Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

BrahMap: Map-making for the future CMB experiments

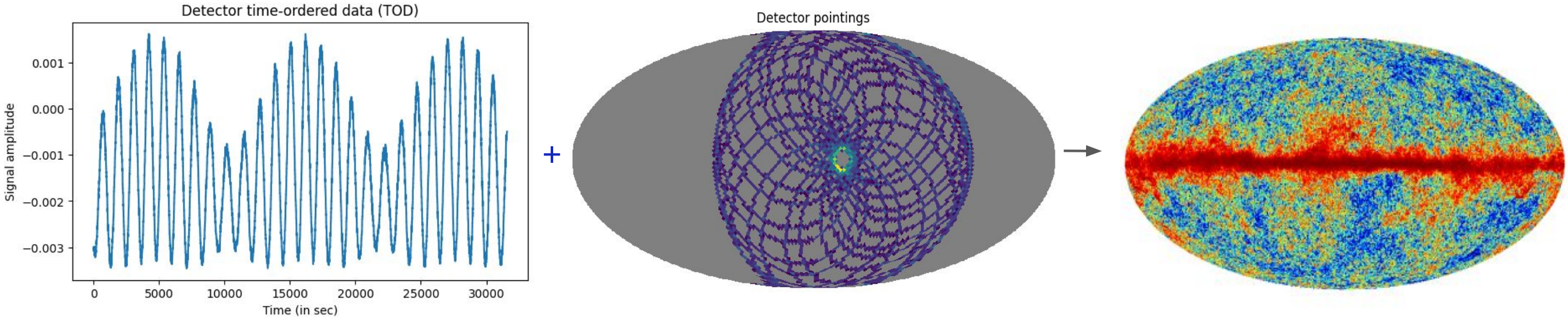
Avinash Anand[1], *Giuseppe Puglisi*[2]

[1]*University of Rome "Tor Vergata"*, [2]*University of Catania*

Spoke 3 III Technical Workshop, Perugia 26-29 Maggio, 2025

Scientific Rationale

- Future CMB experiments: Targeting the B-mode polarization of CMB
- Detectors: $O(10^3)$ - $O(10^5)$ in number with a very high sampling rate
- Data acquisition: ~250 TB (from space) to ~10 PB (from ground)
- First step of analysis: Reduction of time-series data to sky maps *aka* Map-making
- Map-making goals:
  - Reduction of enormous amount of data in a reasonable timeframe
  - Mitigation of instrumental systematics
  - Removal of both un-correlated and correlated noise



- First step of analysis: Reduction of time-series data to sky maps *aka* Map-making
- Map-making goals:
  - Reduction of enormous amount of data in a reasonable timeframe
  - Mitigation of instrumental systematics
  - Removal of both un-correlated and correlated noise

Technical Objectives, Methodologies and Solutions

Data model of the sky signal

$$d_{p,t} = I_p + Q_p \cos 2\phi_t + U_p \sin 2\phi_t + n_t$$

- $d_{p,t}$ → Signal measured by the detector
- ϕ_t → Detector polarization angle
- I_p, Q_p, U_p → CMB stokes parameters
- n_t → Un-correlated and correlated noise contribution

Data model in matrix equation form

$$d = P \cdot s + n$$

- d → Signal vector
- n → noise vector
- s → Sky map vector
- P → Pointing matrix (sparse matrix with 3 non-zero elements per row)

Minimizing the sum of square of residuals

$$S = (d - P \cdot \hat{s})^T N^{-1} (d - P \cdot \hat{s})$$

- $N \rightarrow n_t \times n_t$ Noise covariance matrix

Generalized least-square solution (GLS)

$$\hat{s} = \left(P^T N^{-1} P\right)^{-1} P^T N^{-1} d$$

Assuming the noise is stationary

Technical Objectives, Methodologies and Solutions

$$d = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n_{t-2}} \\ d_{n_{t-1}} \\ d_{n_t} \end{bmatrix} \quad P = \begin{bmatrix} \dots & \dots & \dots & \dots & \dots & 1 & \cos 2\phi_{t_1} & \sin 2\phi_{t_1} & \dots \\ \dots & 1 & \cos 2\phi_{t_2} & \sin 2\phi_{t_2} & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & 1 & \cos 2\phi_{t_3} & \sin 2\phi_{t_3} & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & 1 & \cos 2\phi_{t_{n_t-2}} & \sin 2\phi_{t_{n_t-2}} & \dots \\ \dots & 1 & \cos 2\phi_{t_{n_t-1}} & \sin 2\phi_{t_{n_t-1}} & \dots & \dots & \dots & \dots & \dots \\ \dots & 1 & \cos 2\phi_{t_{n_t}} & \sin 2\phi_{t_{n_t}} & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \quad s = \begin{bmatrix} I_1 \\ Q_1 \\ U_1 \\ \vdots \\ I_{N_p} \\ Q_{N_p} \\ U_{N_p} \end{bmatrix} \quad n = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ \vdots \\ n_{n_{t-2}} \\ n_{n_{t-1}} \\ n_{n_t} \end{bmatrix}$$

Minimizing the sum of square of residuals

$$S = (d - P \cdot \hat{s})^T N^{-1} (d - P \cdot \hat{s})$$

- $N \rightarrow n_t \times n_t$ Noise covariance matrix

Generalized least-square solution (GLS)

$$\hat{s} = \left(P^T N^{-1} P\right)^{-1} P^T N^{-1} d$$

Assuming the noise is stationary

Technical Objectives, Methodologies and Solutions

BrahMap : A scalable and modular map-making framework for future CMB experiments

Target

- Hardware-level optimization to squeeze the most of the computational resources
- Scalability across multiple CPUs and compute nodes
- Utilizing complex noise covariance models
- Offloading the computations to multiple GPUs

