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


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

1. IFT – Information Field Theory
2. NIFTY – Numerical Information Field Theory
3. Applications
4. Summary
Theory
What's the problem?

- features in the Galactic diffuse $\gamma$-ray emission
- separation of diffuse and point-like components
- medical and Galactic tomography
Data \xrightarrow{\text{observation}} \text{Signal estimate}

- data vector
- finite set of numbers

\[ d = (d_1, d_2, d_3, \ldots)^T \]

- signal field
- infinite number of degrees of freedom

\[ s = s(x) , \quad x \in \Omega \]
Bayes' Theorem ...

\[ P(s|d) = \frac{P(d|s)P(s)}{P(d)} \]
Bayes' Theorem …

\[ P(s|d) = \frac{P(d|s)P(s)}{P(d)} = \frac{\exp(-H(d, s))}{Z(d)} \]

\[ H(d, s) = -\log(P(d, s)) \]

… Information Field Theory
Wiener filter

• data model
  ◦ linear response
  ◦ additive Gaussian noise
  \[ d = Rs + n \]

• a priori assumptions
  ◦ signal \( s \) ← multidimensional Gaussian prior

• information Hamiltonian
  \[
  H(d, s) = \frac{1}{2} (d - Rs)^\dagger N^{-1} (d - Rs) + \frac{1}{2} s^\dagger S^{-1} s + \text{[const.]} \\
  = H_0 + \frac{1}{2} s^\dagger \left( S^{-1} + R^\dagger N^{-1} R \right) s + s^\dagger \left( R^\dagger N^{-1} d \right)
  \]
Wiener filter

- *a posteriori* solution

\[
\langle s \rangle_{(s|d)} = m = \left( S^{-1} + R^\dagger N^{-1} R \right)^{-1} \left( R^\dagger N^{-1} d \right)
\]

\[
\langle (s - m)(s - m)^\dagger \rangle_{(s|d)} = D
\]
Wiener filter

- *a posteriori* solution

\[
\langle s \rangle_{s|d} = m = \left( S^{-1} + R^\dagger N^{-1} R \right)^{-1} \left( R^\dagger N^{-1} d \right)
\]

---

**Theory**

**NIFTY**

**Applications**

**Summary**

---

**Observation**

**Inference**

**Reconstruction**

---

**Signal**

**Data**

**Reconstruction**

---
Discretizing continuous fields

\[ \varphi = \varphi(x), \quad x \in \Omega \]

\[ \varphi \mapsto \varphi_q \]
Discretizing continuous fields

\[ \varphi = \varphi(x), \quad x \in \Omega \]

\[ \varphi \mapsto \varphi_q \equiv \begin{cases} \langle \varphi(x) \rangle_{\Omega_q} \\ \varphi\left(\langle x \rangle_{\Omega_q}\right) \end{cases} \]
Discretizing continuous fields

\[ \varphi = \varphi(x), \quad x \in \Omega \]

\[ \varphi \mapsto \varphi_q \equiv \begin{cases} \langle \varphi(x) \rangle_{\Omega_q} \\ \varphi(\langle x \rangle_{\Omega_q}) \end{cases} \]

\[ \varphi^\dagger \psi = \int_{\Omega} dx \, \varphi^*(x) \psi(x) \approx \sum_q V_q \, \varphi_q^* \psi_q \]
The NIFTY way of Bayesian signal inference

by

Marco Selig
• is a versatile PYTHON library incorporating CYTHON, C++, and C libraries
The NIFTY way of Bayesian signal inference

- R
- Python
- Cython
- C, C++
- Fortran
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]:
NIFTY...

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

\[ p(k) = (k + 1)^{-3} \]
Grid independence

\[ p(k) = (k + 1)^{-3} \]
Grid independence

\[ p(k) = (k + 1)^{-3} \]
NIFTY...

- 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
**NIFTY classes**

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

**object**
**NIFTY classes**

- **space**
- **field**
- **operator**

---

- **point_space**
- **rg_space**
- **lm_space**
- **hp_space**
- **gl_space**
- **nested_space**

---

**object**

**unstructured list of points**

**Theory**

**NIFTY**

**Applications**

**Summary**

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

### Theory

**NIFTY**

- **space**: field → operator

### Applications

- Selig et al. (2013)
The NIFTY way of Bayesian signal inference

**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**
The NIFTY way of Bayesian signal inference

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)
NIFTY classes

- point_space
- rg_space
- lm_space
- hp_space
- gl_space
- nested_space

object

field

operator

probing

- diagonal_operator
- power_operator
- projection_operator
- vecvec_operator
- response_operator

Selig et al. (2013)
NIFTY...

