CUQIpy

Computational **U**ncertainty **Q**uantification for **I**nverse problems in **py**thon

Joint work with:
Babak Afkham
Silja L. Christensen
Felipe Uribe
Per Christian Hansen
and CUQI

Nicolai Riis – DTU
Amal Alghamdi – DTU
Jakob Sauer Jørgensen – DTU

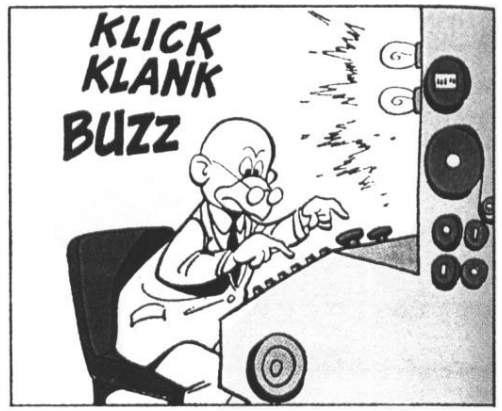
IUQ Workshop
Helsingør – Denmark – 27 September 2022

Pronounced:

CUQIpy in a Nutshell

Vision

Build a software package that uses uncertainty quantification (UQ)
to access and quantify uncertainties in solutions to inverse problems.



- **Simplify** the mathematics, statistics and code for the non-expert user.
- Provide **full control** for expert users.
- Allow users to focus on **modeling aspects**.

Features

- Easy access to **state-of-the-art** tools in one framework (including 3rd party libraries).
- A suite of **test problems** to allow users to get started.
- Allow users to provide **custom code** for models, distributions, samplers etc.
- Exploit structure to support **large-scale** problems.

Why not use an existing software package?

General UQ software:

- Tends to break down for large-scale problems.

Software for UQ in inverse problems:

- Often specialized for certain types of problems.

The niche that CUQIpy is aimed at:

- Unified interface for broad range of problems.
- Simple “non-expert” interface.
- Test problem suite.
- Linking to other software libraries.
- Support user-defined code.

Collaborative development

Developed on GitLab/GitHub since 2020

- Sept 2022 early version public
- github.com/CUQI-DTU/CUQIpy

Core team

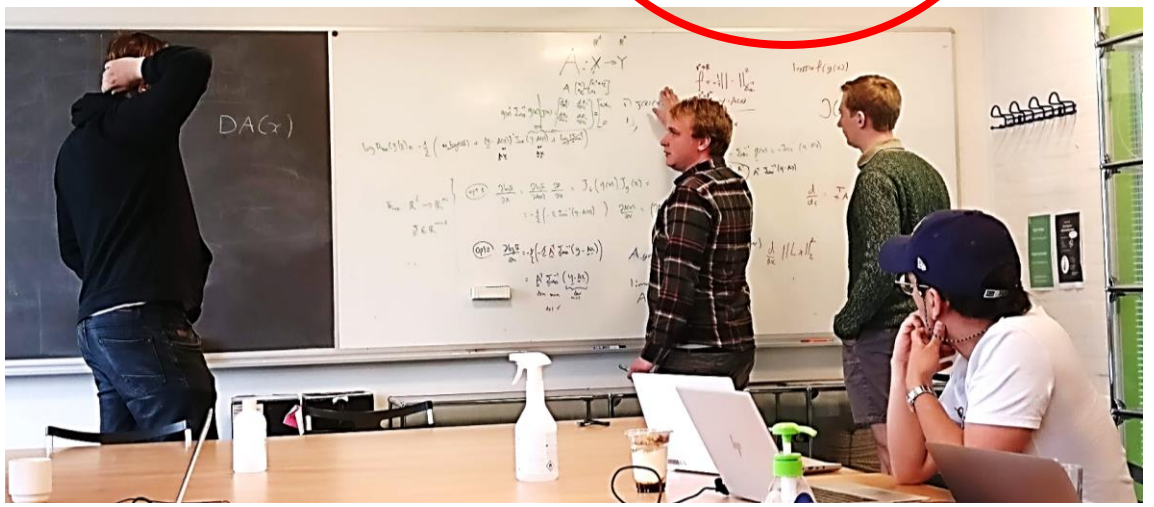
- Nicolai, Amal and myself
- New postdoc position opening soon!

Involvement of CUQI team

- Problem and feature requests
- Code contributions

Hackathons

- 2-day collaborative development days
- New users, new features – vehicle for collaboration!



CUQIpy training yesterday

cuqi-dtu.github.io/CUQIpy

CUQIpy's Documentation

Computational Uncertainty Quantification for Inverse Problems in python (CUQIpy) is a python package for modeling and solving inverse problems in a Bayesian inference framework. CUQIpy provides a simple high-level interface to perform UQ analysis of inverse problems, while still allowing full control of the models and methods. The package comes equipped with a number of predefined distributions, samplers, models and test problems and is built to be easily further extended when needed.

This software package is part of the CUQI project funded by the Villum Foundation.

Quick Links: Installation | Tutorials | How-To Guides | Source Repository

Getting Started

Get CUQIpy up and running in your machine and learn the basics.

User Guide

Step-by-step tutorials, how-to guide to help you accomplish various tasks with CUQIpy, and in-depth explanations of background theory.

API Reference

Detailed descriptions of CUQIpy library components (modules, classes, methods, etc.).

Contributor's Guide

Here you find information on how to contribute to CUQIpy. All contributions are welcome, small and big!

CUQIpy is built as a Bayesian framework

Generic inverse problem:

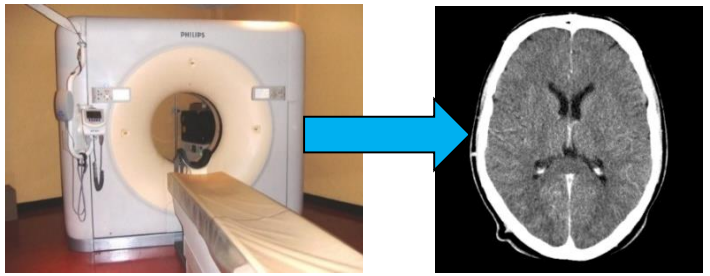
Observed data

Parameters to estimate

$$\mathbf{b} = \mathbf{A}(\mathbf{x})$$

Forward model

Classic solution: a single estimate



Bayesian inverse problem:

$$\pi(\mathbf{x} \mid \mathbf{b}) \propto \pi(\mathbf{b} \mid \mathbf{x})\pi(\mathbf{x})$$

Posterior: the probability density function (pdf) of the parameters given the observed data

Likelihood: density of the data given the parameters

Prior: (subjective) pdf of parameters before observing the data

Bayesian solution:
samples from the **posterior**:
a range of possible estimates!

