NeuroSpin METRIC MIND Inria université PARIS-SACLAY

cea joliot

MRI-NUFFT: Non Cartesian operators and trajectories

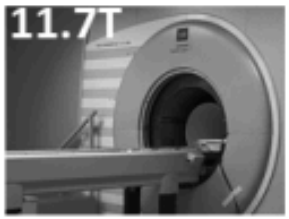
2026-05-22 – SFB MR-Dynamo

Pierre-Antoine Comby

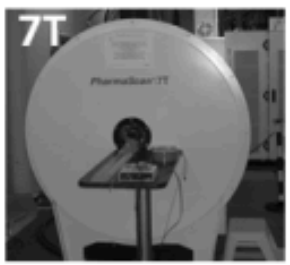
Neurospin, CEA, Paris-Saclay University

Neurospin Lab

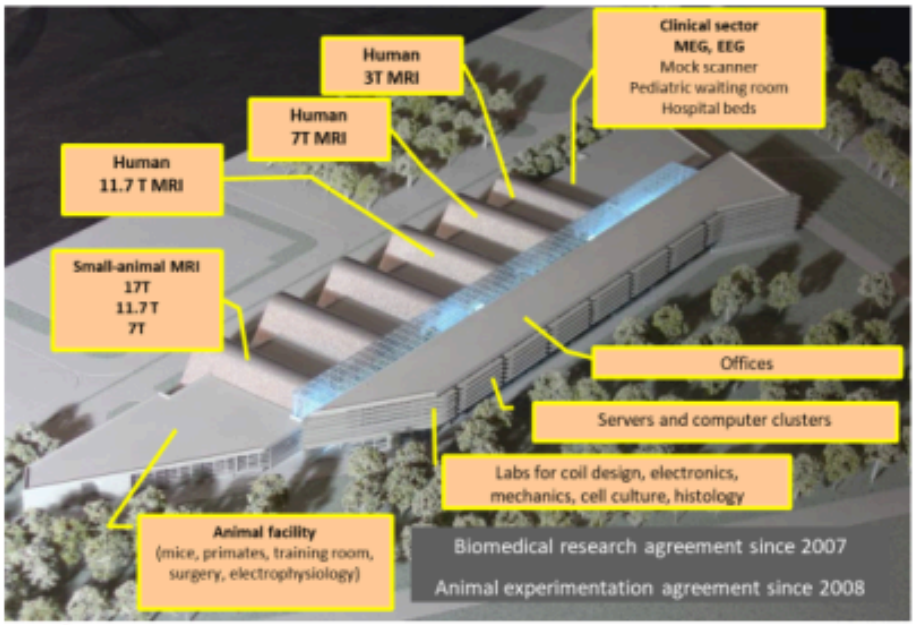
Clinical MRI



Preclinical MRI



EEG, MEG

UNICOG Cognitive Neuroimaging Lab

UNIACT unité de neuroimagerie appliquée, clinique et translationnelle

MIND

BAOBAB

Inserm

université PARIS-SACLAY

Inria

cnrs

Outline

1. Context

(Yet Another) Recall on MRI Acquisition

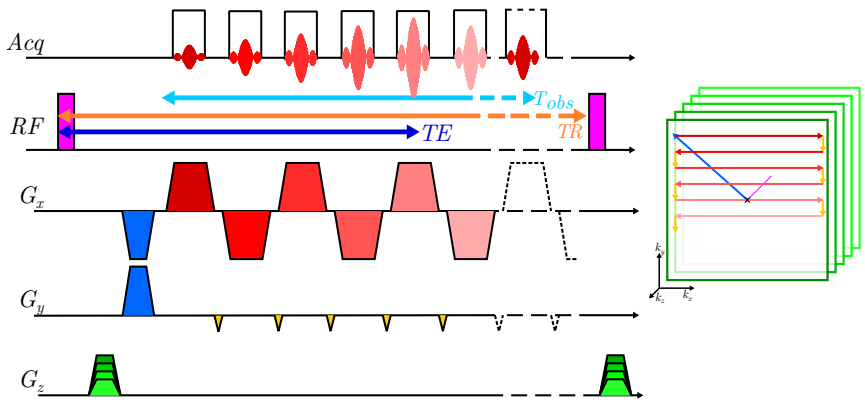


Figure 1: Simple 3D EPI MRI sequence

- MRI is typically acquired using a Cartesian sampling strategy
  - Line-readout EPI, 3D EPI, SMS, …
  - Undersampling (Acceleration) is limited to sub-sampling along k_x or k_y axis (phase-encoding)