- 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
Wiener filtering

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

class propagator(operator):
    def __init__(self, operator):
        # define propagator class

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

        def _multiply(self, x):
            # some numerical inversion 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")

d.plot(title="data", vmin=s.val.min(), vmax=s.val.max())  # plot signal

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
Wiener filtering

\[
d = R s + n
\]

\[
m = \left( S^{-1} + R^\dagger N^{-1} R \right)^{-1} \left( R^\dagger N^{-1} d \right)
\]

# 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

# some noise variance
N = diagonal_operator(d_space, diag=1, bare=True)  # define noise covariance
n = N.get_random_field(domain=d_space)  # generate noise

D = propagator(s_space, sym=True, imp=True, para=[S, N, R])  # define propagator
m = D(j)  # reconstruct map

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))
Wiener filtering

```python
# Some signal space: e.g. a one-dimensional regular grid
s_space = rg_space(512, ...)
# or rg_space
# or hp_space
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
R.target = d_space
N = diagonal_operator(d_space, diag=1, bare=True)  # Define noise covariance
n = N.get_random_field(domain=d_space)  # Generate noise
D = propagator(512, sym=True, imp=True, para=[S,N,R])  # Define propagator
m = D(j)
```

The NIFTY way of Bayesian signal inference

Selig et al. (2013)
The NIFTY way of Bayesian signal inference

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

class propagator(operator):
    _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([256, 256])
# or  rg_space()
# 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
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
```
Wiener filtering

Original signal

Noisy data

Reconstruction

# some signal space; e.g. a one-dimensional regular grid
s_space = hp_space(128)
# or     rg_space
# or     hp_space
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
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.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 and more
Wiener filtering and more

Selig et al. (2013)
The NIFTY way of Bayesian signal inference

- 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
The NIFTY way of Bayesian signal inference

by

Marco Selig
Applications
## The Galactic free electron density

Greiner et al. (in prep.)

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

\[
\langle d \rangle_{(d|\rho)} = \left( \int_{\text{observer}} \, d\vec{l}_j \, \rho(\vec{x}) \right)_j = R\rho
\]

---

**Theory**

**NIFTY**

**Applications**

**Summary**

1 of 67 pulsars

1 of 67 pulsars
The Galactic free electron density

Greiner et al. (in prep.)

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

\[
\langle d \rangle_{(d|\rho)} = \left( \int_{\text{observer}}^{\text{pulsar}} d\vec{l}_j \, \rho(\vec{x}) \right)_j = R\rho
\]

- **a priori** assumptions
  - electron density \( \rho \leftarrow \) 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

\[
\langle d \rangle_{(d|\rho)} = \lambda_0 \exp \left( -a \int_{\text{source}} \int_{\text{detector}} d\vec{l}_j \rho(\vec{x}) \right)_j = \lambda_0 \exp (-a R \rho)
\]

data (736 detectors x 4608 projections)
Computer tomography

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

\[
\langle d \rangle_{(d|\rho)} = \lambda_0 \exp \left( -a \int_{\text{source}}^{\text{detector}} d\vec{l}_j \rho(\vec{x}) \right) = \lambda_0 \exp \left( -a R\rho \right)
\]
Computer tomography

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

\[
\langle d \rangle_{(d|\rho)} = \lambda_0 \exp \left( -a \int_{\text{source}} d\vec{l}_j \rho(\vec{x}) \right) = \lambda_0 \exp \left( -a R \rho \right)
\]

- a priori assumptions
  - matter density \( \rho \) ← log-normal prior
  - known correlations ← medical databases
Computer tomography

\[ a = 1 \quad \text{,} \quad \lambda_0 = 10^8 \]
Computer tomography

\[ a = 1 , \quad \lambda_0 = 10^7 \]
Computer tomography

\[ a = 1 \quad , \quad \lambda_0 = 10^6 \]
Computer tomography

\[ a = 1, \quad \lambda_0 = 10^5 \]
Computer tomography

\[ a = 1, \quad \lambda_0 = 10^4 \]
Computer tomography

\[ a = 1 \quad , \quad \lambda_0 = 10^8 \]
The Fermi $\gamma$-ray sky

Selig et al. (in prep.)

$E_\gamma \sim 100 \text{ MeV} \ldots 100 \text{ GeV}$, $t_{\text{mission}} \sim \text{week 9} \ldots 220$
The Fermi $\gamma$-ray sky

Selig et al. (in prep.)

- data model
  - uneven survey coverage
  - Poissonian noise
  \[
  \langle d \rangle_{(d|\rho)} = R\rho
  \]

- a priori assumptions
  - diffuse flux $\rho \leftarrow$ 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

\[
\langle d \rangle_{(d|\rho)} = R (\rho_s + \rho_u)
\]

- \textit{a priori} assumptions
  - diffuse flux $\rho_s \leftarrow$ log-normal prior
  - known correlations
  - point source flux $\rho_u \leftarrow$ inverse-Gamma priors
The NIFTY way of Bayesian signal inference   by   Marco Selig

Theory

NIFTY

Applications

Summary
The NIFTY way of Bayesian signal inference

Theory

NIFTY

Applications

Summary
The NIFTY way of Bayesian signal inference

Theory

Applications

Summary
The NIFTY way of Bayesian signal inference

Theory

NIFTY

Applications

Summary
The NIFTY way of Bayesian signal inference

Theory

NIFTY

Applications

Summary
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
  \[ S = \sum_k p_k S_k \quad , \quad p \sim \prod_k \mathcal{I} (p_k, \alpha \to 1, q \to 0) \]

- inverse-Gamma prior
  \[ \mathcal{I} (p_k, \alpha, q) \propto p_k^{-\alpha} \exp \left( -\frac{q}{p_k} \right) \]
Spectral smoothness prior
Oppermann et al. (2012)

- unknown signal correlations

\[ S = \sum_k p_k S_k \quad , \quad p \sim P_{sm}(p) \prod_k I(p_k, \alpha \to 1, q \to 0) \]

- inverse-Gamma prior and spectral smoothness prior

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

\[
\sigma^2 = 1000 \\
\sigma^2 = 10
\]