CUQIpy is built as a Bayesian framework

Do UQ in 5 steps by <u>defining</u>:

Can be exchanged with user code or 3rd party library

1. The model
2. The prior(s)
3. The data distribution (likelihood)
4. The posterior sampling
5. The posterior analysis

For example:

$$\mathbf{A} : \mathbf{x} \mapsto \mathbf{b} \text{ (X-ray CT model)}$$

$$\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \delta^2 \mathbf{I})$$

$$\mathbf{b} \mid \mathbf{x} \sim \mathcal{N}(\mathbf{A}\mathbf{x}, \sigma^2 \mathbf{I}) \text{ with observed data } \mathbf{b}^{obs}$$

$$\mathbf{x}^{s+1} = \mathrm{MCMC}_{\pi(\mathbf{x}|\mathbf{b})}(\mathbf{x}^s)$$

- Metropolis-Hastings
- NUTS
- …

MAP, sample mean, CI, etc..

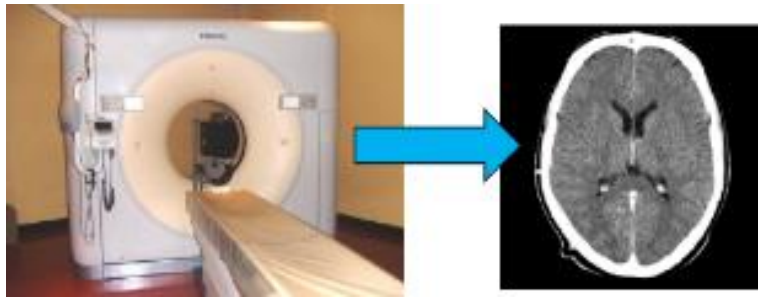
This provides a unified framework for any kind of inverse problem!

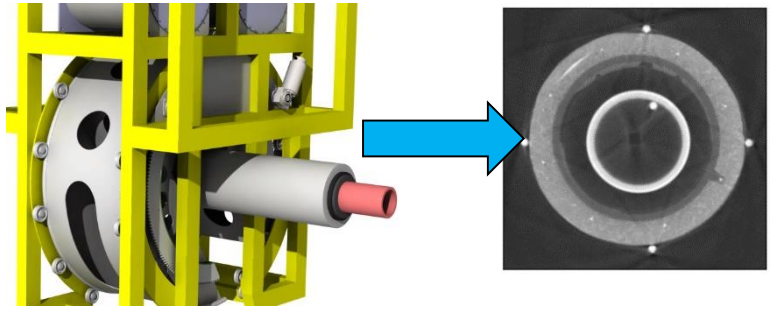
Handling two very different inverse problems

1: Linear inverse problems

$$\mathbf{b} = \mathbf{A}\mathbf{x}$$

Application: **CT (Ray model)**





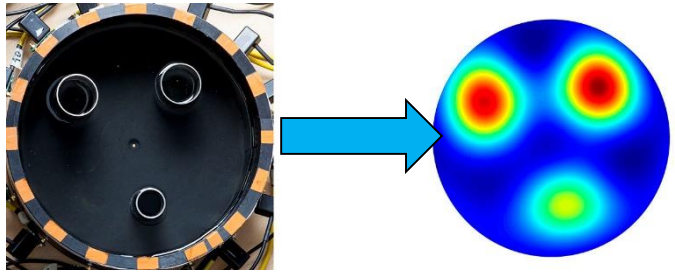
Courtesy of Force Technology

Christensen et. al. 2022

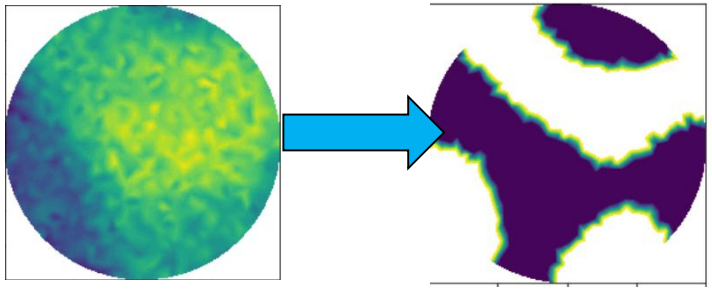
2: PDE-based inverse problems

$$\mathbf{b} = \mathcal{G}(\boldsymbol{\theta})$$

Application**: EIT (Steady-state diffusion)**



Hauptman et. al. Open 2D Electrical Impedance Tomography data archive. ***fips.fi***



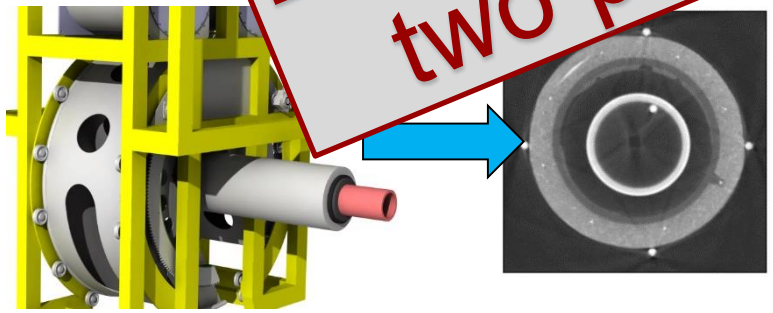
Stochastic Elliptic PDE ***Unpublished work***

Handling two very different inverse problems

1: Linear inverse problems

$$\mathbf{b} = \mathbf{A}\mathbf{x}$$

Application: **CT (Ray model)**



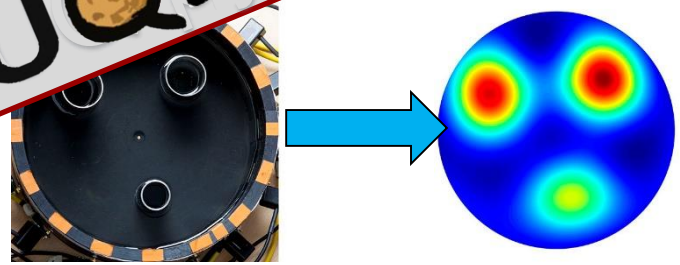
Courtesy of Force Technology

Christensen et. al. 2022

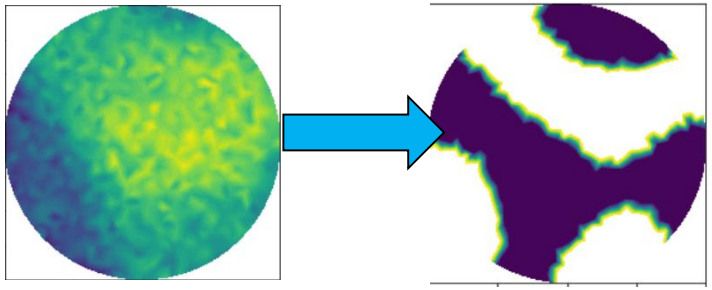
2: PDE-based inverse problems

$$\mathbf{b}$$

(Steady-state diffusion)



Hauptman et. al. Open 2D Electrical Impedance Tomography data archive. ***fips.fi***



Stochastic Elliptic PDE ***Unpublished work***

Lets see how to handle these two problems in CUQIpy

Step 1: Define the model (CT)

```
from cuqipy_cil.testproblem
import ParallelBeam2DProblem
```

Mathematical model

$$\mathbf{b}_i = (\mathbf{A}\mathbf{x})_i \approx \int_{\ell_i(\omega,s)} f(t)\mathrm{d}t$$

$$\ell_i(\omega, s) = \left\{\omega s + \omega^{\perp} t \mid t \in \mathbb{R}\right\}.$$