(Yet Another) Recall on MRI Acquisition

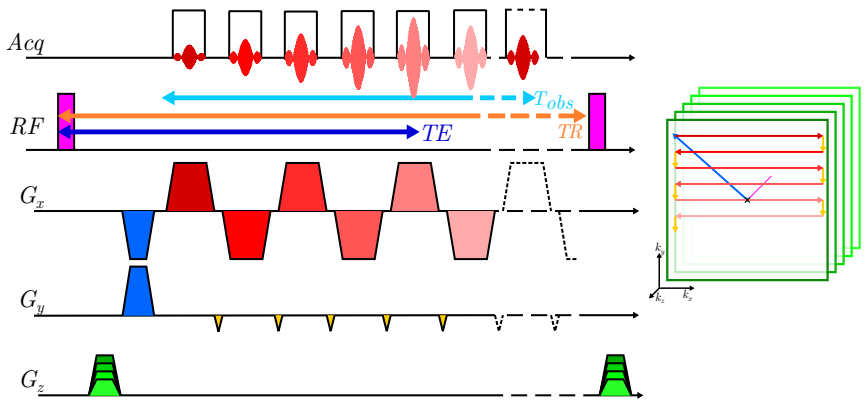
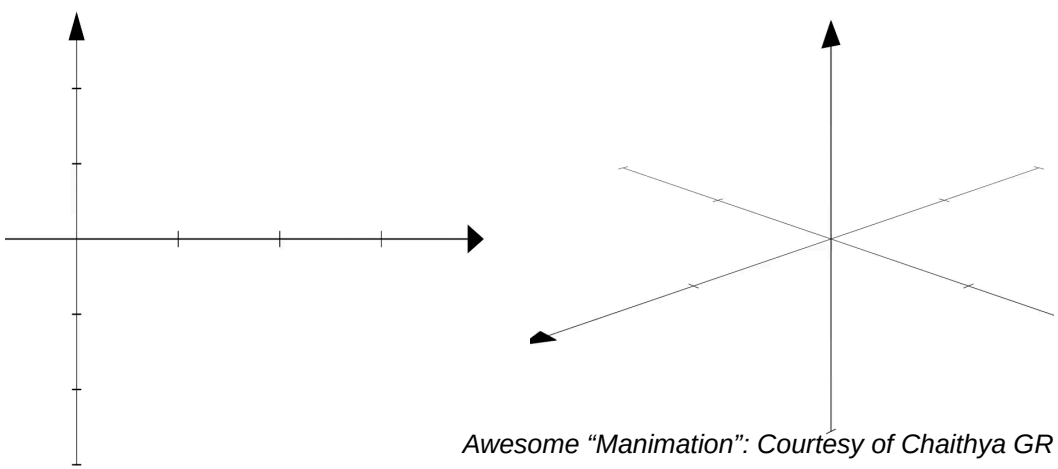


Figure 1: Simple 3D EPI MRI sequence



Awesome "Manimation": Courtesy of Chaithya GR

- MRI is typically acquired using a Cartesian sampling strategy
  - Line-readout EPI, 3D EPI, SMS, …
  - Undersampling (Acceleration) is limited to sub-sampling along k_x or k_y axis (phase-encoding)
- The trajectory is determined by the gradient waveform:

$$k_u(t) = \frac{\gamma}{2\pi} \int_0^t G_u(\tau) d\tau, \forall u \in \{x, y, z\}$$

(Yet Another) Recall on MRI Acquisition

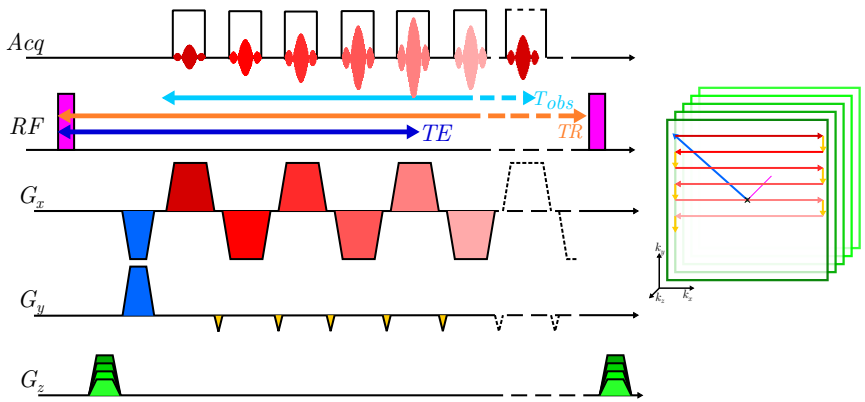
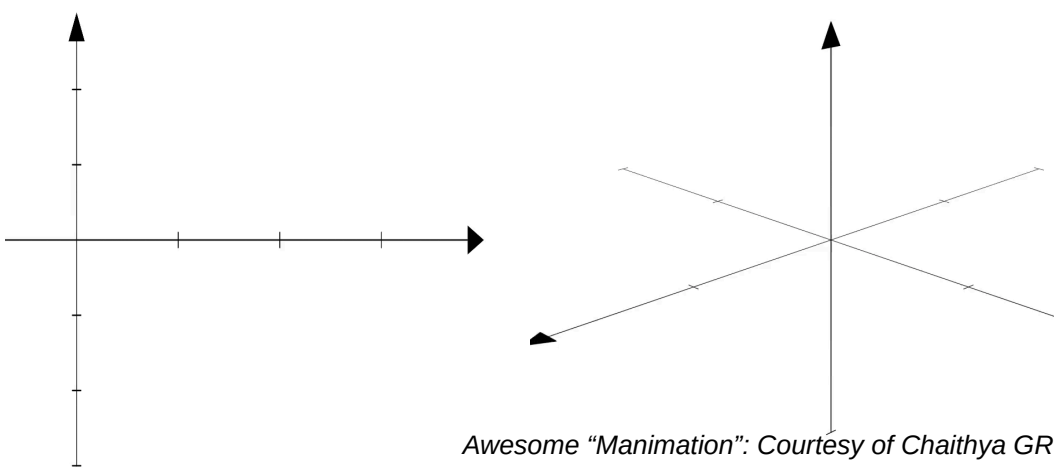