Features

- An intuitive Python 3 interface for linear operators used in map-making
- Interface to create new linear operators
- High performance code base with vectorization, and MPI+OpenMP based hybrid parallelization
- C++ based backend for compute intensive routines

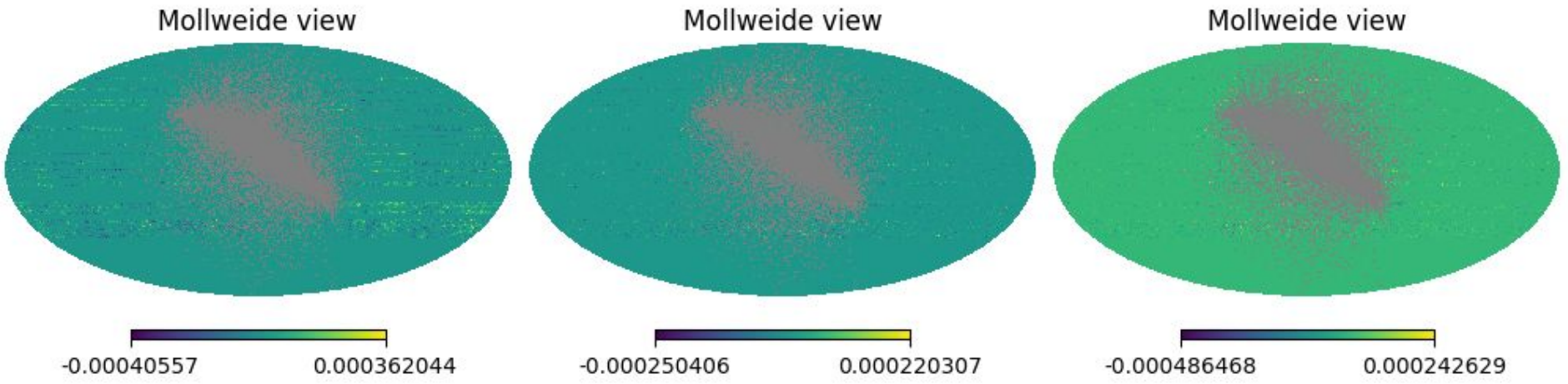
Main Results

Milestone	Target
M6	- Conversion of code base from Python 2 to Python 3 - Debugging and validation
M7	- Writing computationally extensive parts to C++ with pybind11 - **11x** performance gain over previous version
M8	- Identification of bottlenecks - Code refactoring and vectorization - **2x** performance gain over previous version
M9	- Code parallelization with MPI
M10	- OpenMP + MPI based hybrid parallelization - Implementing more realistic noise covariance operators

Main Results

OpenMP based parallelization

- Some integration tests were failing randomly
- Output maps in case of some unobserved pixels were bad sometimes
- Output maps for $N_{side} > 256$ were always corrupted



Main Results

OpenMP based parallelization

- The function `repixelize_pol_IQU()` maps the observed pixel indices to the new pixel indices

- Loop iterations are not independent - it fails when there are two or more unobserved pixels in single vector lane

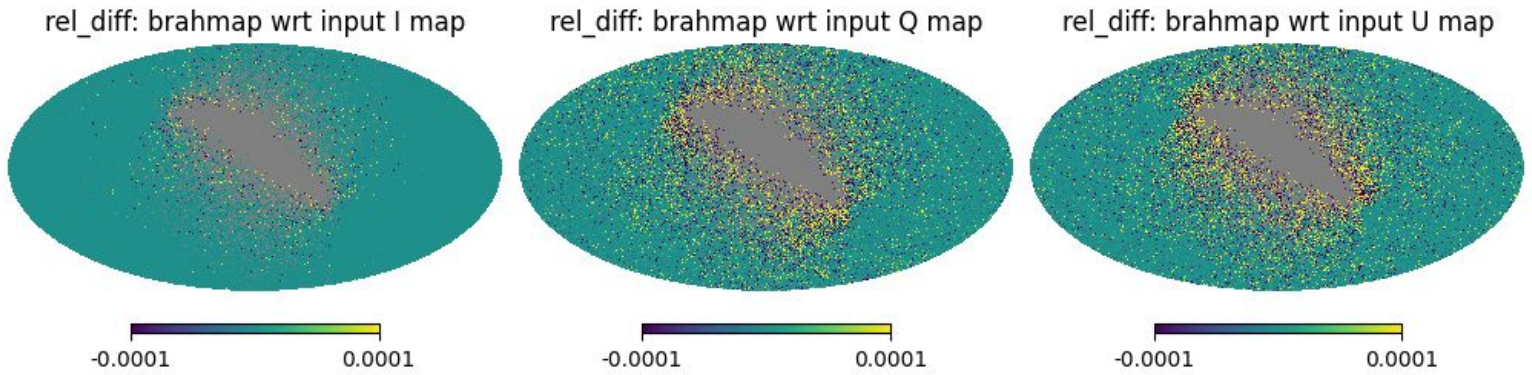
- Commenting the `pragma` directive fixes everything

```
48
49  template <typename dint, typename dfloat>
50  void repixelize_pol_IQU(                        //
51      const ssize_t new_npix,                     //
52      const dint *__restrict observed_pixels,     //
53      dfloat *__restrict weighted_counts,         //
54      dfloat *__restrict weighted_sin_sq,         //
55      dfloat *__restrict weighted_cos_sq,         //
56      dfloat *__restrict weighted_sincos,         //
57      dfloat *__restrict weighted_sin,            //
58      dfloat *__restrict weighted_cos,            //
59      dfloat *__restrict one_over_determinant     //
60  ) {
61
62  #pragma omp parallel for simd
63    for (ssize_t idx = 0; idx < new_npix; ++idx) {
64      dint pixel = observed_pixels[idx];
65      weighted_counts[idx] = weighted_counts[pixel];
66      weighted_sin_sq[idx] = weighted_sin_sq[pixel];
67      weighted_cos_sq[idx] = weighted_cos_sq[pixel];
68      weighted_sincos[idx] = weighted_sincos[pixel];
69      weighted_sin[idx] = weighted_sin[pixel];
70      weighted_cos[idx] = weighted_cos[pixel];
71      one_over_determinant[idx] = 1.0 / one_over_determinant[pixel];
72    } // for
73
74    return;
75
76  } // repixelize_pol_IQU()
```