Data: $\mathbf{b}$ Parameters: $\mathbf{x}$

Commonly used libraries:

- Astra Toolbox
- Tigre
- Core Imaging Library (CIL)
- …

CUQIpy code for CT model with CIL

```
model = ParallelBeam2DProblem(im_size=(512,512), phantom="shepp_logan", ...).model
```

```
print(model)
>>> CUQI LinearModel: Image2D(512,512) -> Image2D(512,360), Forward parameters: ['x']
```

Object representation of model domain and range spaces

Step 1: Define the model (Poisson 2D)

```
from cuqipy_fenics.testproblem
import Poisson2D_FEniCS
```

Mathematical model

PDE

$$-\nabla \cdot (\kappa(x)\nabla u(x)) = f(x), \quad x \in \mathcal{B}_{(0,1)},$$
$$\nabla u(x) \cdot n = 0, \quad x \in \partial\mathcal{B}_{(0,1)}$$
$$\int_{\partial\mathcal{B}_{(0,1)}} u(x) = 0.$$

KL expansion

$$\kappa(x) = h\left(\sum_{i\in\mathbb{N}} \sqrt{\lambda_i}\theta_i e_i(x)\right)$$

Data: $\mathbf{b} := u|_{\mathcal{B}_{(0,1)}}$ Parameters: $\boldsymbol{\theta}$

Commonly used libraries:

- FEniCS
- Finite difference (e.g. NumPy)
- …

Poisson 2D model with FEniCS

```
model = Poisson2D_FEniCS(domain="circle", field="KL", ...).model
```

```
print(model)
```

```
>>> CUQI PDEModel: FEniCSKL(128,) -> FEniCSContinuous(834,). Forward parameters: ['theta'].
    PDE: SteadyStateLinearFEniCSPDE
```

KL expansion on FEniCS mesh

Custom user-defined models

```
from cuqi.model
import LinearModel

from cuqi.geometry
import Image2d
```

Using a custom linear model

```
model = LinearModel(forward_func,
                    adjoint_func,
                    range=Image2d((512,360)),
                    domain=Image2d((512,512)))
```

```
print(model)
```

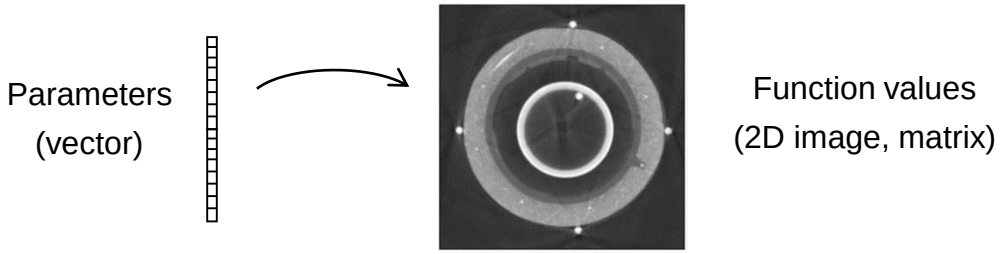
```
>>> CUQI LinearModel: Image2D(512,512) -> Image2D(512,360). Forward parameters: ['x']
```

Quick aside: CUQIpy geometries

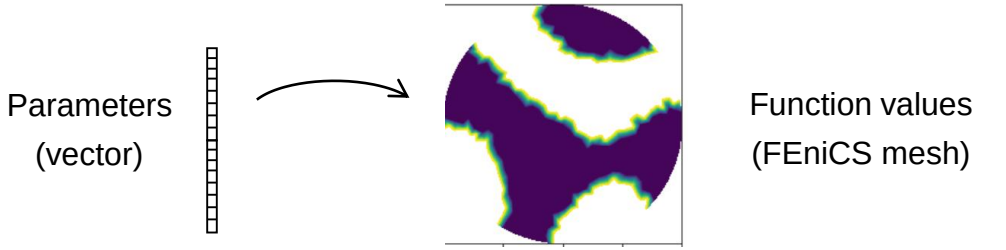
The `Geometry` object represents the spaces of the model domain and range

- Maps parameters to model input (function values).
- Contains tools for visualization.
- Provides an interface to connect to 3rd party libraries.

`Image2D:` $\mathbf{x} \mapsto \mathbf{X}$



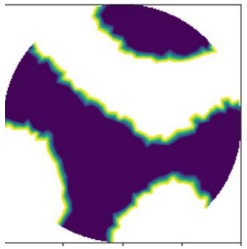
`FEniCSKL:` $\boldsymbol{\theta} \mapsto h\left(\sum_{i\in\mathbb{N}} \sqrt{\lambda_i}\theta_i e_i(x)\right)$

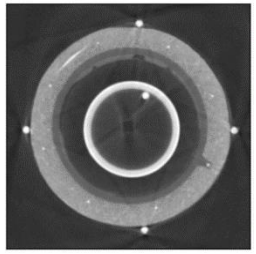


Step 2: Prior

Defining a Gaussian prior: $\mathcal{N}(\mathbf{0}, \delta^2 \mathbf{I}_n)$

```
from cuqi.distribution
import Gaussian
```



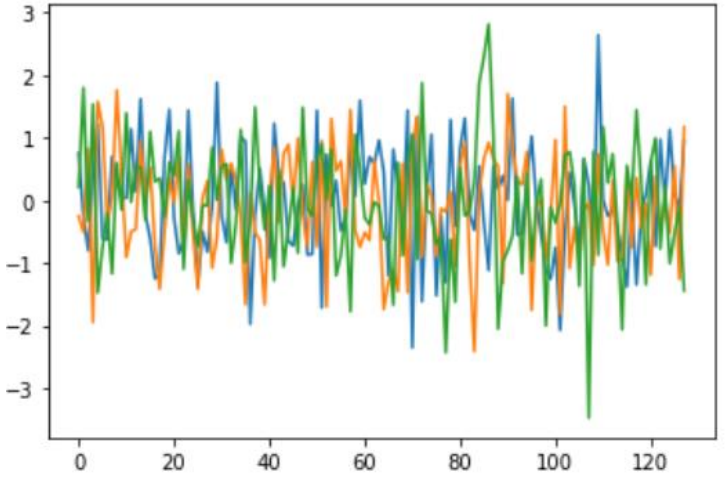


Gaussian prior

```
x = Gaussian(mean=np.zeros(n), std=delta)
```

Sampling and plotting prior

```
prior_samples = x.sample(3)
prior_samples.plot()
```



