 \exp\left(-a \int_{\text{source}}^{\text{detector}} \mathrm{d}\vec{l}_j \; \rho(\vec{x}) \right)_j = \lambda_0 \exp\left(-a\boldsymbol{R}\boldsymbol{\rho}\right)$$





Computer tomography

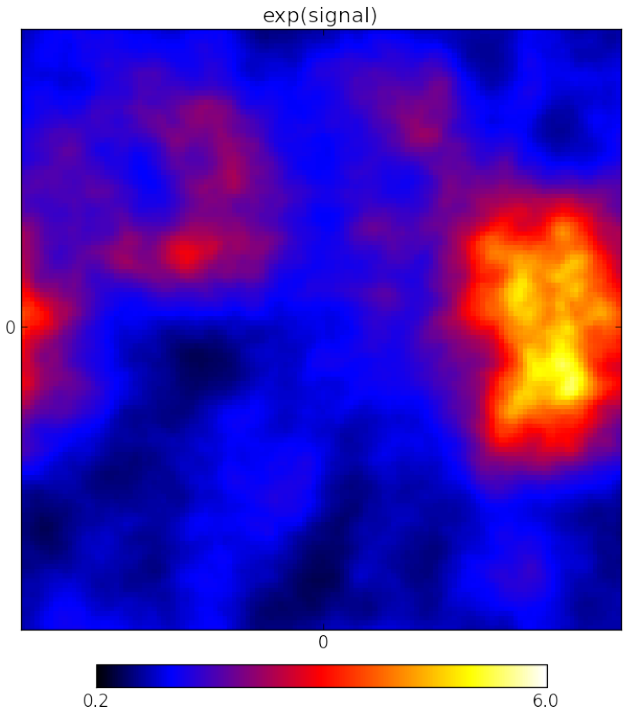
- data model
  - absorption along the line of sight
  - Poissonian noise

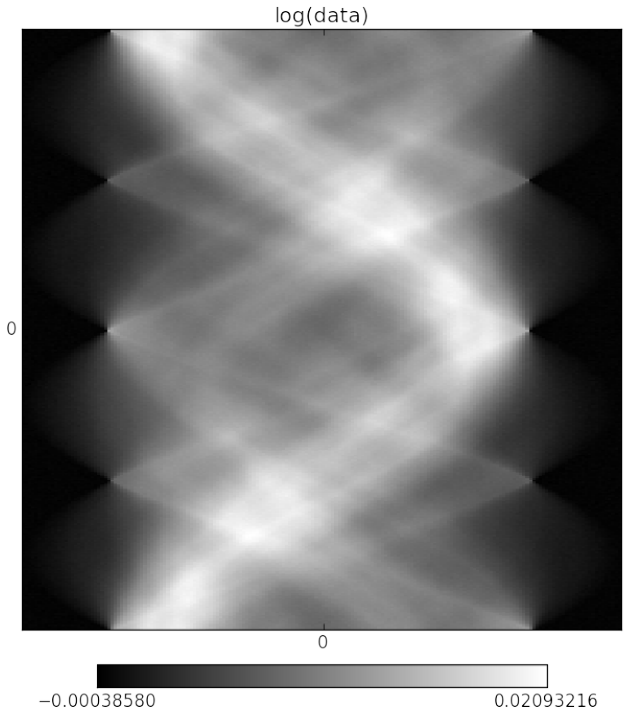
$$\langle \boldsymbol{d} \rangle_{(\boldsymbol{d}|\boldsymbol{\rho})} = \lambda_0 \exp\left(-a \int_{\text{source}}^{\text{detector}} \mathrm{d}\vec{l}_j \; \rho(\vec{x}) \right)_j = \lambda_0 \exp\left(-a\boldsymbol{R}\boldsymbol{\rho}\right)$$

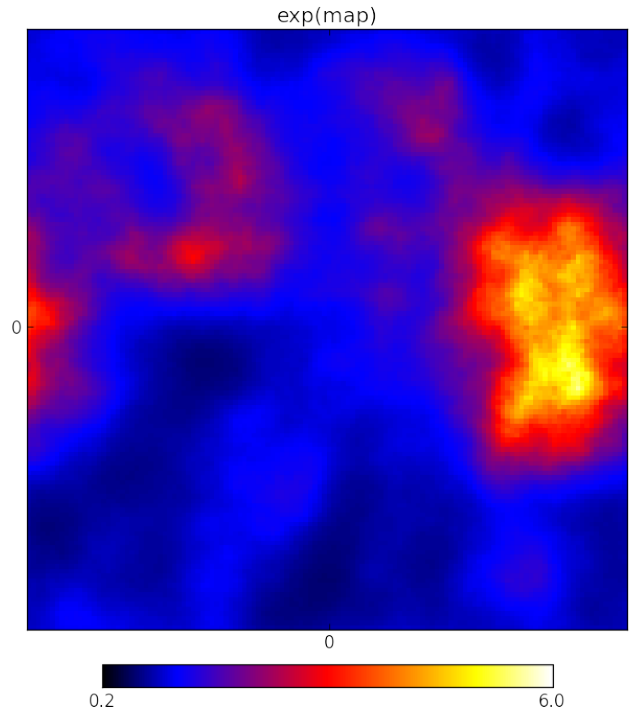
- *a priori* assumptions
  - matter density $\boldsymbol{\rho}$ ← log-normal prior
  - known correlations ← medical databases