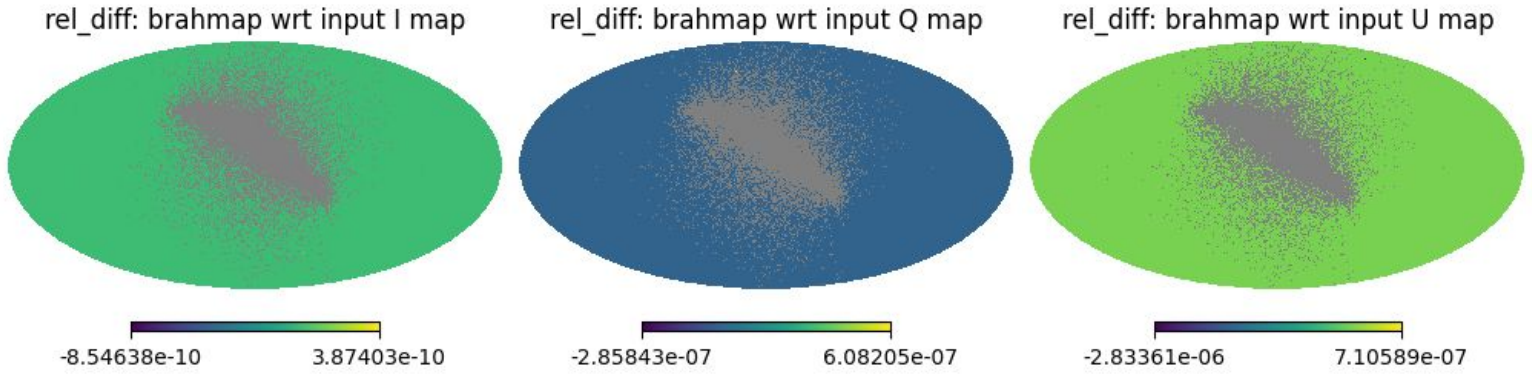
Main Results

OpenMP based parallelization

Single precision



Double precision



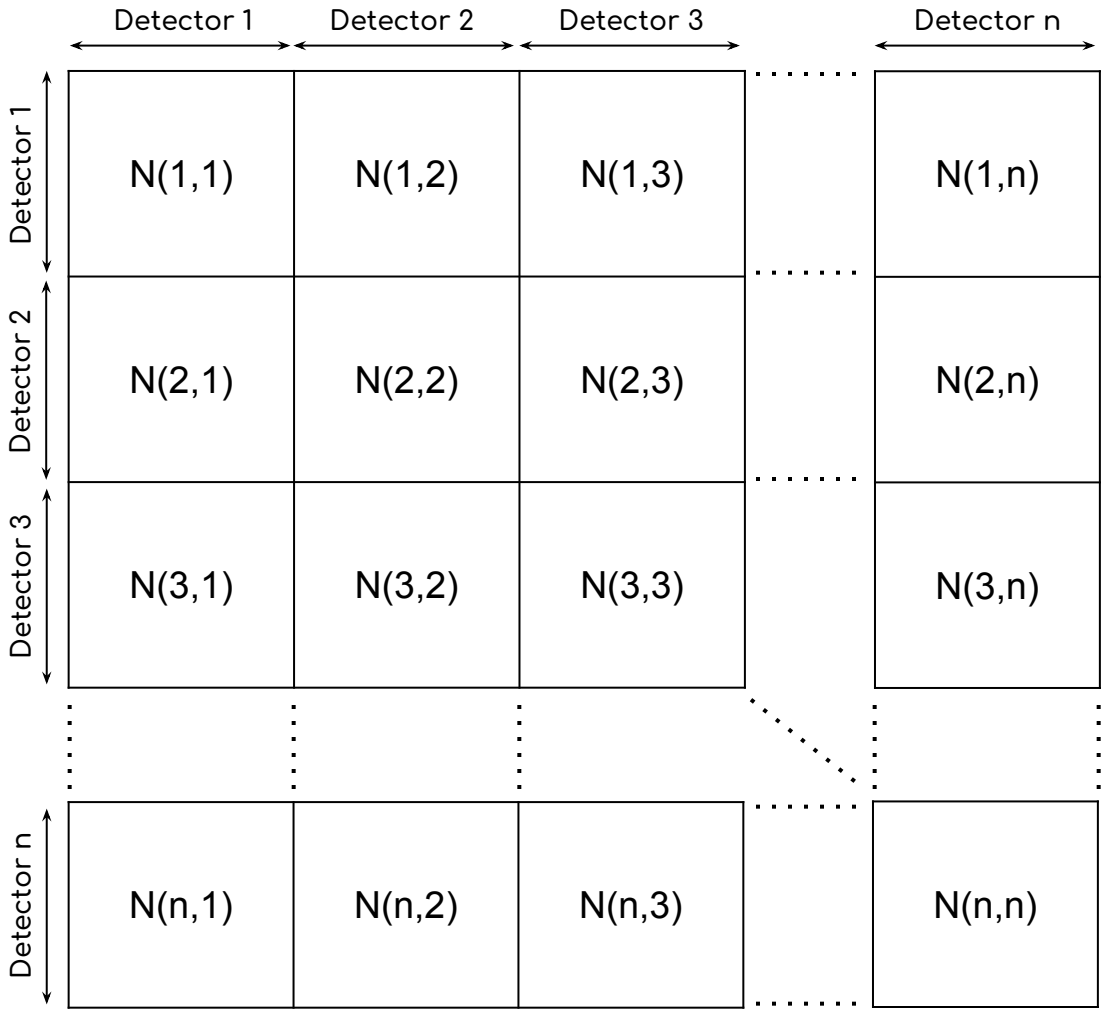
Main Results

Noise covariance operators

$$\left(P^T N^{-1} P\right) \cdot \hat{s} = P^T N^{-1} d$$

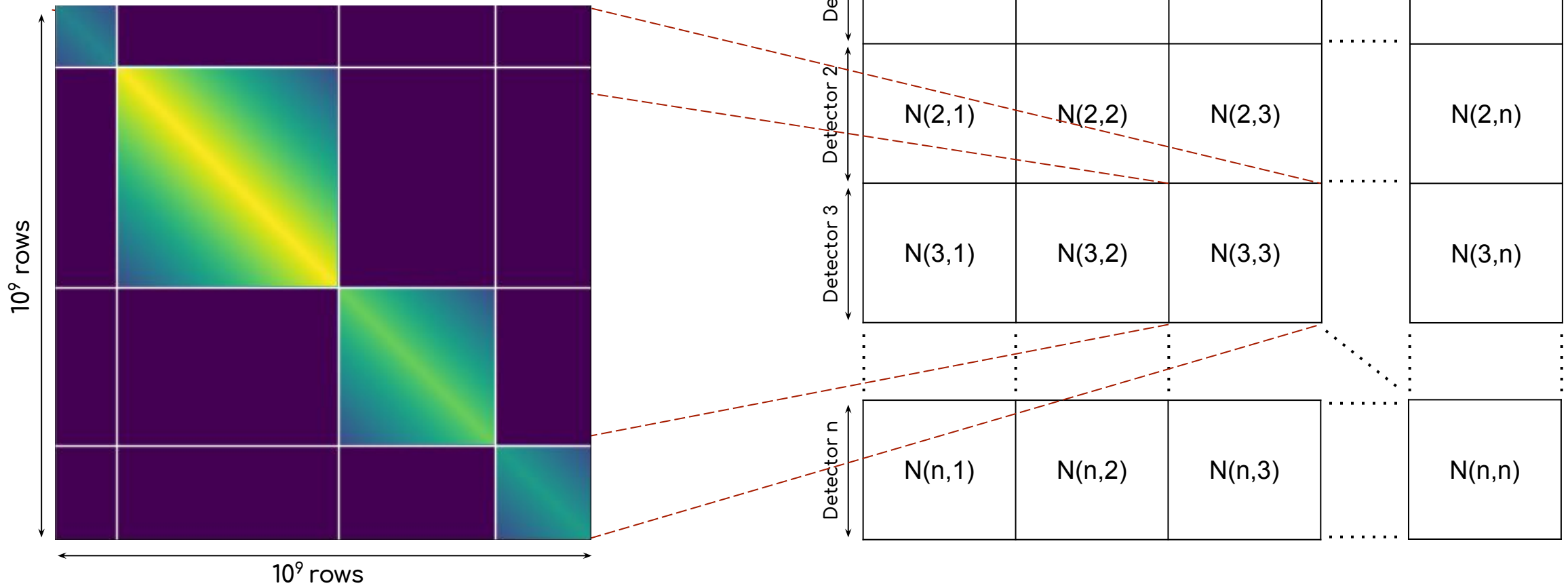
$$A \cdot x = b$$