Step 2: Prior (model context)

```
from cuqi.distribution
import Gaussian
```

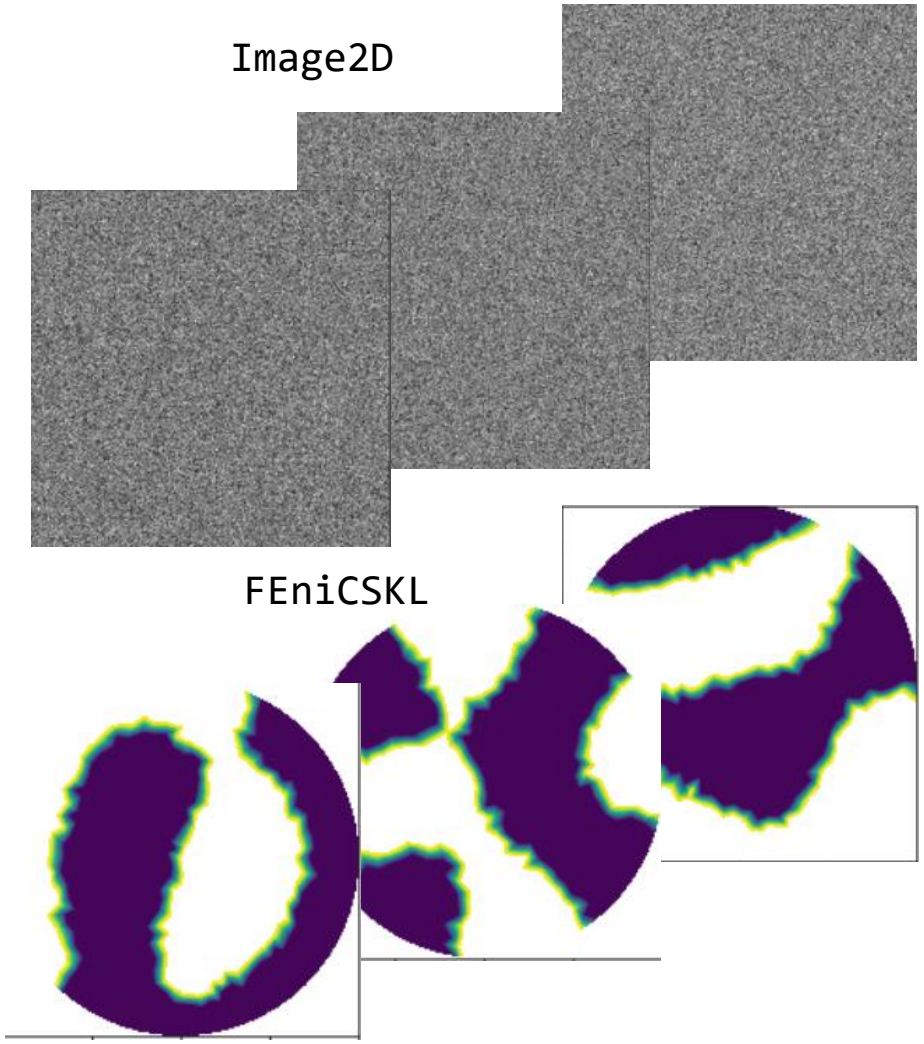
Defining a Gaussian prior: $\mathcal{N}(\mathbf{0}, \delta^2 \mathbf{I}_n)$
(now taking into account the model domain)

Gaussian prior

```
x = Gaussian(mean=np.zeros(n), std=delta,
             geometry=model.domain_geometry)
```

Sampling and plotting prior

```
prior_samples = x.sample(3)
prior_samples.plot()
```



Step 2: Prior (pre-made distributions)

```
from cuqi.distribution
import GMRF
```

Let us define a Gaussian Markov Random Field $\mathcal{N}(\mathbf{0}, \delta^2 \mathbf{L}_{2D})$

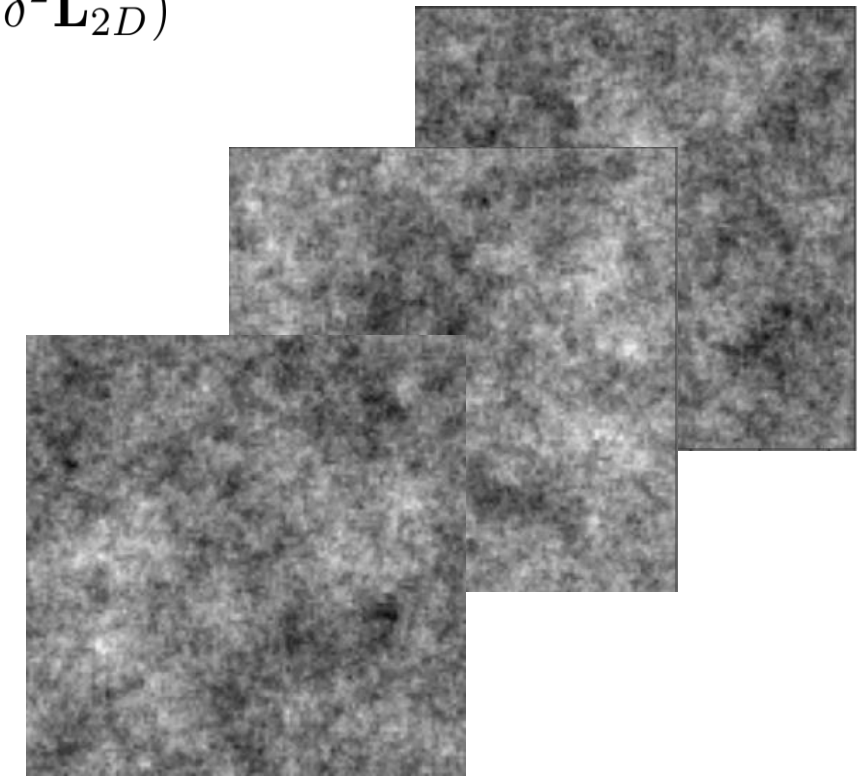
Common use-case, so pre-made:

GMRF prior – built-in geometry

```
x = GMRF(mean=np.zeros(n), std=delta,
         physical_dim=2, bc_type="zero")
```

Sampling and plotting prior

```
prior_samples = x.sample(3)

prior_samples.plot()
```



Step 3: Data distribution

```
from cuqi.distribution
import Gaussian
```

Gaussian noise model: $\mathbf{b} = \mathbf{A}(\mathbf{x}) + \mathbf{e} \quad \text{where} \quad \mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$

"Data distribution" given $\mathbf{x}$ $\pi(\mathbf{b} \mid \mathbf{x}) = \mathcal{N}(\mathbf{A}(\mathbf{x}), \sigma^2\mathbf{I})$

Data distribution

```
b = Gaussian(mean=model(x), std=sigma)
```

```
>>> CUQI Gaussian. Conditional parameters ['x'].
```

Note: At this point `x` is not a vector but a distribution (random variable)

For Poisson problem:

$\pi(\mathbf{b} \mid \boldsymbol{\theta}) = \mathcal{N}(\mathcal{G}(\boldsymbol{\theta}), \sigma^2\mathbf{I})$

```
>>> ['theta']
```

Step 4: Constructing and sampling posterior

```
from cuqi.distribution import JointDistribution

from cuqi.sampler import NUTS
```

We combine prior and data distribution in a joint distribution, construct posterior and choose sampler

Set up and sample posterior

```
# Construct joint distribution
joint  = JointDistribution(b, x)

# Construct posterior by fixing observed data
posterior  = joint(b=b_obs)

# Sample posterior
samples = NUTS(posterior).sample(200)
```

Under the hood:
Given observation $\mathbf{b}^{obs}$
define **likelihood function:**

$$L(\mathbf{x} \mid \mathbf{b}^{obs}) := \pi(\mathbf{b}^{obs} \mid \mathbf{x})$$

- For the CT problem we used `Linear_RTO`
- For the Poisson problem we used `pCN`