Figure 1: Simple 3D EPI MRI sequence



Awesome "Manimation": Courtesy of Chaithya GR

- MRI is typically acquired using a Cartesian sampling strategy
  - Line-readout EPI, 3D EPI, SMS, …
  - Undersampling (Acceleration) is limited to sub-sampling along k_x or k_y axis (phase-encoding)
- The trajectory is determined by the gradient waveform:

$$k_u(t) = \frac{\gamma}{2\pi} \int_0^t G_u(\tau) d\tau, \forall u \in \{x, y, z\}$$

- There are hardware and safety limits on the gradient

$$\|G_u(t)\|_2 \leq G_{\max}$$

$$\left\| \frac{dG_u(t)}{dt} \right\|_2 \leq \frac{G_{\max}}{\Delta T} = S_{\max}$$

MRI Reconstruction

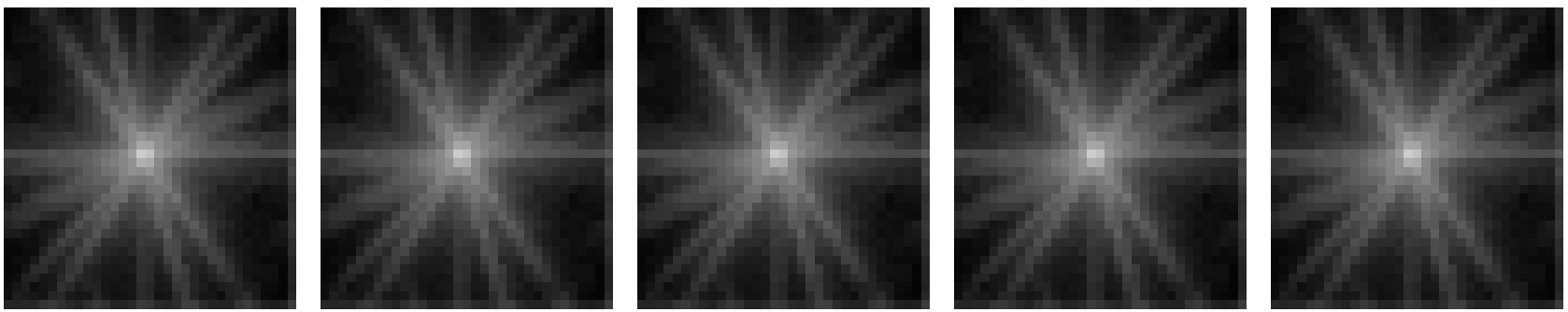
- Solving one (or many) *ill-posed* **inverse problem**

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}$$

MRI Reconstruction

- Solving one (or many) *ill-posed* **inverse problem**

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}$$

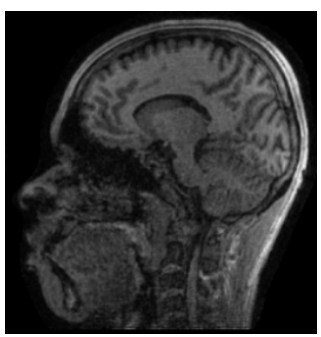


Anatomical MRI | Cardiac /CINE | Quantitative MRI | Diffusion MRI | Functional MRI