Computer tomography

$$\mathrm{a} = 1 \quad , \quad \lambda_0 = 10^8$$

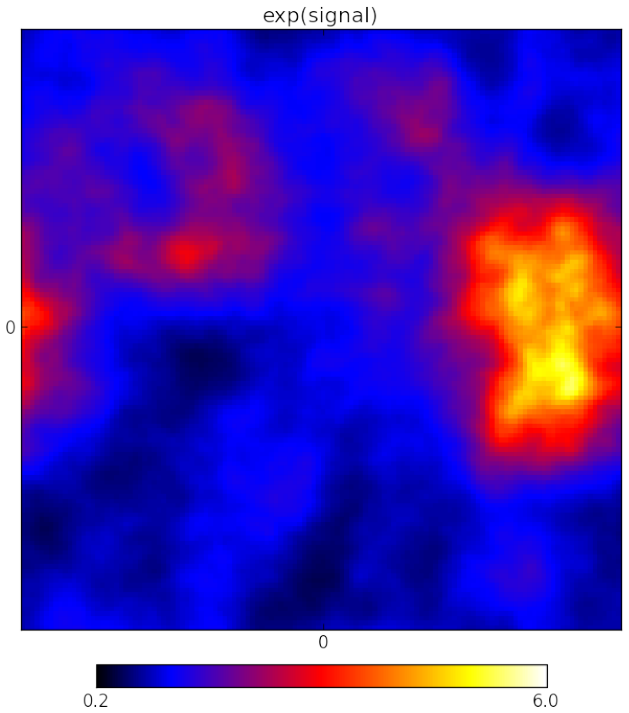


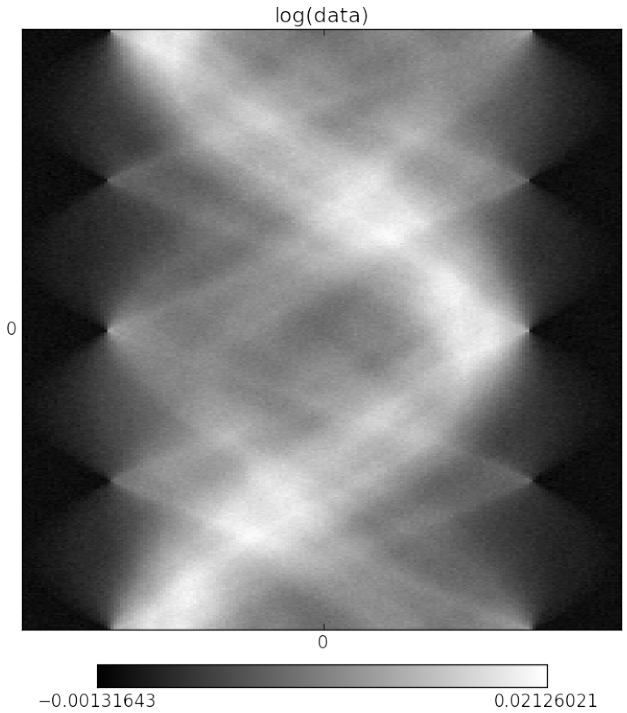


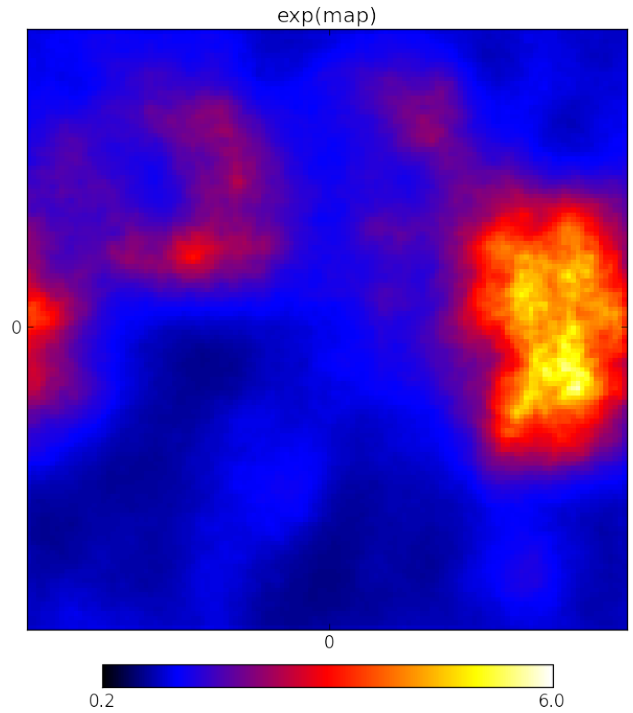


Computer tomography

$$\mathrm{a} = 1 \quad , \quad \lambda_0 = 10^7$$

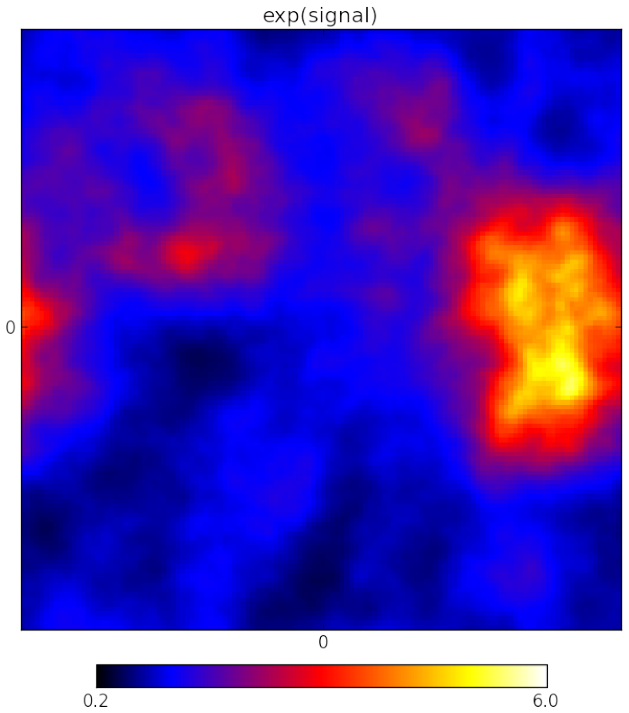


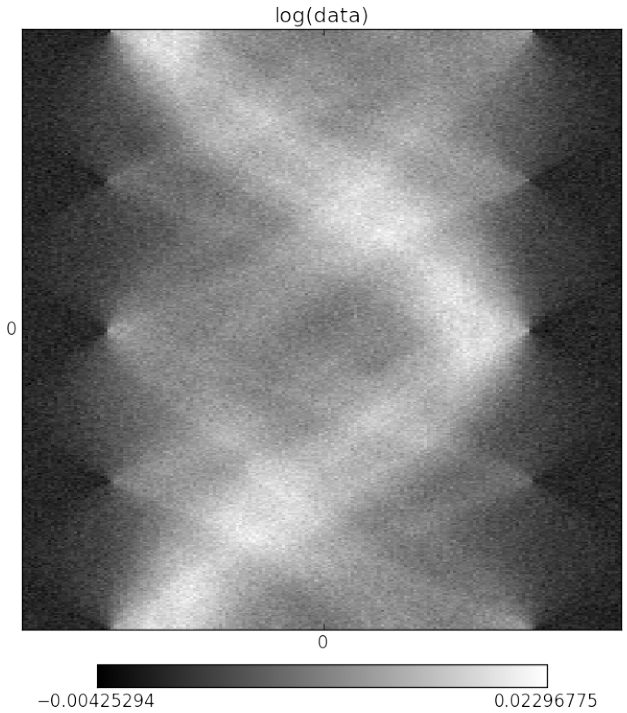


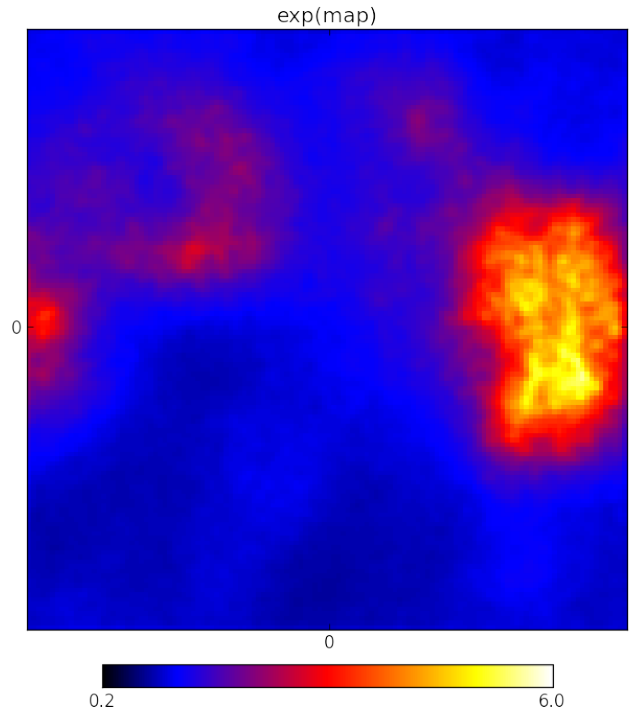


Computer tomography

$$\mathrm{a} = 1 \quad , \quad \lambda_0 = 10^6$$

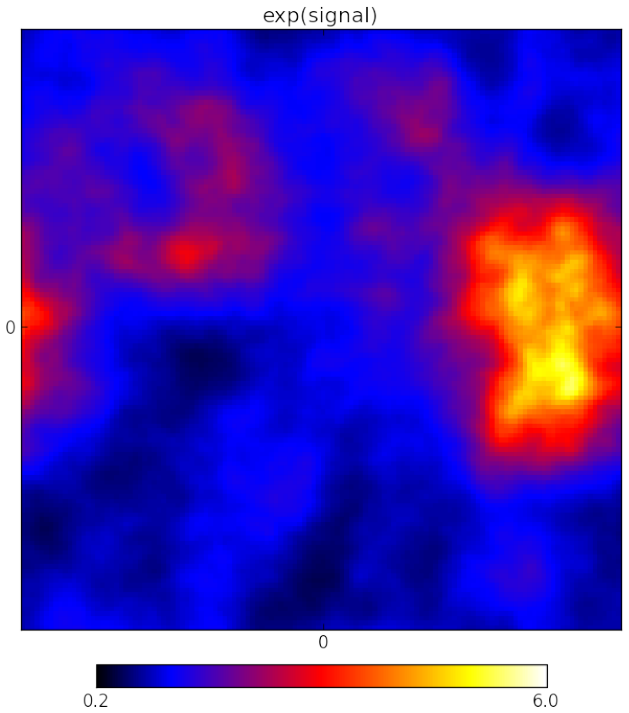