Automatic sampler selection being developed:
`BayesianProblem.UQ()`

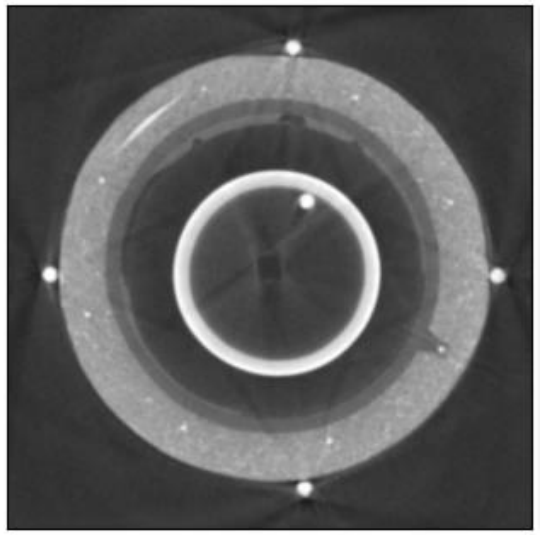
Step 5: Analyzing posterior

After sampling has finished we analyze the posterior `samples`

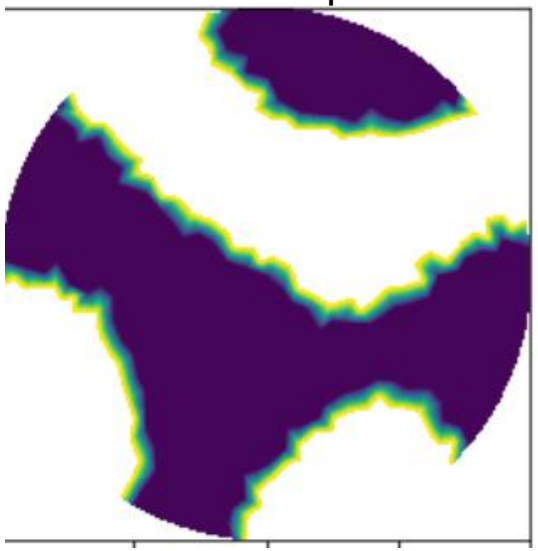
Plotting samples

```
# plot mean
samples.plot_mean()

# traceplot
samples.plot_trace()

# scatter plot
samples.plot_pair(marginals=True)
```

Posterior mean (CT)
GMRF prior



Posterior mean (Poisson)
Gaussian prior



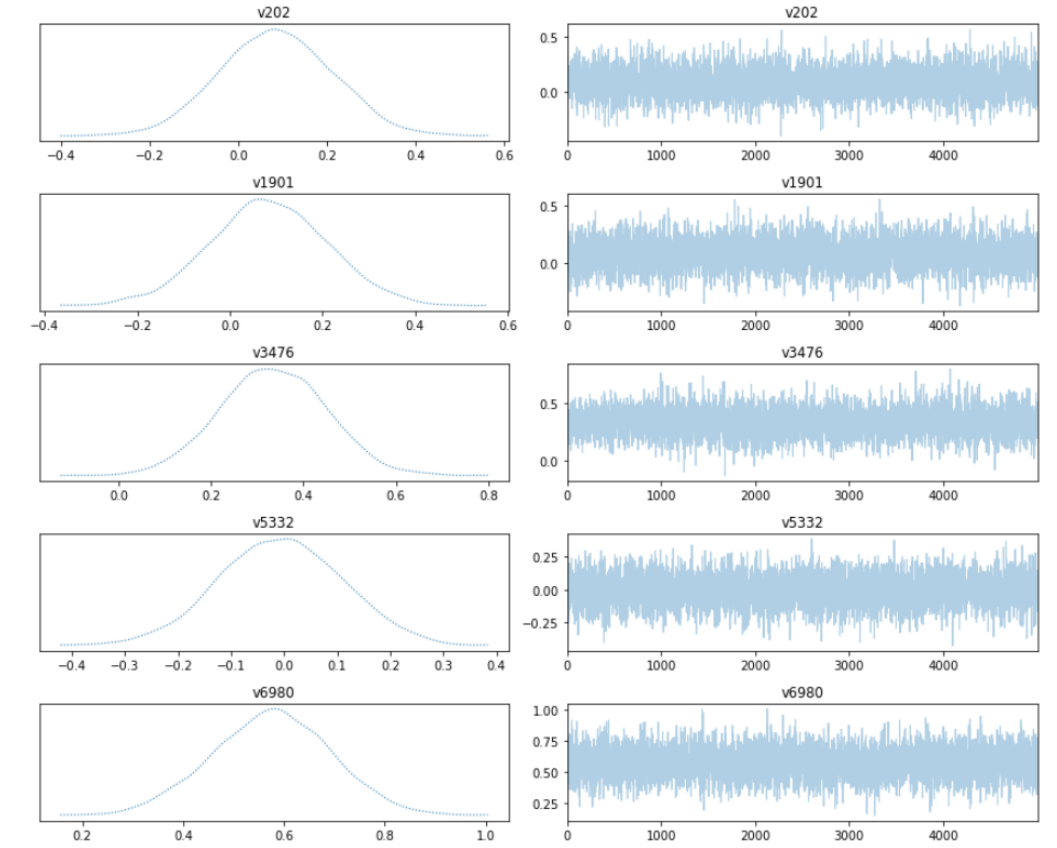
Step 5: Analyzing posterior

After sampling has finished we analyze the posterior `samples`

Plotting samples

```
# plot mean
samples.plot_mean()

# traceplot
samples.plot_trace()

# scatter plot
samples.plot_pair(marginals=True)
```

Traceplots for 5 variables (CT)