$$A = P^T N^{-1} P \qquad x = \hat{s} \qquad b = P^T N^{-1} d$$



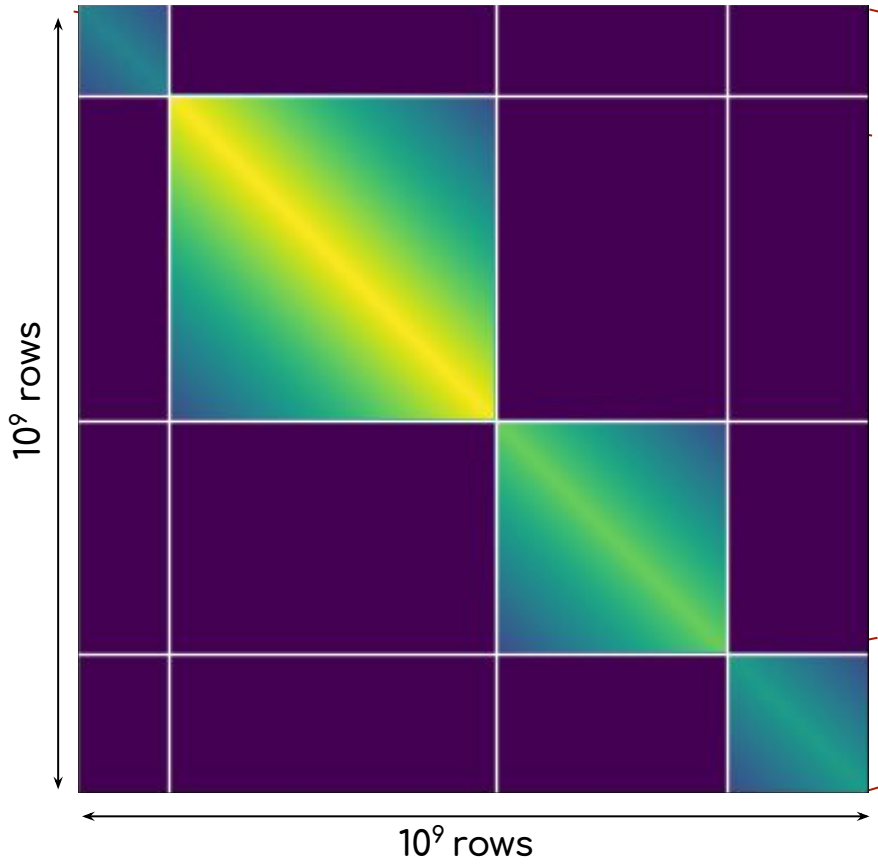
Main Results

Noise covariance operators



Main Results

Noise covariance operators



	Detector 1	Detector 2	Detector 3	Detector n
Detector 1	N(1,1)	0	0	0
Detector 2	0	N(2,2)	0	0
Detector 3	0	0	N(3,3)	0
Detector n	0	0	0	N(n,n)

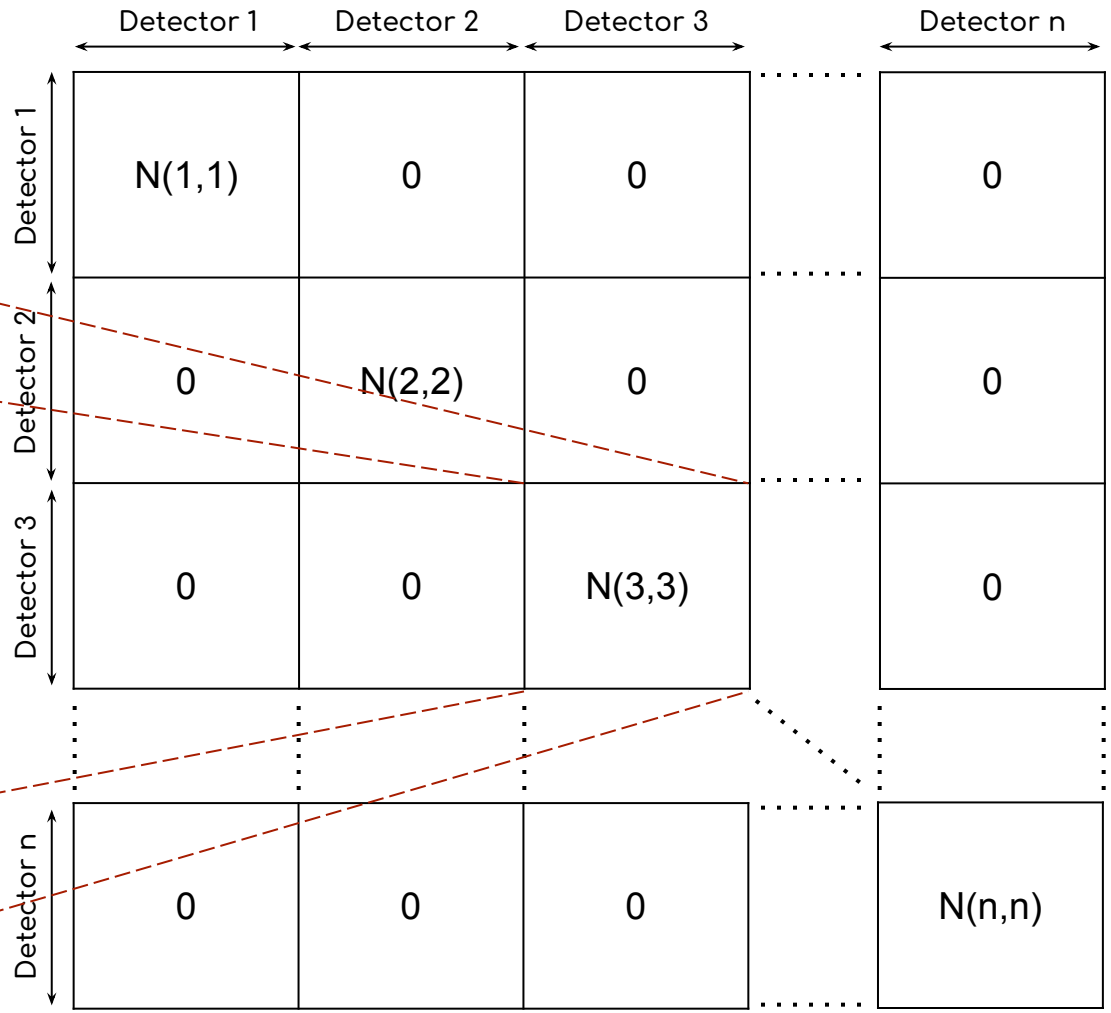
Main Results

Noise covariance operators

$$\begin{bmatrix} A_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & A_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & A_n \end{bmatrix}^{-1} = \begin{bmatrix} A_1^{-1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & A_2^{-1} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & A_n^{-1} \end{bmatrix}$$