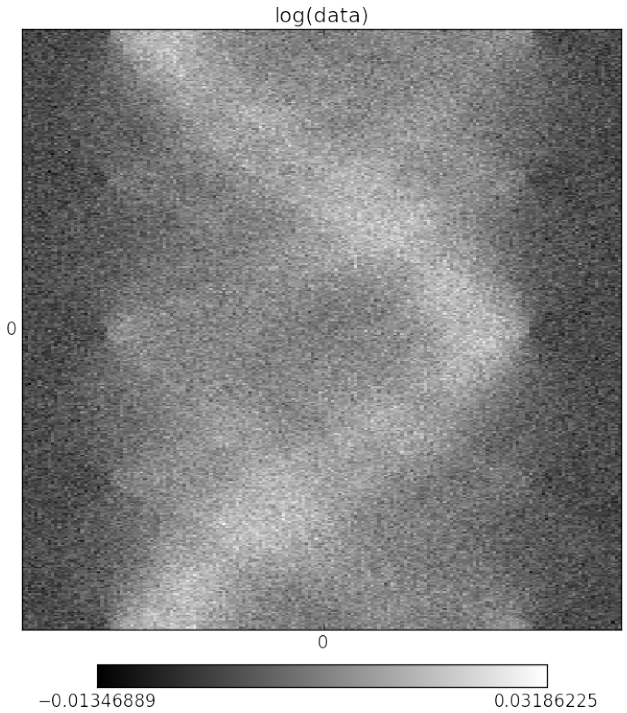


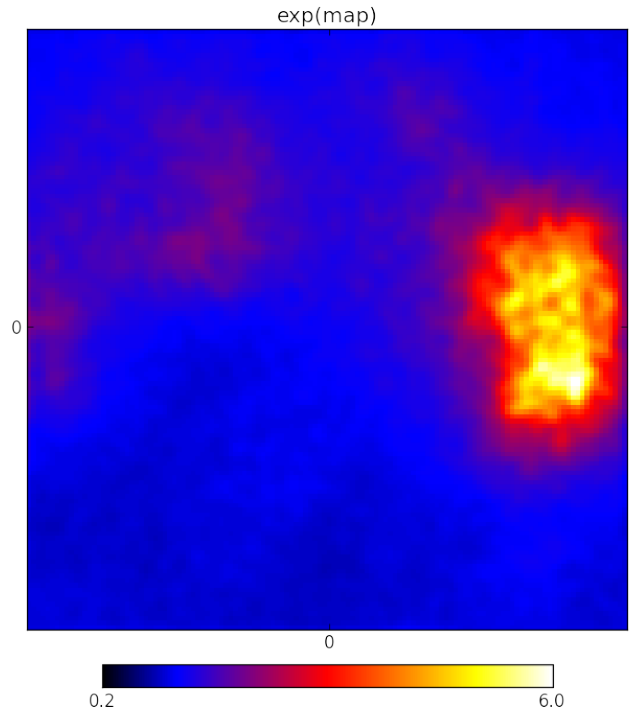


Computer tomography

$$\mathrm{a} = 1 \quad , \quad \lambda_0 = 10^5$$

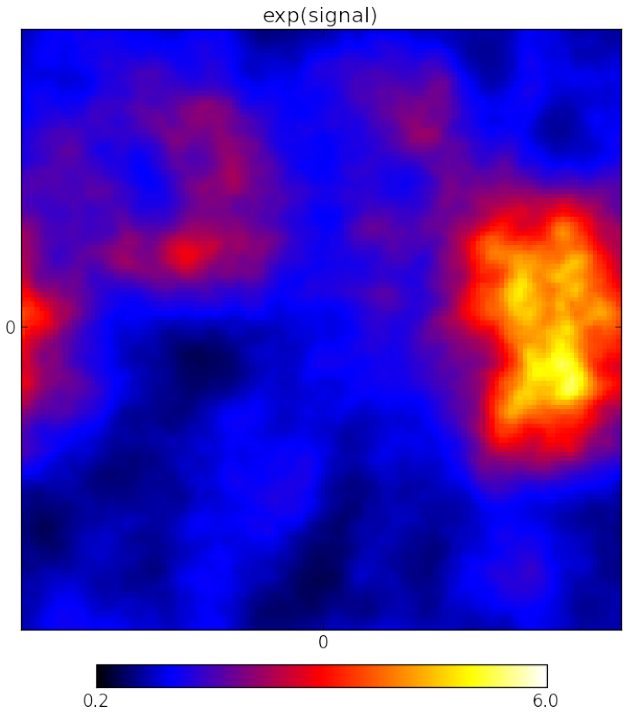


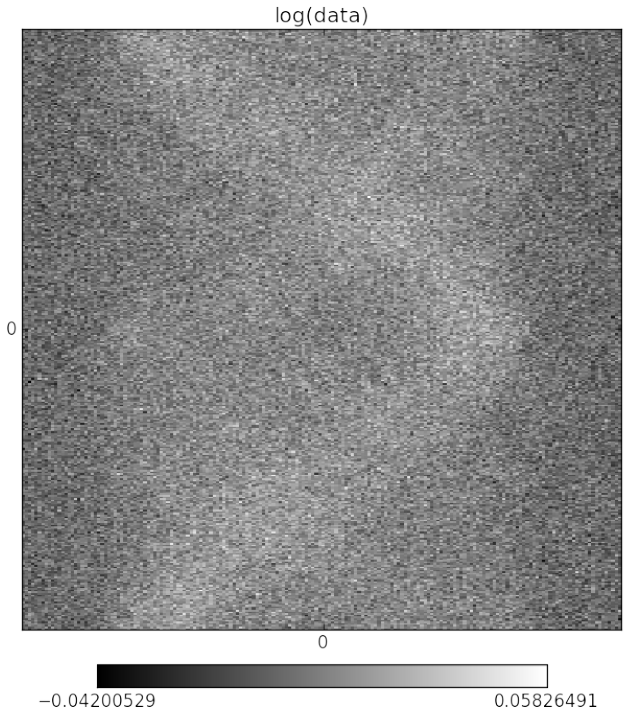


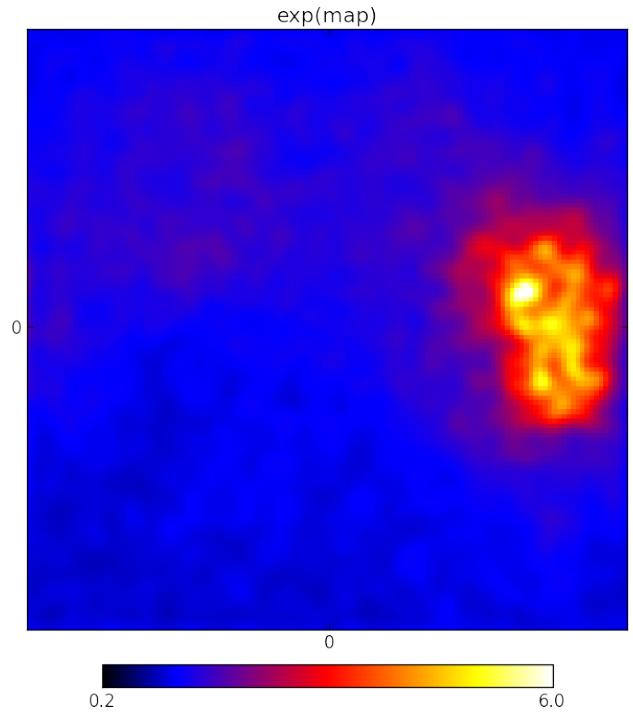


Computer tomography

$$\mathrm{a} = 1 \quad , \quad \lambda_0 = 10^4$$

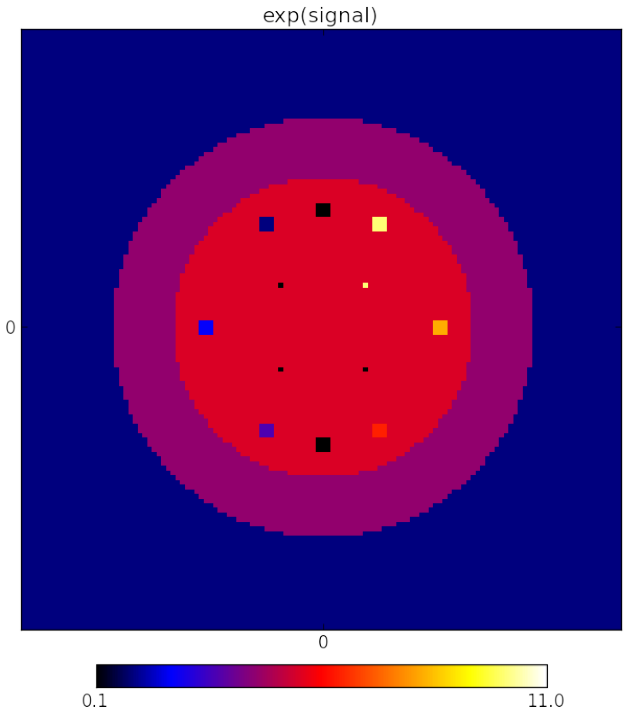


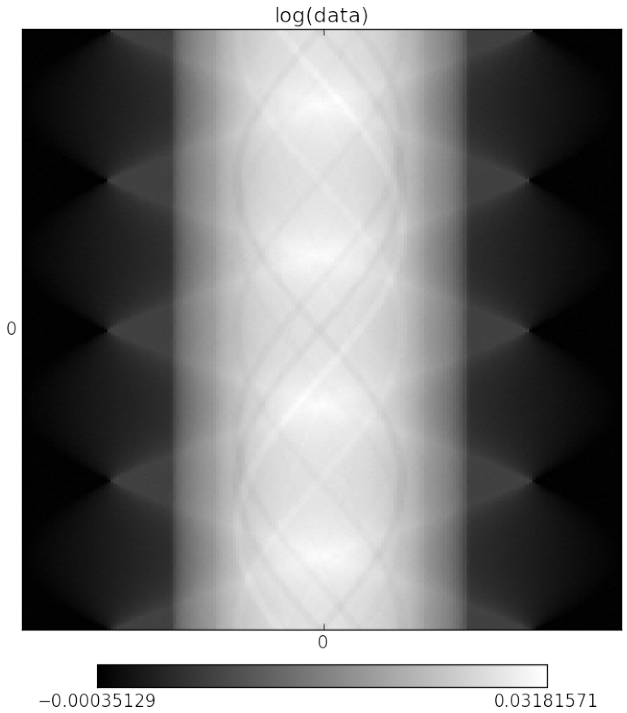


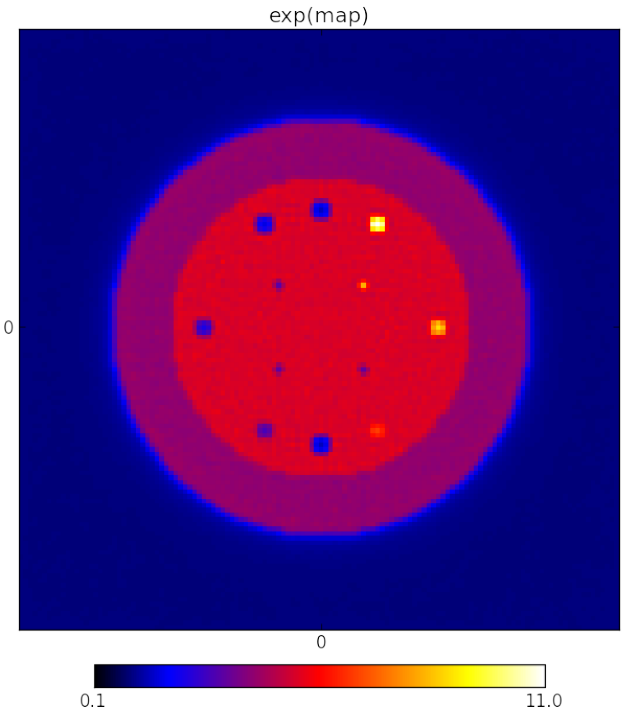


Computer tomography