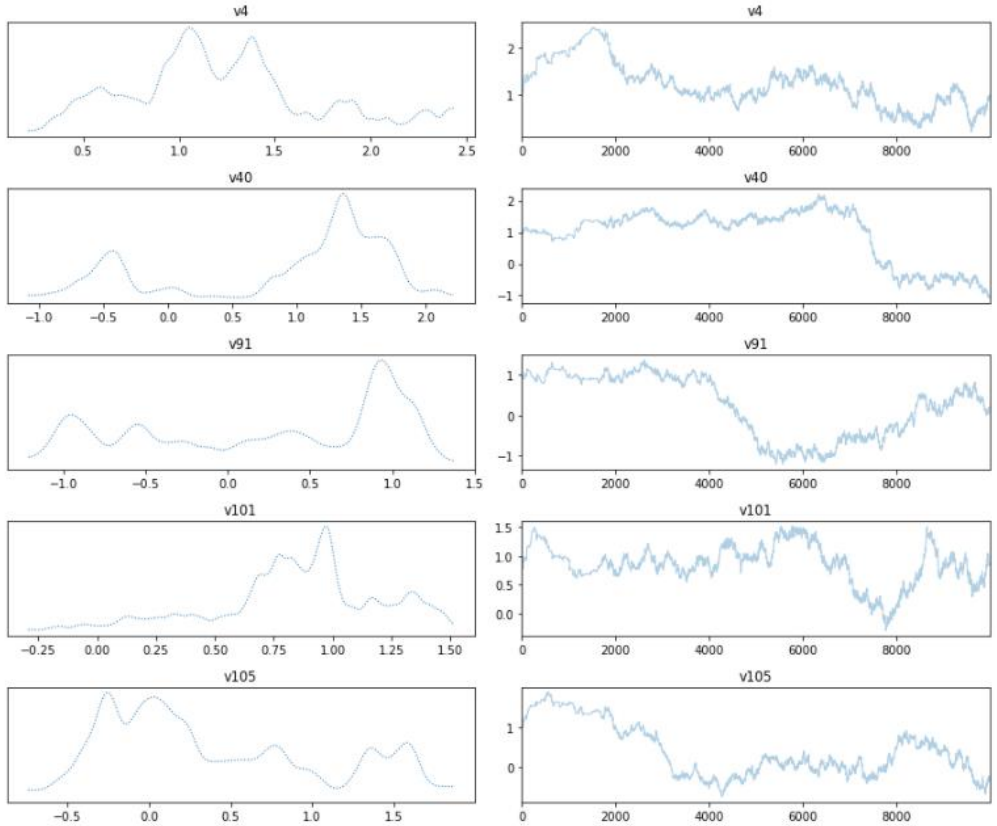
Step 5: Analyzing posterior

After sampling has finished we analyze the posterior `samples`

Plotting samples

```
# plot mean
samples.plot_mean()

# traceplot
samples.plot_trace()

# scatter plot
samples.plot_pair(marginals=True)
```

Traceplot for 5 variables (Poisson KL)



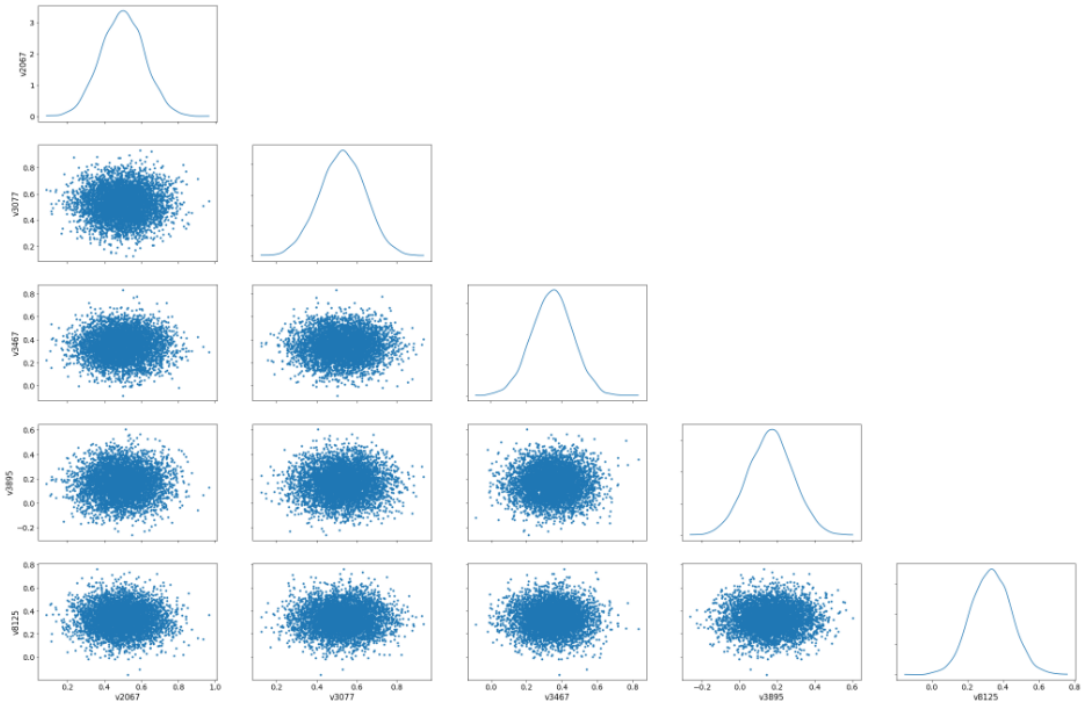
Step 5: Analyzing posterior

After sampling has finished we analyze the posterior `samples`

Plotting samples

```
# plot mean
samples.plot_mean()

# traceplot
samples.plot_trace()


# scatter plot
samples.plot_pair(marginals=True)
```

Scatterplots for 5 variables (CT)



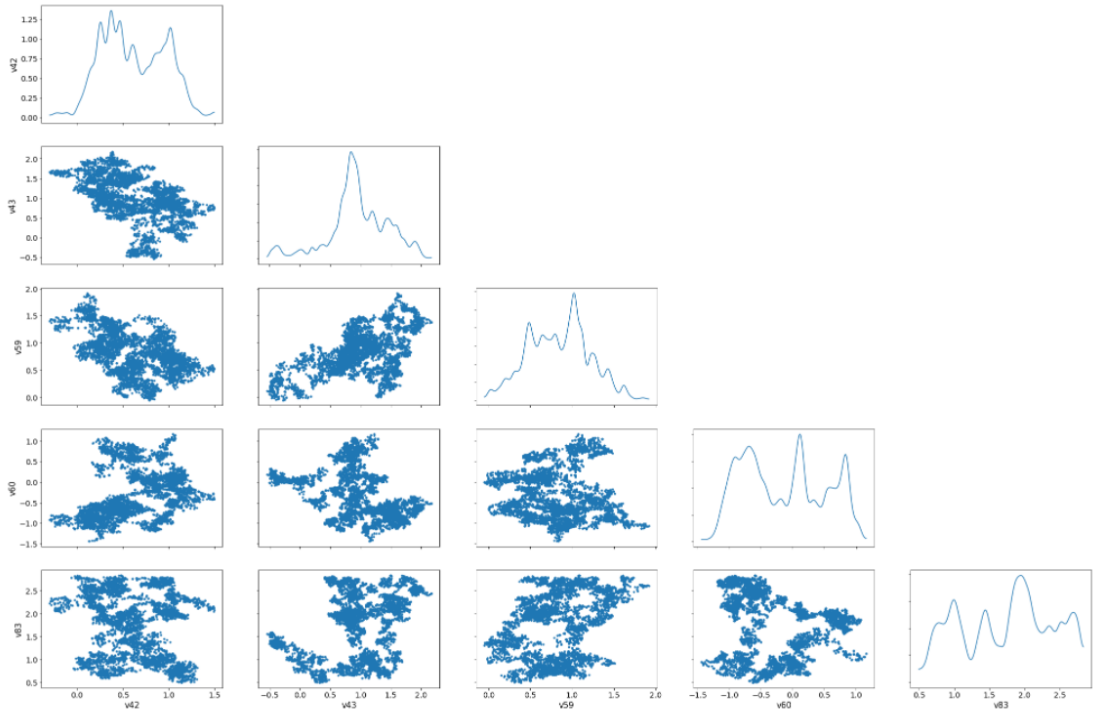
Step 5: Analyzing posterior

After sampling has finished we analyze the posterior `samples`

Plotting samples

```
# plot mean
samples.plot_mean()

# traceplot
samples.plot_trace()


# scatter plot
samples.plot_pair(marginals=True)
```