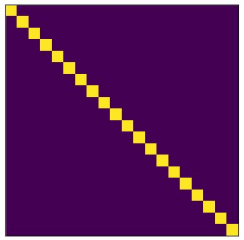
10^9 rows

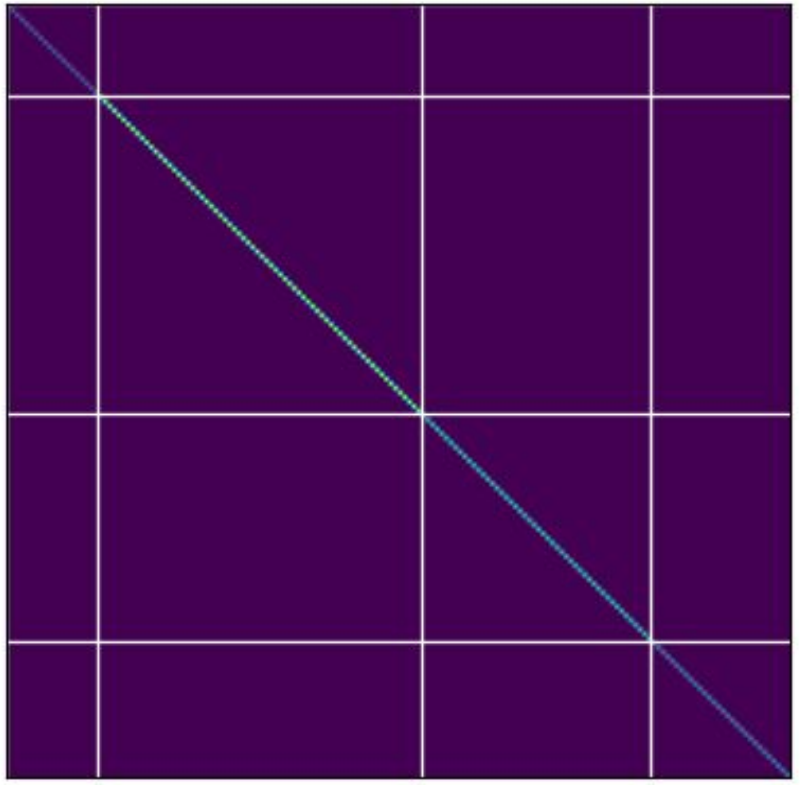
10^9 rows



Main Results

Noise covariance operators



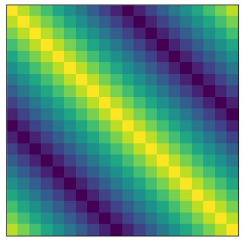


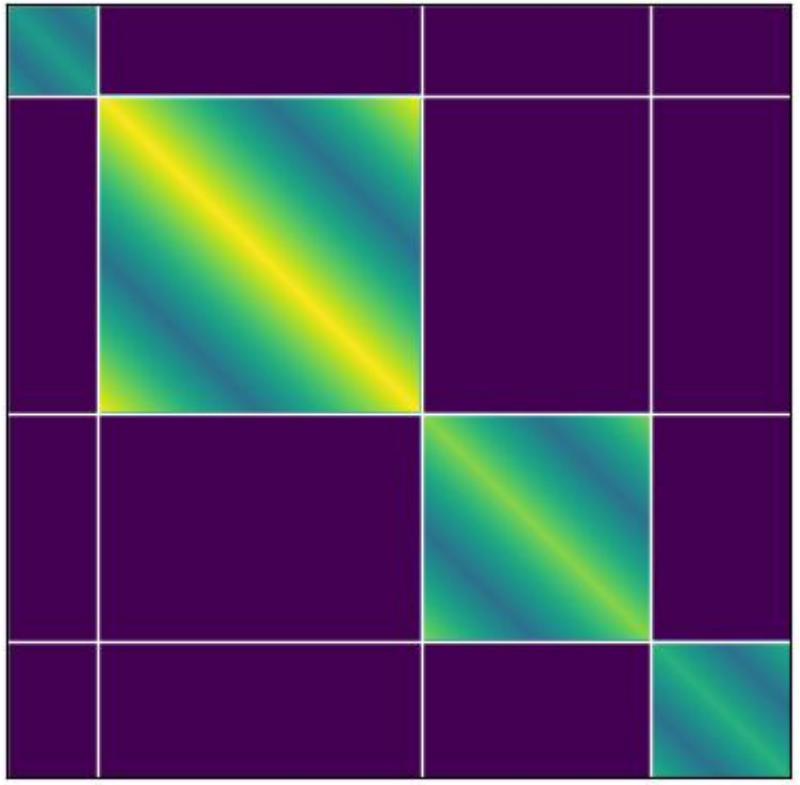
1. Noise covariance (and their inverse)
   a. Diagonal noise covariance
   b. Circulant noise covariance
   c. Band-diagonal noise covariance
   d. Toeplitz noise covariance (inverse computed through PCG)
2. Block diagonal noise covariance (incorporating the noise covariances listed above)

Main Results

Noise covariance operators

$$C = \begin{bmatrix} c_0 & c_{n-1} & c_{n-2} & \cdots & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & \ddots & & & c_2 \\ c_2 & c_1 & c_0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & c_0 & c_{n-1} & c_{n-2} \\ c_{n-2} & & & \ddots & c_1 & c_0 & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & \cdots & c_2 & c_1 & c_0 \end{bmatrix}$$