$$\mathrm{a} = 1 \quad , \quad \lambda_0 = 10^8$$

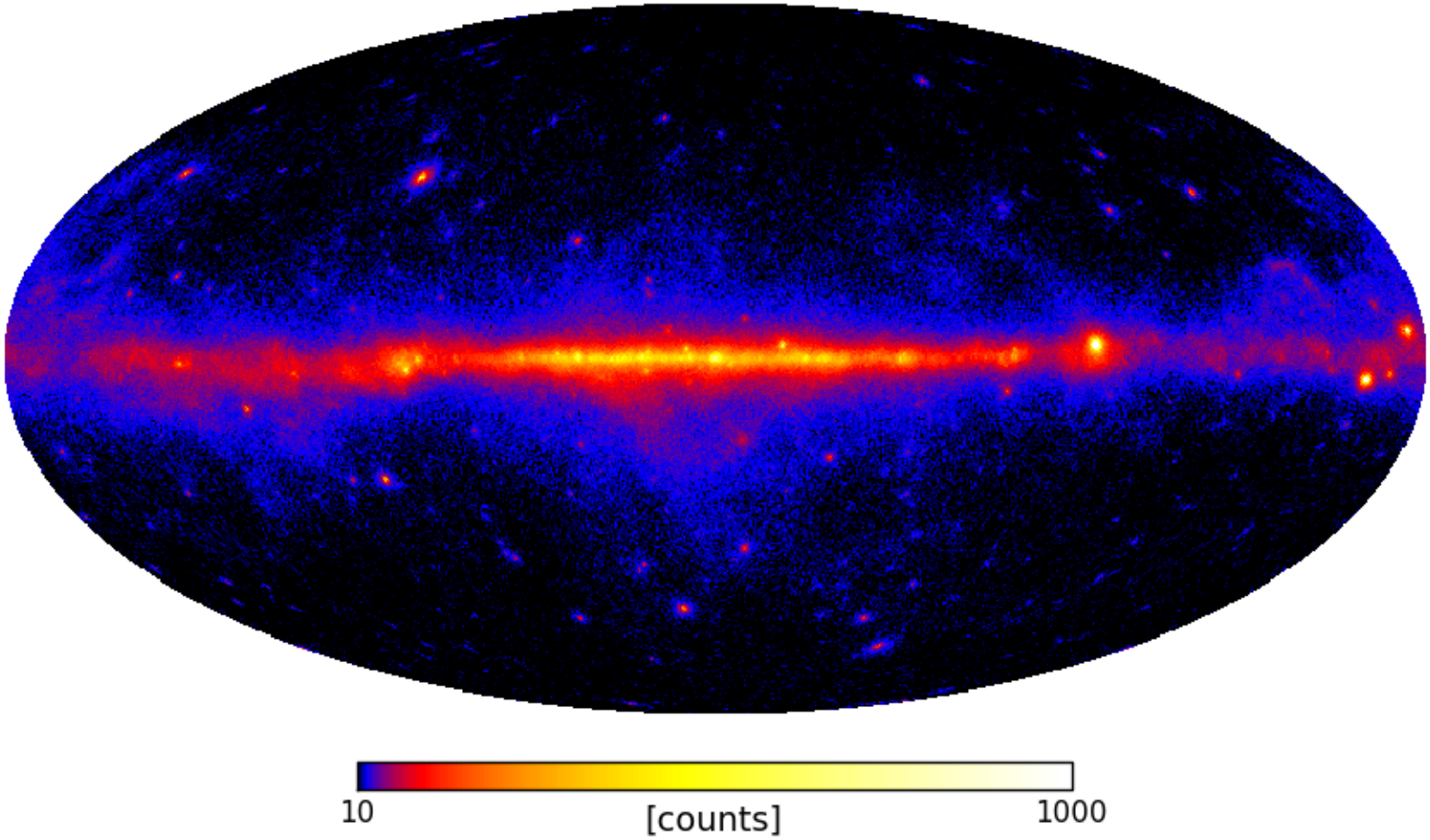






The Fermi γ-ray sky

Selig et al. (in prep.)



$$\mathrm{E}_\gamma \sim 100\,\mathrm{MeV} \,..\, 100\,\mathrm{GeV} \quad , \quad t_{\mathrm{mission}} \sim \mathrm{week}\ 9 \,..\, 220$$

The Fermi γ-ray sky

Selig et al. (in prep.)

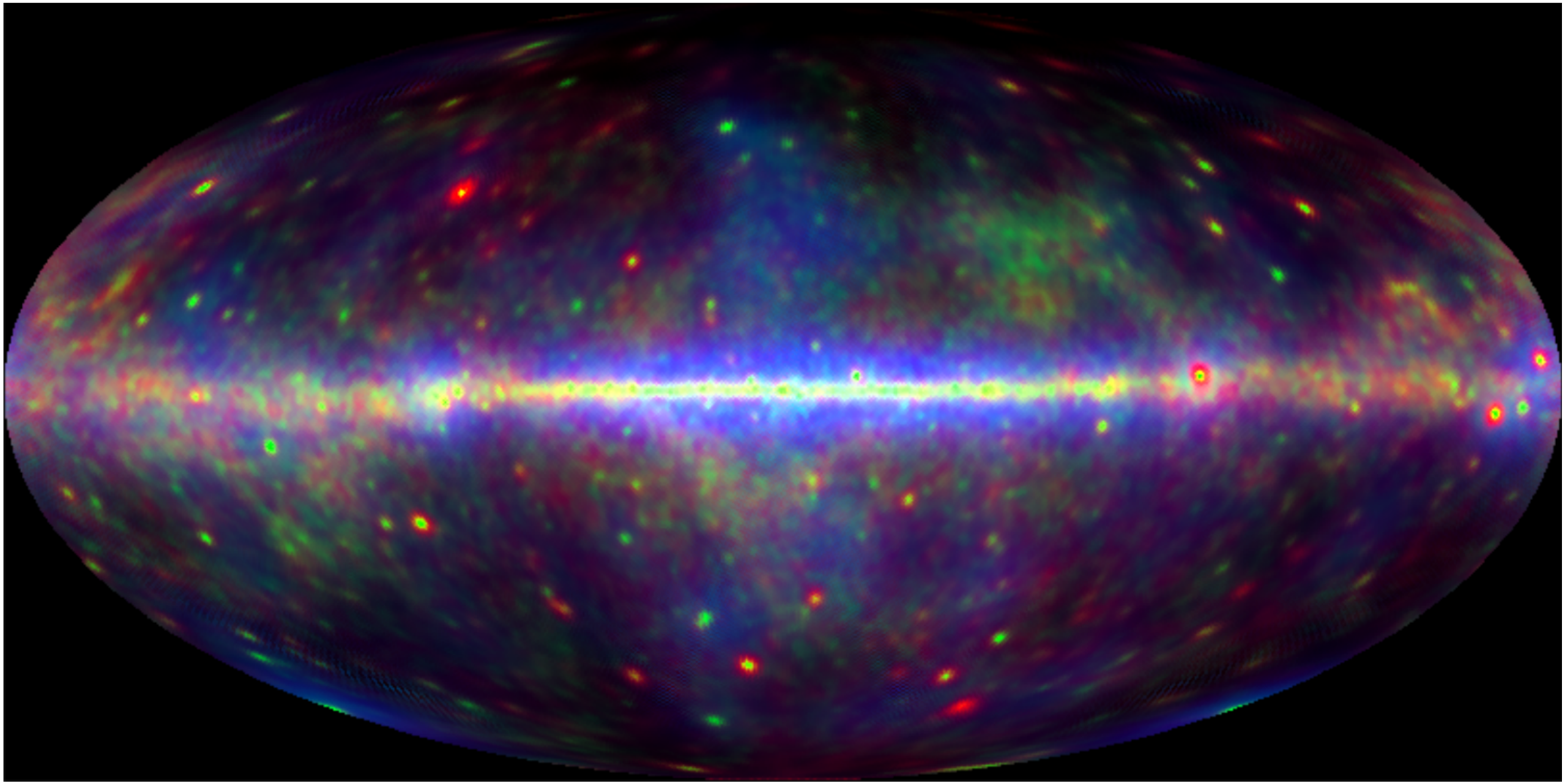
- data model
  - uneven survey coverage
  - Poissonian noise

$$\langle \boldsymbol{d} \rangle_{(\boldsymbol{d}|\boldsymbol{\rho})} = \boldsymbol{R}\boldsymbol{\rho}$$