Scatterplot for 5 variables (Poisson KL)



Putting it all together – UQ in 5 steps

```
from cuqi.cil.testproblem
import ParallelBeam2DProblem

from cuqi.fenics.testproblem
import Poisson2D_FEniCS

from cuqi.distribution
import Gaussian, GMRF,
JointDistribution

from cuqi.sampler
import Linear_RTO, pCN
```

CT

```
# Step 1: Model
A = ParallelBeam2DProblem(im_size=(N,N), ...).model

# Step 2: Prior
x = GMRF(mean=np.zeros(n), std=delta,
         physical_dim=2, bc_type="zero")

# Step 3: Data distribution / Likelihood
y = Gaussian(mean=A@x, std=sigma)

# Step 4: Posterior sampling
posterior = JointDistribution(y, x)(y=data)
samples = Linear_RTO(posterior).sample(5000)

# Step 5: Analysis
samples.plot_trace()
```

Poisson

```
# Step 1: Model
A = Poisson2D_FEniCS(domain="circle", field="KL", ...).model

# Step 2: Prior
x = Gaussian(mean=np.zeros(n), std=delta)


# Step 3: Data distribution / Likelihood
y = Gaussian(mean=A(x), std=sigma)

# Step 4: Posterior sampling
posterior = JointDistribution(y, x)(y=data)
samples = pCN(posterior).sample(5000)

# Step 5: Analysis
samples.plot_trace()
```

Putting it al

Prior and noise parameters assumed known!

$$\pi(\mathbf{x} \mid \mathbf{b}) \propto \pi(\mathbf{b} \mid \mathbf{x})\pi(\mathbf{x})$$

CT

```
# Step 1: Model
A = ParallelBeam2DProblem(im_size=(N,N), ...).model

# Step 2: Prior
x = GMRF(mean=np.zeros(n), std=delta,
         physical_dim=2, bc_type="zero")

# Step 3: Data distribution / Likelihood
y = Gaussian(mean=A@x, std=sigma)

# Step 4: Posterior sampling
posterior = JointDistribution(y, x)(y=data)
samples = Linear_RTO(posterior).sample(5000)

# Step 5: Analysis
samples.plot_trace()
```

Poisson

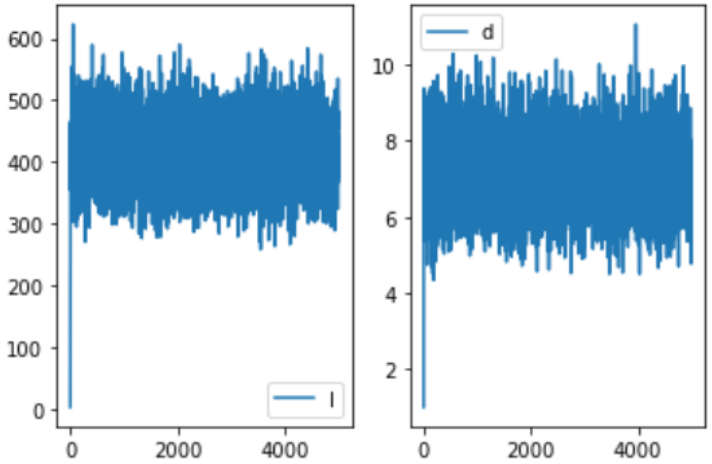
```
# Step 1: Model
A = Poisson2D_FEniCS(domain="circle", field="KL", ...).model

# Step 2: Prior
x = Gaussian(mean=np.zeros(n), std=delta)


# Step 3: Data distribution / Likelihood
y = Gaussian(mean=A(x), std=sigma)

# Step 4: Posterior sampling
posterior = JointDistribution(y, x)(y=data)
samples = pCN(posterior).sample(5000)

# Step 5: Analysis
samples.plot_trace()
```

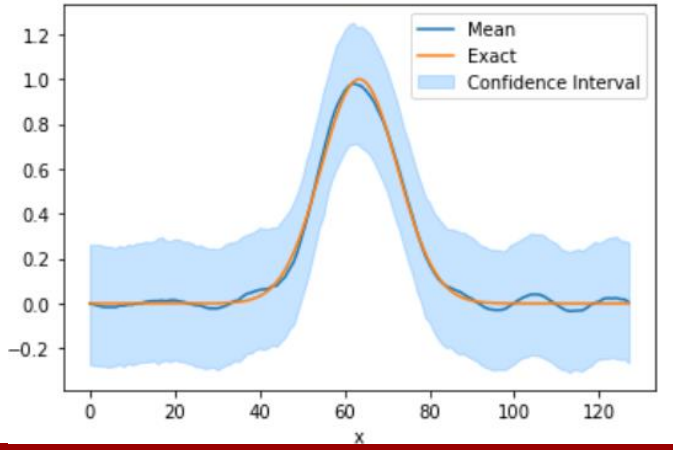
$$\pi(\mathbf{x}, \lambda, \delta \mid \mathbf{b}) \propto \pi(\mathbf{b} \mid \mathbf{x}, \lambda)\pi(\mathbf{x} \mid \delta)\pi(\delta)\pi(\lambda)$$

Gibbs sampling

```
# Step 1: Model
A = Deconvolution1D(dim=n, phantom="gauss", ...).model

# Step 2: Prior
d = Gamma(shape=1, rate=1e-4)
l = Gamma(shape=1, rate=1e-4)
x = Gaussian(mean=np.zeros(n), std=lambda d: 1/d)

# Step 3: Data distribution / Likelihood
y = Gaussian(mean=A@x, std=lambda l: 1/l)

# Step 4: Specify observation in target distribution and specific samplers for Gibbs
posterior = JointDistribution(y, x, d, l)(y=data)
samples = Gibbs(posterior,
                samplers={"x":LinearRTO, ("d", "l"):Conjugate}).sample(5000)

# Step 5: Analysis
samples["l"].plot_chain(), samples["d"].plot_chain()
samples["x"].plot_ci(exact=gt)
```

```
from cuqi.testproblem
import Deconvolution1D

from cuqi.distribution import
Gaussian, Gamma, JointDistribution

from cuqi.sampler
import Gibbs, NUTS, MH, Conjugate
```

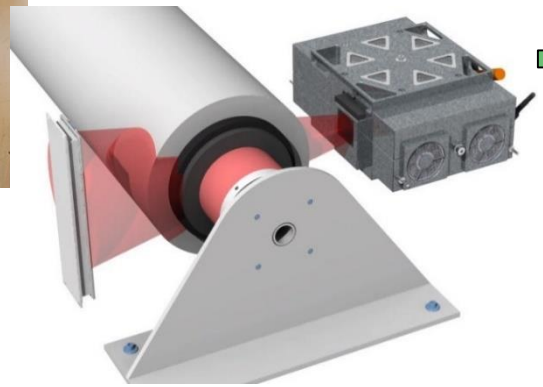




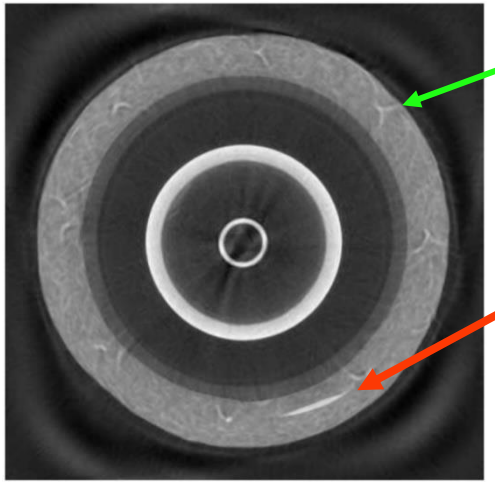
Case: X-Ray for Inspection

X-ray scanning is used to detect *defects*, *cracks*, etc. in an oil/gas pipe.