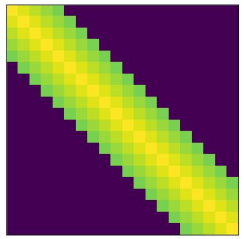


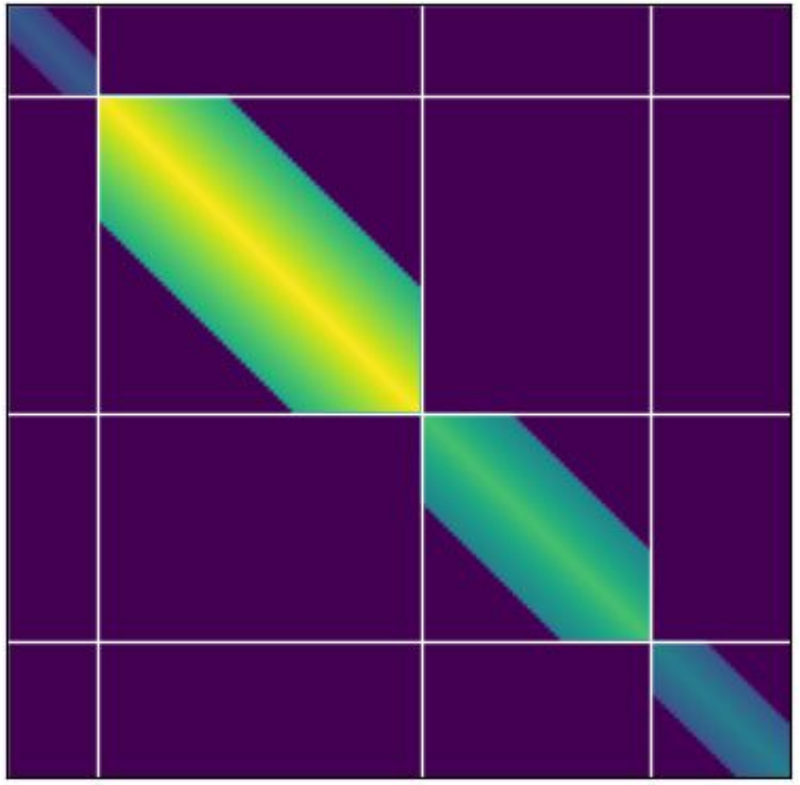
1. Noise covariance (and their inverse)
   a. Diagonal noise covariance
   b. Circulant noise covariance
   c. Band-diagonal noise covariance
   d. Toeplitz noise covariance (inverse computed through PCG)
2. Block diagonal noise covariance (incorporating the noise covariances listed above)

Main Results

Noise covariance operators

$$B = \begin{bmatrix} b_0 & b_{-1} & \dots & b_{-s} & 0 & \dots & 0 \\ b_1 & b_0 & b_{-1} & & \ddots & \ddots & \vdots \\ \vdots & b_1 & b_0 & \ddots & \ddots & \ddots & 0 \\ b_r & & \ddots & \ddots & \ddots & & b_{-s} \\ 0 & \ddots & \ddots & \ddots & b_0 & b_{-1} & \vdots \\ \vdots & \ddots & \ddots & & b_1 & b_0 & b_{-1} \\ 0 & \dots & 0 & b_r & \dots & b_1 & b_0 \end{bmatrix}$$



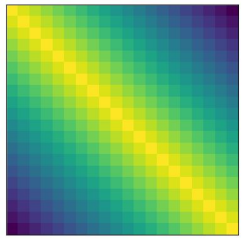


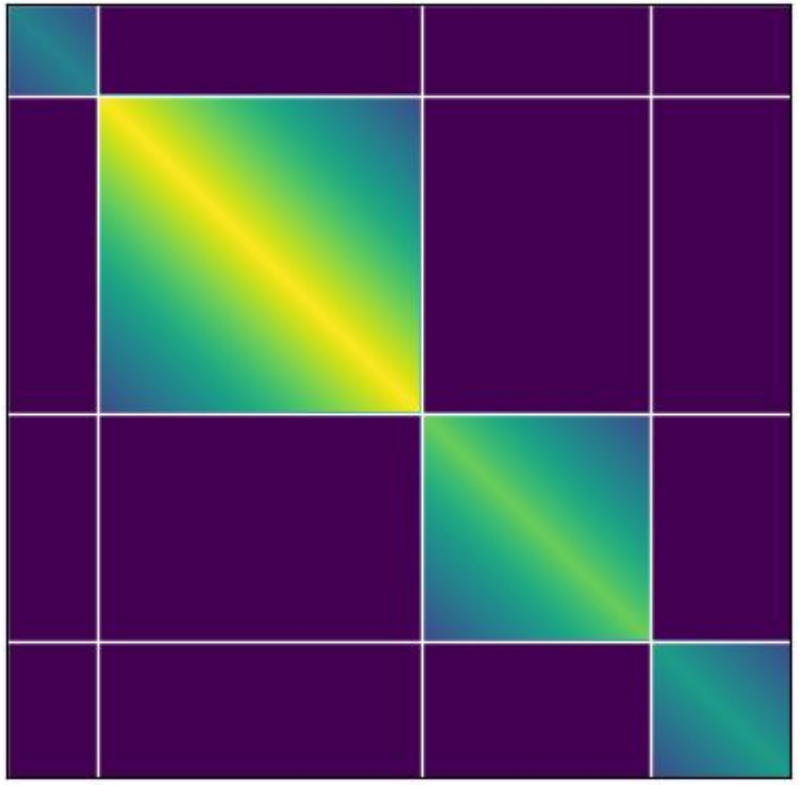
1. Noise covariance (and their inverse)
   a. Diagonal noise covariance
   b. Circulant noise covariance
   c. Band-diagonal noise covariance
   d. Toeplitz noise covariance (inverse computed through PCG)
2. Block diagonal noise covariance (incorporating the noise covariances listed above)