- *a priori* assumptions
  - diffuse flux $\boldsymbol{\rho} \leftarrow$ log-normal prior
  - unknown correlations (but spectral smoothness)

The Fermi γ-ray sky

Selig et al. (in prep.)



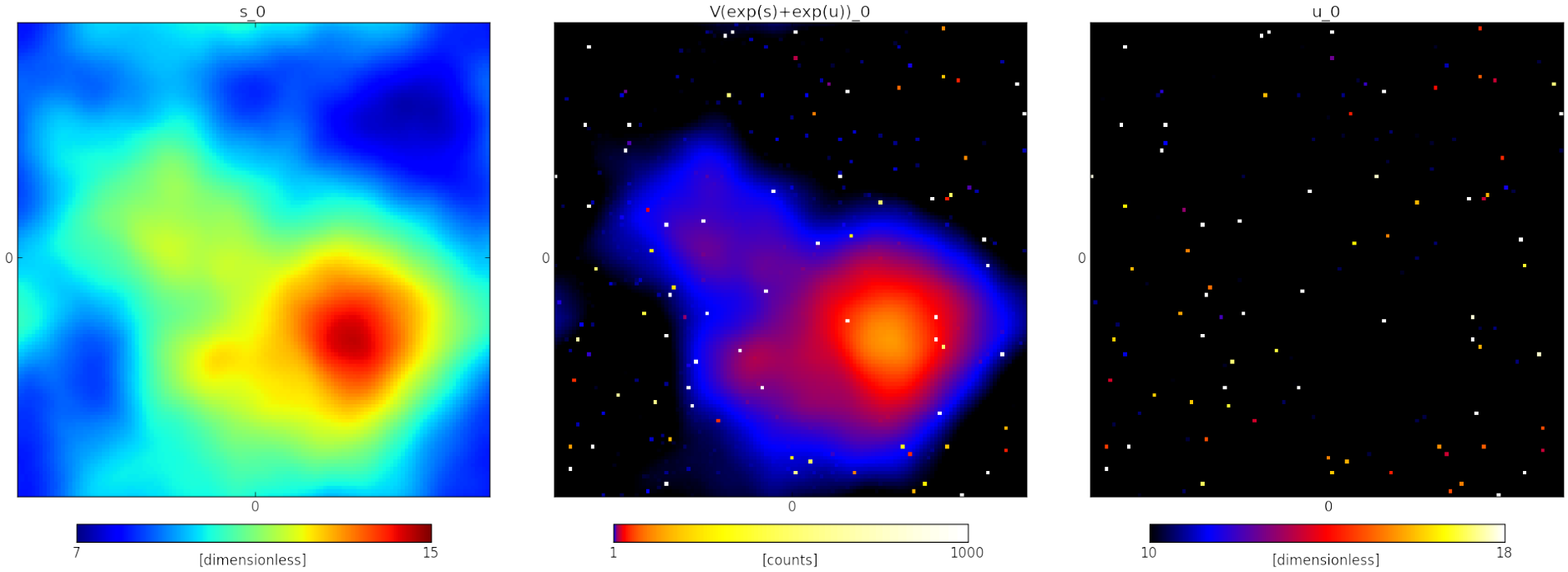
Component separation

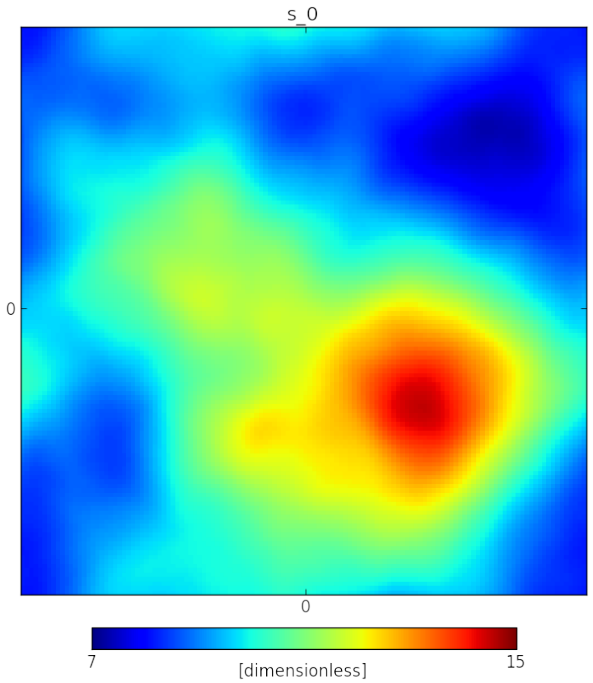
Selig et al. (in prep.)

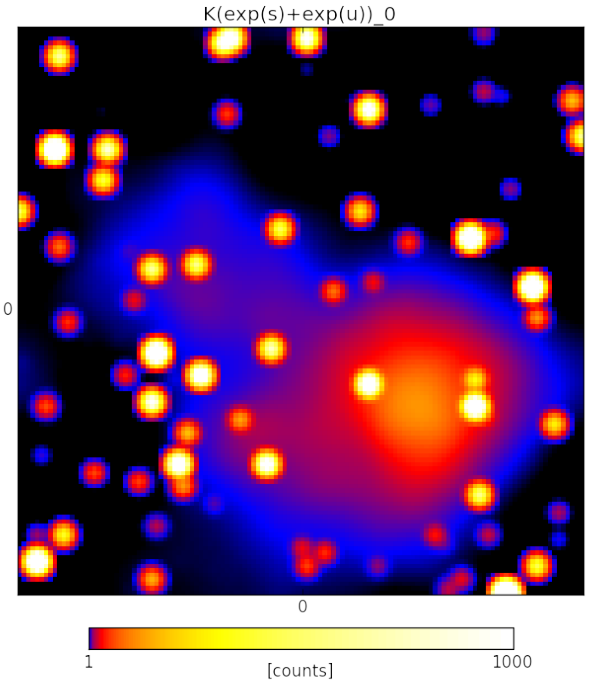
- data model
  - superposition of flux components
  - complex instrument response function
  - Poissonian noise

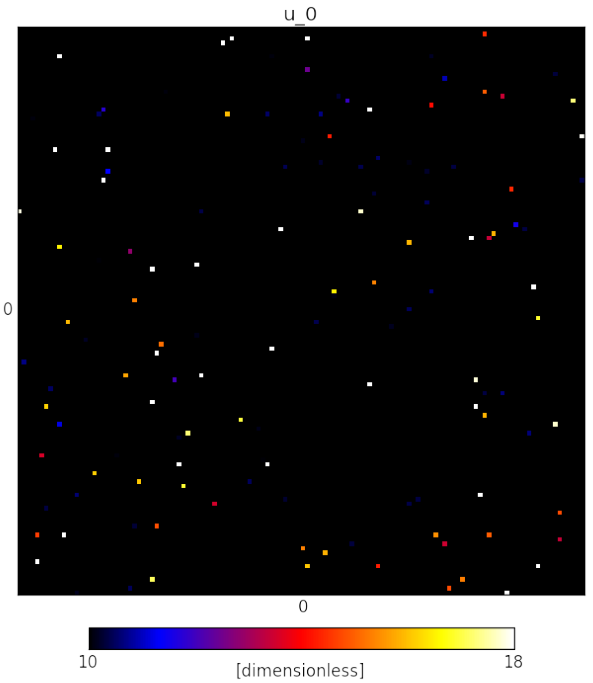
$$\langle \boldsymbol{d} \rangle_{(\boldsymbol{d}|\boldsymbol{\rho})} = \boldsymbol{R}\left(\boldsymbol{\rho}_s + \boldsymbol{\rho}_u\right)$$