Pipe



Scanner



CT image

Reinforcing bars are ok.

Is this a defect or an artifact?!

We need tools to make a decision.

Defect Detection in Bayesian CT Imaging of Subsea Pipes

Silja L. Christensen[1] (swech@dtu.dk)

Nicolai A. B. Riis[1],
Marcelo Pereyra[2,3], and
Jakob S. Jørgensen[1]

METHODOLOGY

CT forward problem

$$\mathbf{d} = \mathbf{A}(\mathbf{x} + \boldsymbol{\varepsilon}) + \mathbf{e}$$

$\mathbf{d} \in \mathbb{R}^m$: observed projections
$\mathbf{e} \in \mathbb{R}^m$: data noise with variance λ^{-1}
$\mathbf{x} \in \mathbb{R}^n$: pixel representation of the pipe
$\boldsymbol{\varepsilon} \in \mathbb{R}^n$: pixel representation of defects
$\mathbf{A} \in \mathbb{R}^{m \times n}$: CT forward model

Prior distributions

We use a Structural Gaussian Prior [1] to promote the pipe structure in $\mathbf{x}$:

$$\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{\mu}_{SGP}, \left(\mathbf{R}_{SGP}^T \mathbf{R}_{SGP}\right)^{-1}\right).$$

There are few defects, so we enforce sparsity in $\boldsymbol{\varepsilon}$ with a hierarchical prior:

$$\varepsilon_i | \eta_i \sim \mathcal{N}(0, \eta_i), \qquad \eta_i \sim \mathcal{IG}\left(\frac{\nu}{2}, \frac{\nu}{2}\omega^2\right),$$

where $i = 1 \ldots n$ and $\mathcal{IG}(\cdot)$ is the inverse gamma distribution. This is equivalent to the following student's t-distribution:

$$\varepsilon_i \sim t(\nu, 0, \omega),$$

therefore, lower ν leads to more sparsity, and lower ω leads to lower variance.

Bayesian inverse problem

From the priors and the likelihood

$$\mathbf{d}|\mathbf{x}, \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{A}(\mathbf{x} + \boldsymbol{\varepsilon}), \lambda^{-1}\mathbf{I}),$$

we can derive the posterior distributions from Bayes' Theorem:

$$p(\boldsymbol{\varepsilon}|\boldsymbol{d}, \mathbf{x}, \boldsymbol{\eta}) \propto p(\boldsymbol{d}|\mathbf{x}, \boldsymbol{\varepsilon}) \prod_{i=1}^n p(\varepsilon_i|\eta_i) p(\eta_i),$$
$$p(\boldsymbol{x}|\boldsymbol{d}, \boldsymbol{\varepsilon}) \propto p(\boldsymbol{d}|\mathbf{x}, \boldsymbol{\varepsilon}) p(\mathbf{x})$$

Gibbs sampling

We use CUQIpy to sample the posterior distribution. The Gibbs sampling scheme is:
Repeat:

1) One sample from $\boldsymbol{x}|\boldsymbol{d}, \boldsymbol{\varepsilon}$ using Linear RTO [2].
2) One sample from $\boldsymbol{\varepsilon}|\boldsymbol{d}, \mathbf{x}, \boldsymbol{\eta}$ using Linear RTO.
3) One sample from $\boldsymbol{\varepsilon}|\boldsymbol{\eta}$ utilizing conjugacy.

CUQIpy

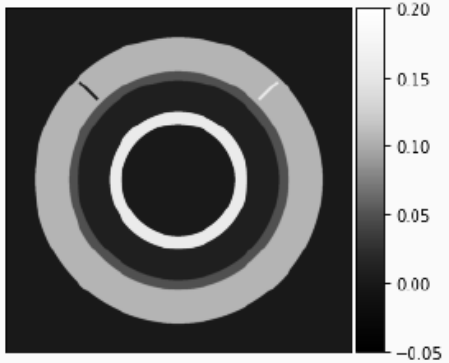
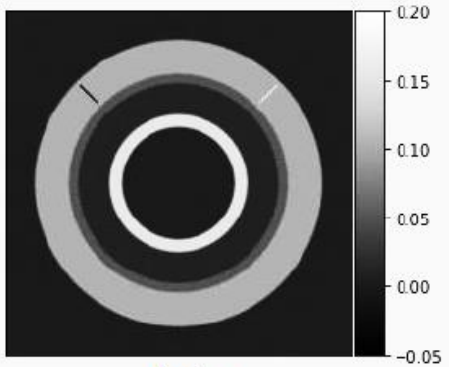
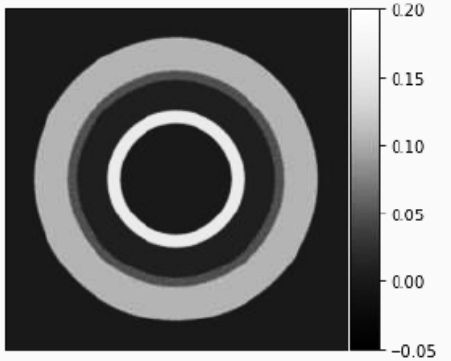


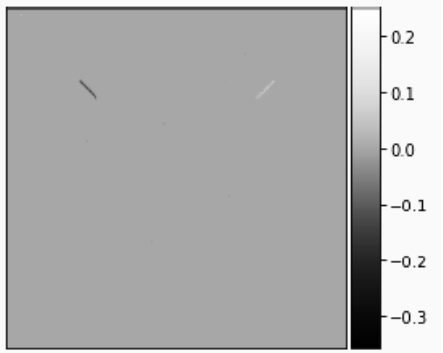
Figure 1: Phantom



c) $\mathbf{x} + \boldsymbol{\varepsilon}$



a) $\mathbf{x}$



b) $\boldsymbol{\varepsilon}$

CUQIpy

- **Install:** `pip install cuqipy`
- **Website:** cuqi-dtu.github.io/CUQIpy
- **Training notebooks:** github.com/CUQI-DTU/CUQIpy-demos
- **Plugins:**
  - CUQIpy_CIL: github.com/CUQI-DTU/CUQIpy-CIL
  - CUQIpy_FEniCS: github.com/CUQI-DTU/CUQIpy-FEniCS
  - CUQIpy_PyTorch: github.com/CUQI-DTU/CUQIpy-PyTorch
- **Publications:** Two articles in preparation – stay tuned!
- **Contact:** jakj@dtu.dk

Thanks for your attention!