Main Results

Noise covariance operators

$$T = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & \cdots & \cdots & t_{-(n-2)} & t_{-(n-1)} \\ t_1 & t_0 & t_{-1} & \ddots & & & t_{-(n-2)} \\ t_2 & t_1 & t_0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & t_0 & t_{-1} & t_{-2} \\ t_{n-2} & & & \ddots & t_1 & t_0 & t_{-1} \\ t_{n-1} & t_{n-2} & \cdots & \cdots & t_2 & t_1 & t_0 \end{bmatrix}$$





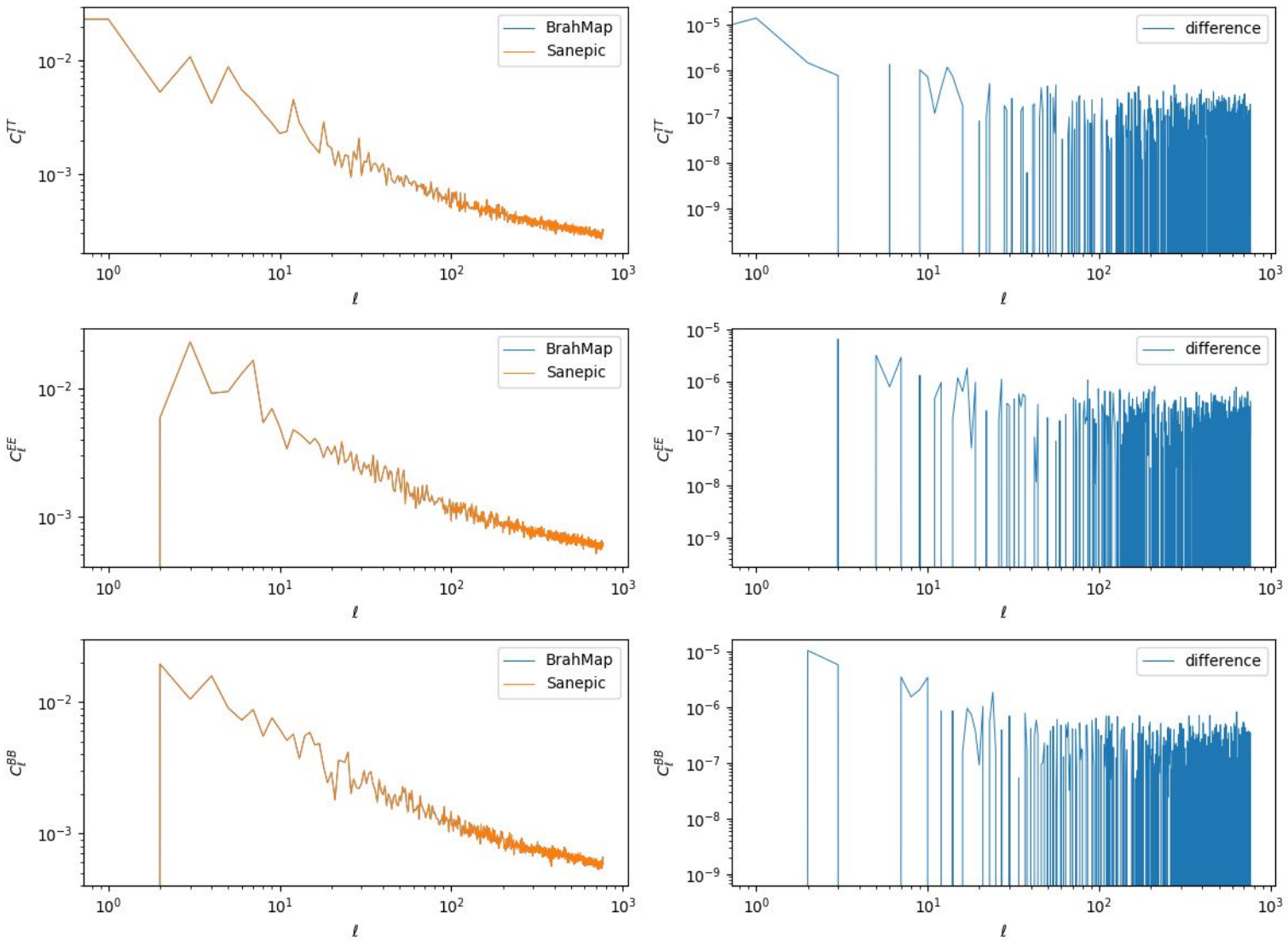
1. Noise covariance (and their inverse)
   a. Diagonal noise covariance
   b. Circulant noise covariance
   c. Band-diagonal noise covariance
   d. Toeplitz noise covariance (inverse computed through PCG)
2. Block diagonal noise covariance (incorporating the noise covariances listed above)

Main Results

Noise covariance operators

Validation against SANEPIC

- Output maps
- Power spectra



Final Steps

- Documentation update

June - July 2025

- Profiling OpenMP implementation
- **KPI**: Offloading to GPUs
  - cupy arrays on Python side
  - Exposing cupy arrays to C++ with Array API
  - OR use in cupy.RawKernel Python

August - Sept 2025

- Benchmarking with large simulations
  - Map-making with full focal plane simulation
- Preparing the final results