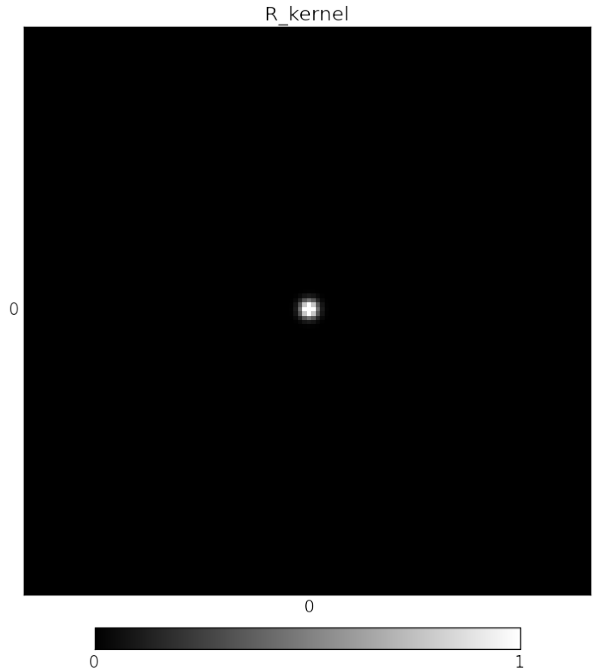
- *a priori* assumptions
  - diffuse flux $\boldsymbol{\rho}_s \leftarrow$ log-normal prior
  - known correlations
  - point source flux $\boldsymbol{\rho}_u \leftarrow$ inverse-Gamma priors

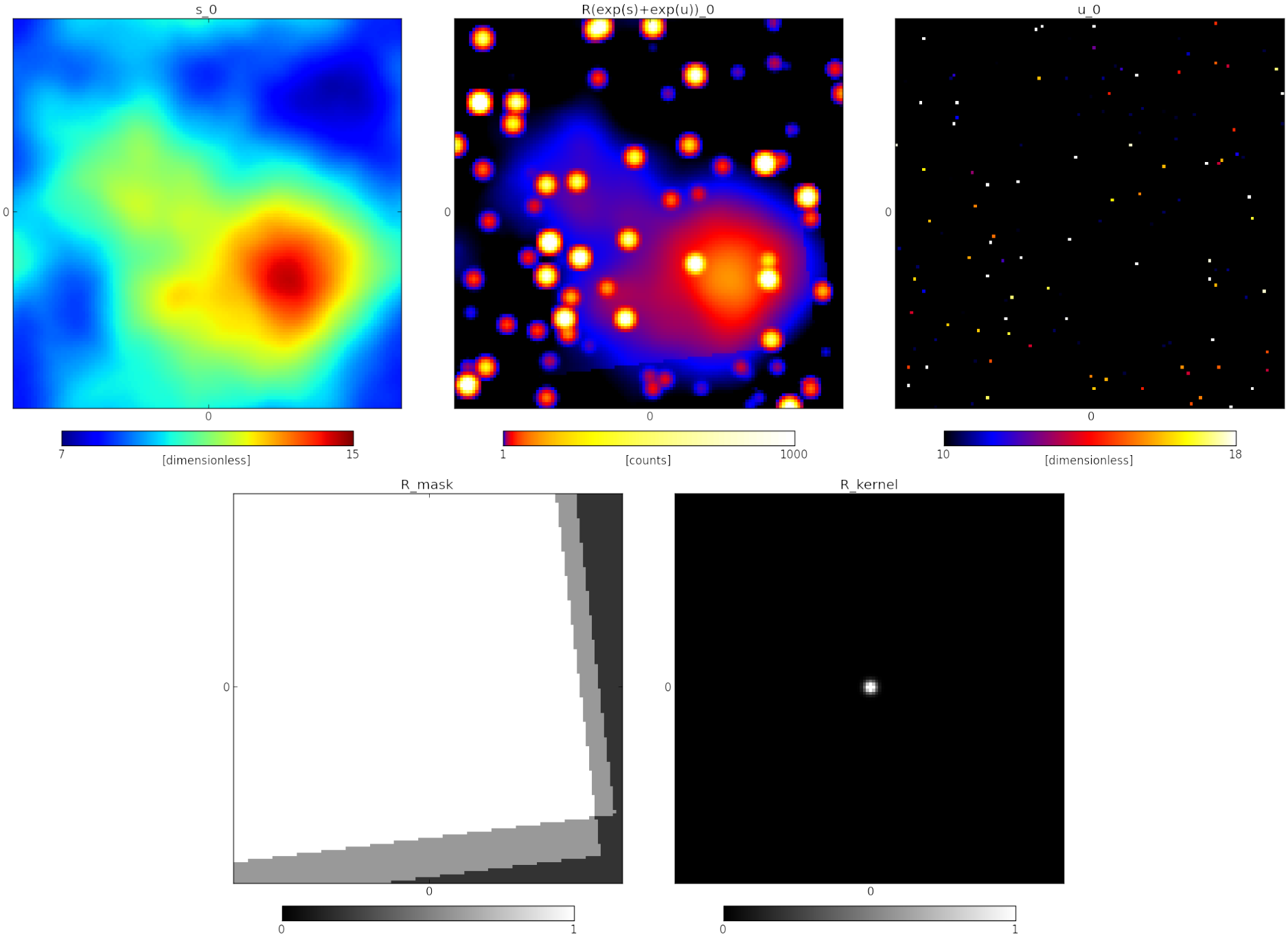

s_0
0
0
7
[dimensionless]
15
V(exp(s)+exp(u))_0
0
0
1
[counts]
1000
u_0
0
0
10
[dimensionless]
18

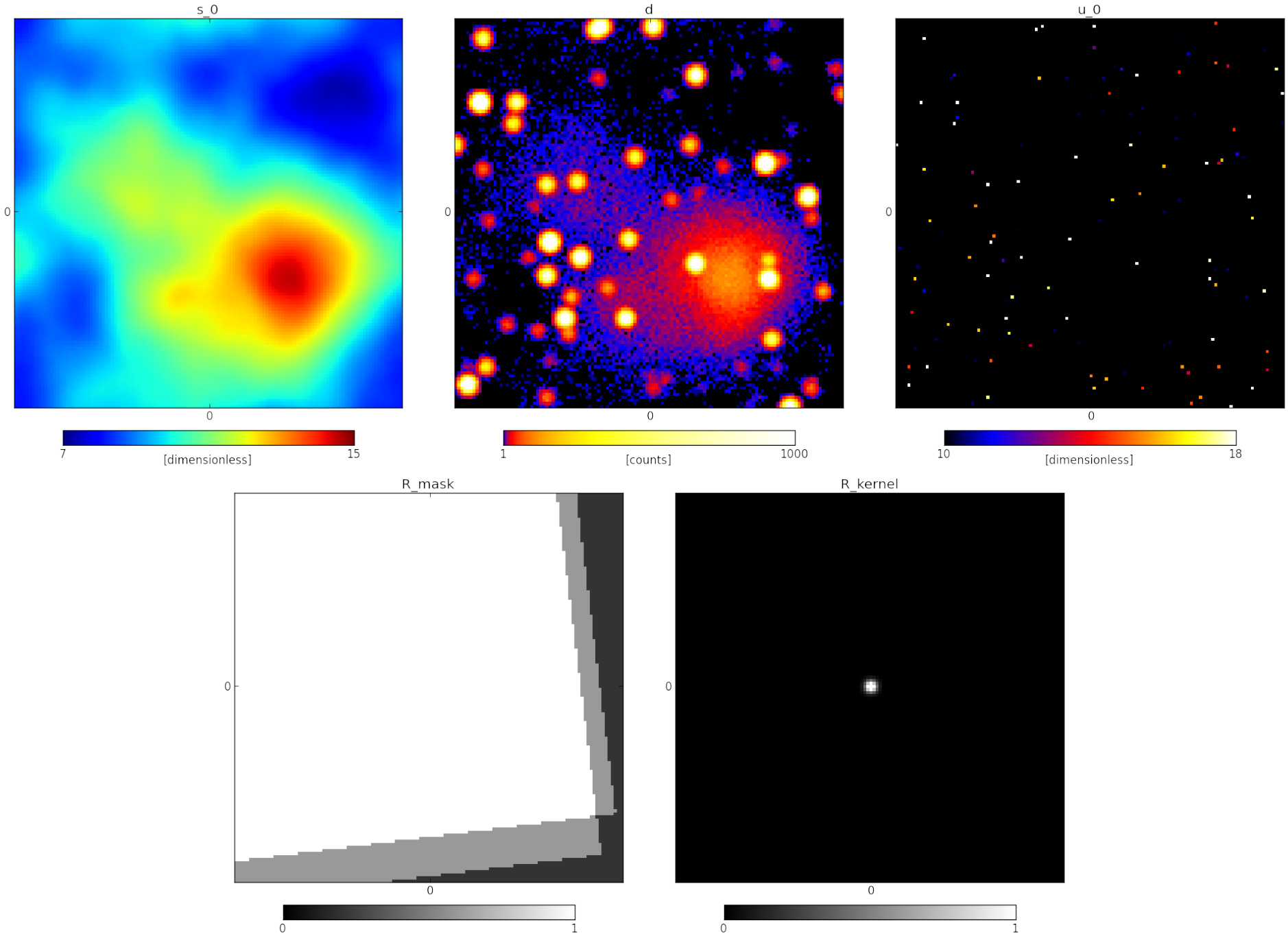

s_0
0
0
7
[dimensionless]
15

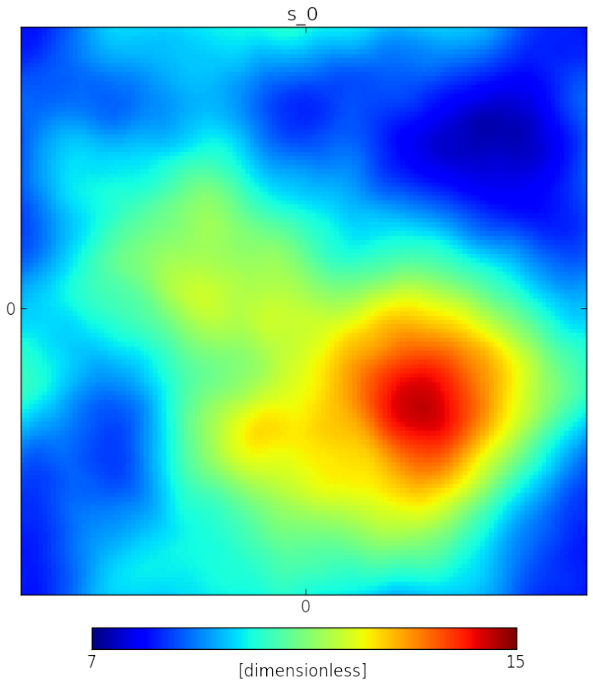

K(exp(s)+exp(u))_0
0
0
1
[counts]
1000

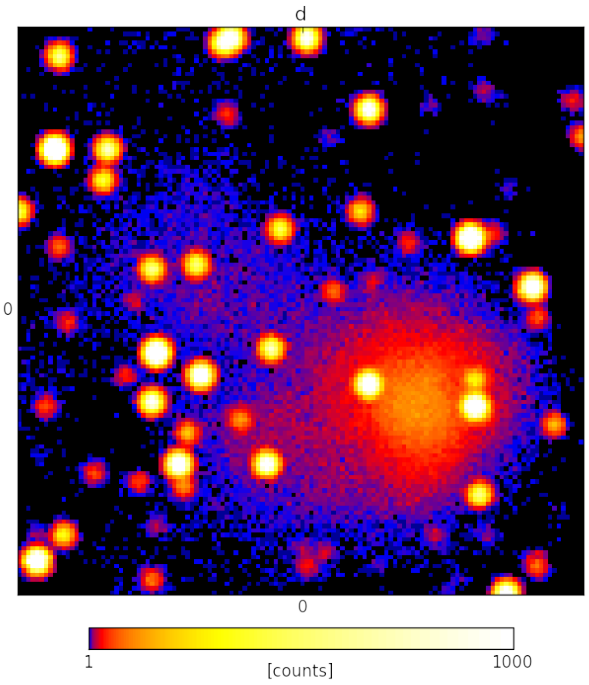

u_0
0
0
10
[dimensionless]
18

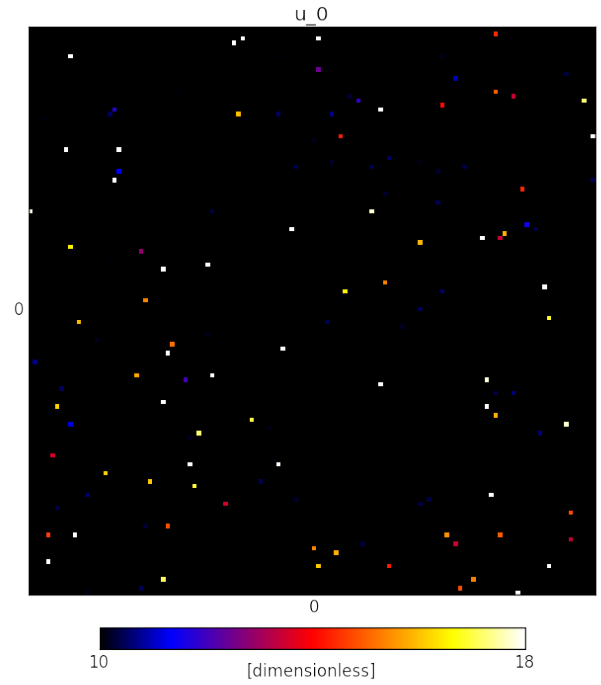

R_kernel
0
0
0
1

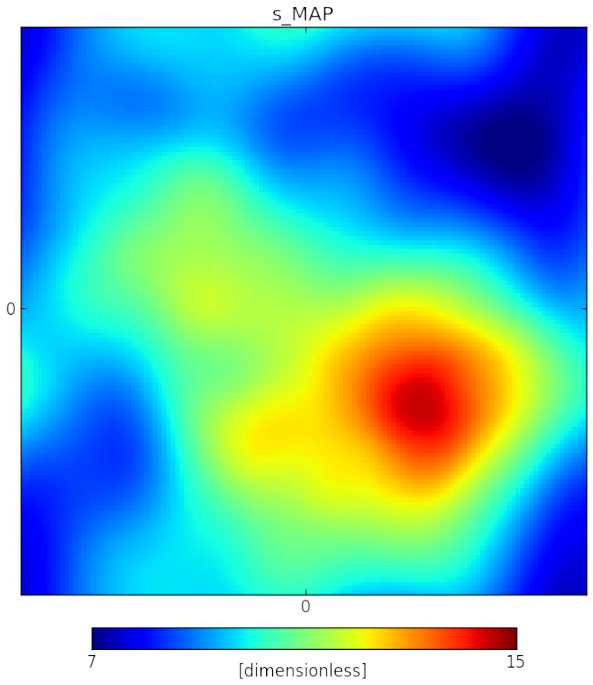

s_0
0
0
7
[dimensionless]
15
R(exp(s)+exp(u))_0
0
0
1
[counts]
1000
u_0
0
0
10
[dimensionless]
18
R_mask
0
0
0
1
R_kernel
0
0
0
1

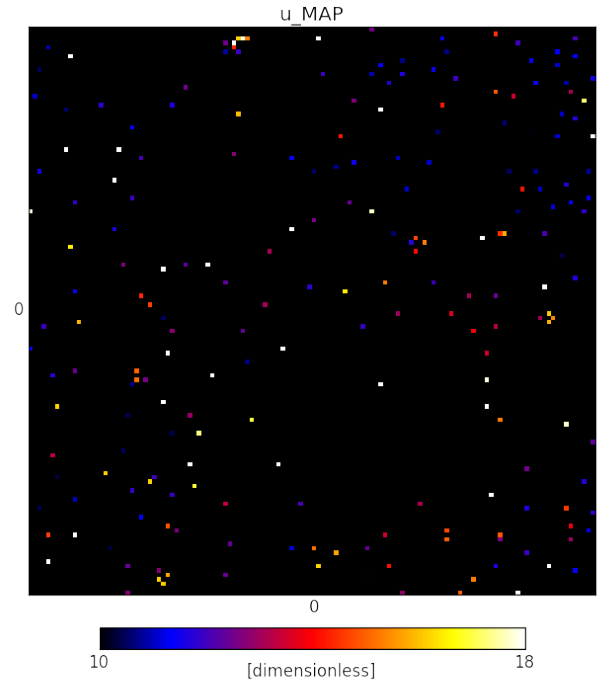

s_0
0
0
7
[dimensionless]
15
d
0
0
1
[counts]
1000
u_0
0
0
10
[dimensionless]
18
R_mask
0
0
0
1
R_kernel
0
0
0
1

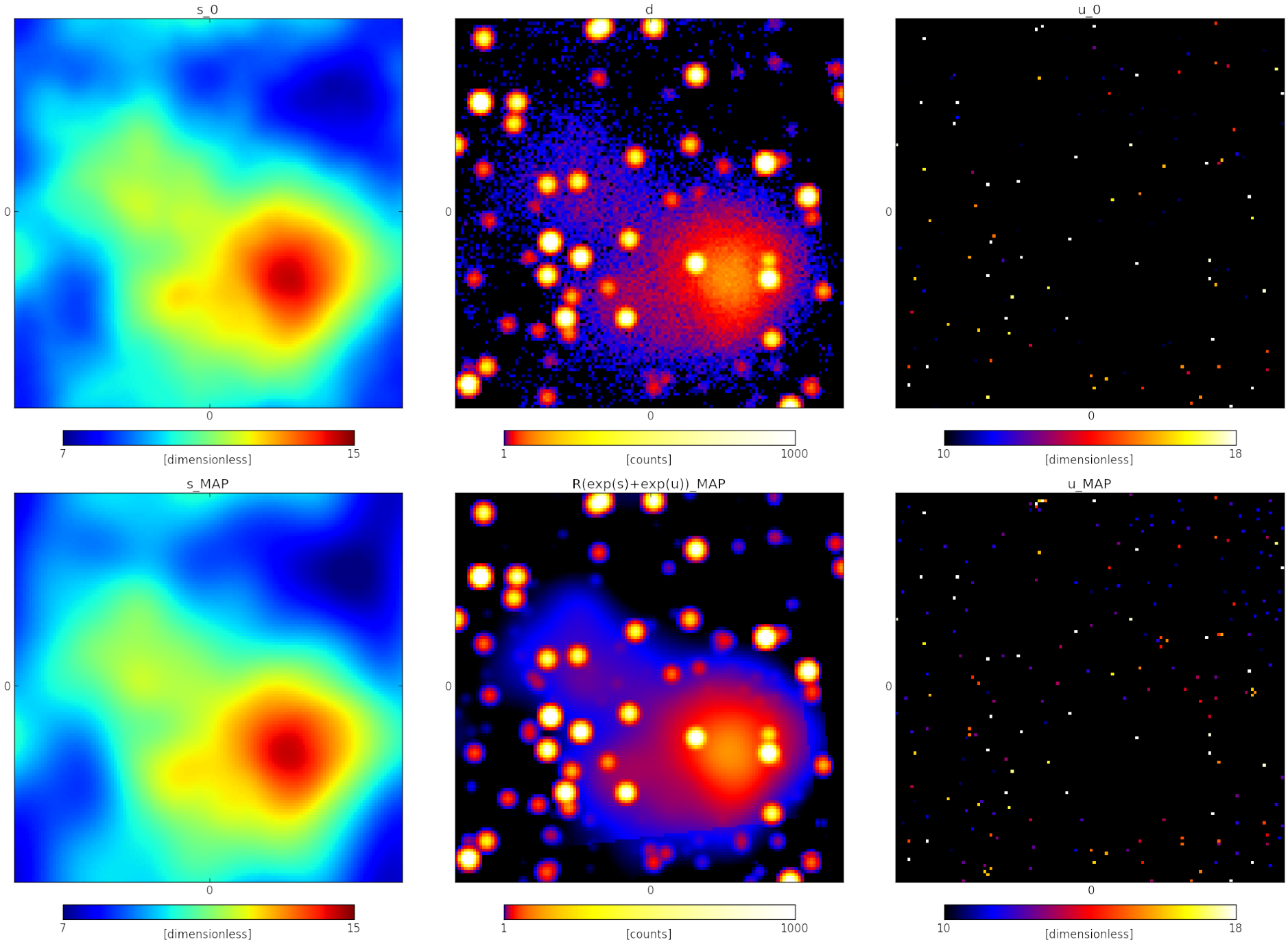

s_0
0
0
7
[dimensionless]
15


d
0
0
1
[counts]
1000


u_0
0
0
10
[dimensionless]
18


s_MAP
0
0
7
[dimensionless]
15


u_MAP
0
0
10
[dimensionless]
18


s_0
d
u_0
s_MAP
R(exp(s)+exp(u))_MAP
u_MAP
0
7
15
1
1000
10
18
[dimensionless]
[counts]

Summary

Summary

- effective **IFT** framework
  - inference on continuous signal fields
  - treatment of unknown correlations
- useful **NIFTy** library
  - versatile toolbox for signal inference algorithms
  - grid and resolution independence
  - applicability to real-life problems
  - extensive documentation (including tutorials)

Spectral smoothness prior

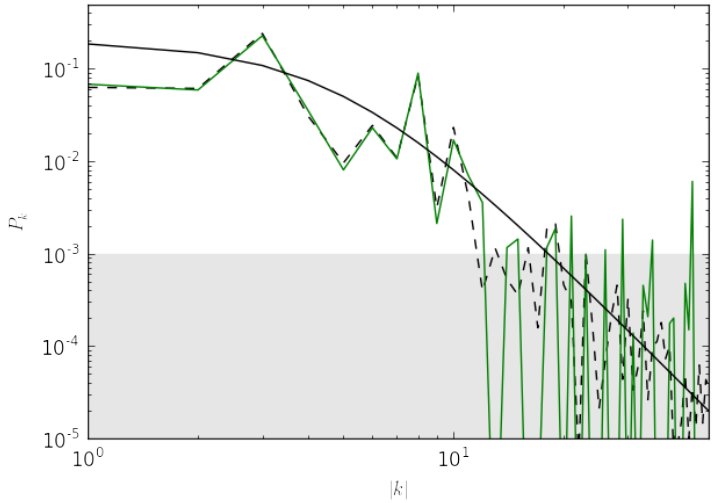
Oppermann et al. (2012)

- unknown signal correlations

$$\boldsymbol{S} = \sum_k p_k \boldsymbol{S}_k \quad , \quad \boldsymbol{p} \curvearrowleft \qquad\qquad \prod_k \mathcal{I}\left(p_k, \alpha \to 1, q \to 0\right)$$

- inverse-Gamma prior

$$\mathcal{I}\left(p_k, \alpha, q\right) \propto p_k^{-\alpha} \exp\left(-\frac{q}{p_k}\right)$$



Spectral smoothness prior

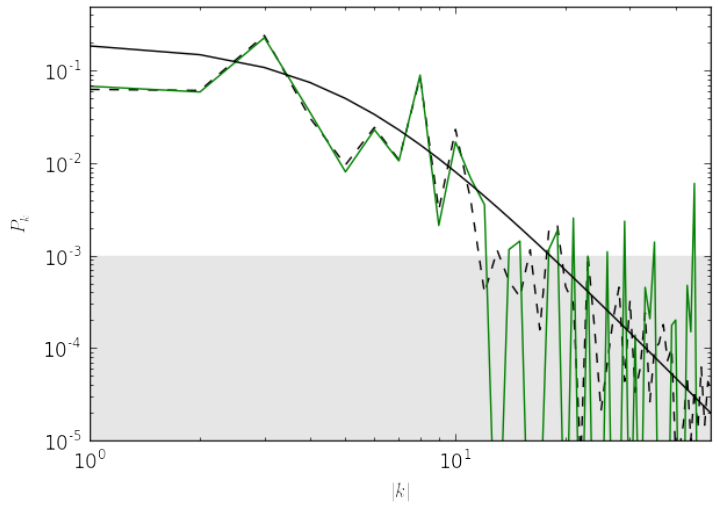
Oppermann et al. (2012)

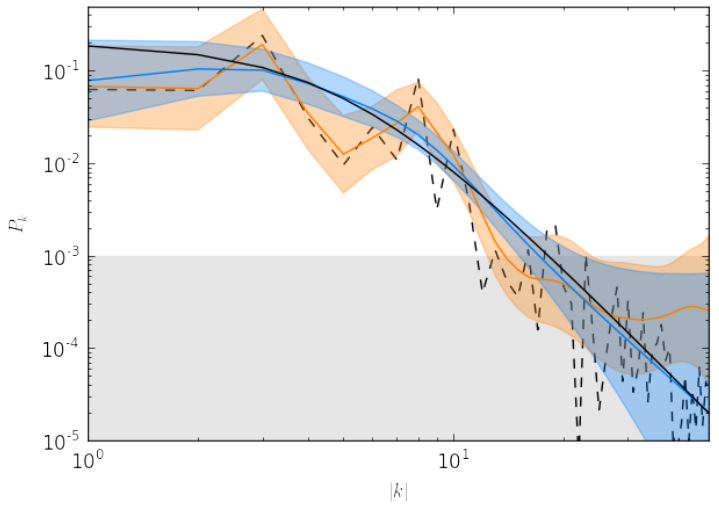
- unknown signal correlations

$$\boldsymbol{S} = \sum_k p_k \boldsymbol{S}_k \quad , \quad \boldsymbol{p} \curvearrowleft P_{\mathrm{sm}}(\boldsymbol{p}) \prod_k \mathcal{I}\left(p_k, \alpha \to 1, q \to 0\right)$$

- inverse-Gamma prior and spectral smoothness prior

$$P_{\mathrm{sm}}(\boldsymbol{p}) \propto \exp\left(-\frac{1}{2\sigma^2} \int \mathrm{d}(\log k) \left(\frac{\partial^2 \log p_k}{\partial (\log k)^2}\right)^2\right)$$





$\sigma^2 = 1000$

$\sigma^2 = 10$