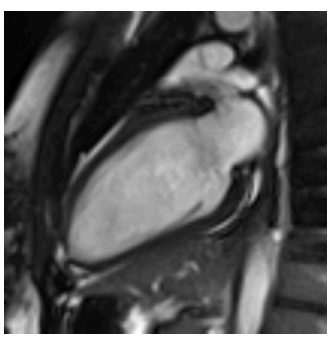
MRI Reconstruction

- Solving one (or many) *ill-posed* **inverse problem**

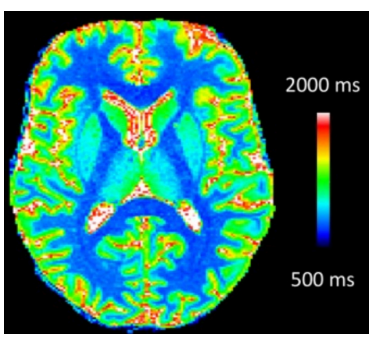
$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon} \quad \Longrightarrow \quad \boldsymbol{x} = \mathcal{M}(\boldsymbol{A}, \boldsymbol{y}, \textit{priors})$$



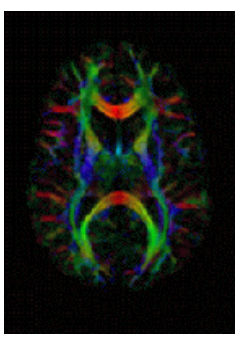
Anatomical MRI



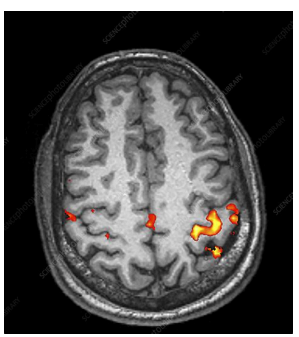
Cardiac /CINE



Quantitative MRI



Diffusion MRI

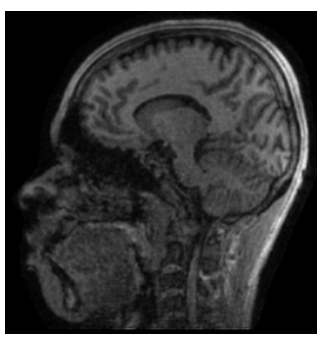


Functional MRI

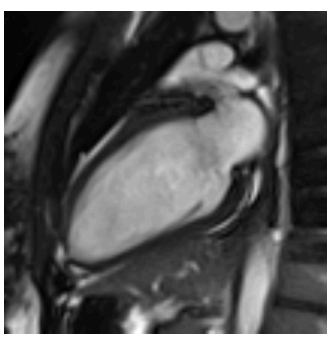
MRI Reconstruction

- Solving one (or many) *ill-posed* **inverse problem**

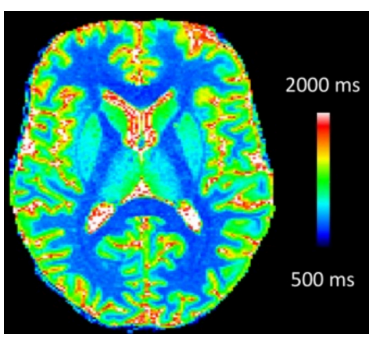
$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon} \quad \Longrightarrow \quad \boldsymbol{x} = \mathcal{M}(\boldsymbol{A}, \boldsymbol{y}, \textit{priors})$$



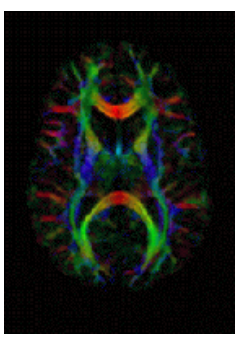
Anatomical MRI



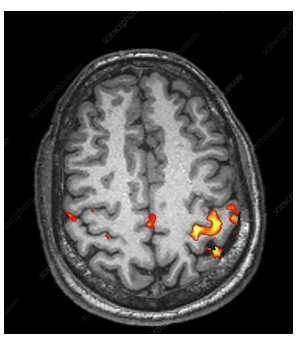
Cardiac /CINE



Quantitative MRI



Diffusion MRI



Functional MRI

Common Goal: Solve the inverse problem

Design (Learn ?) and validate $\mathcal{M}$, $\boldsymbol{A}$ and *priors*

Introducing MRI-NUFFT

- Most MR Toolboxes focus on Cartesian sampling

Introducing MRI-NUFFT

- Most MR Toolboxes focus on Cartesian sampling
- Non-Cartesian: Great potential
  - ✓ Freedom of sampling locations
  - ✓ Increased sampling efficacy
  - ✓ Robustness to motion

Introducing MRI-NUFFT

- Most MR Toolboxes focus on Cartesian sampling
- Non-Cartesian: Great potential
  - ✓ Freedom of sampling locations
  - ✓ Increased sampling efficacy
  - ✓ Robustness to motion
- But …
  - ✗ Require expensive Non-Uniform Fourier Transform
  - ✗ Off-Resonance effects
  - ✗ (Re)-design non-Cartesian trajectories for sequences
  - ✗ MR System constraints

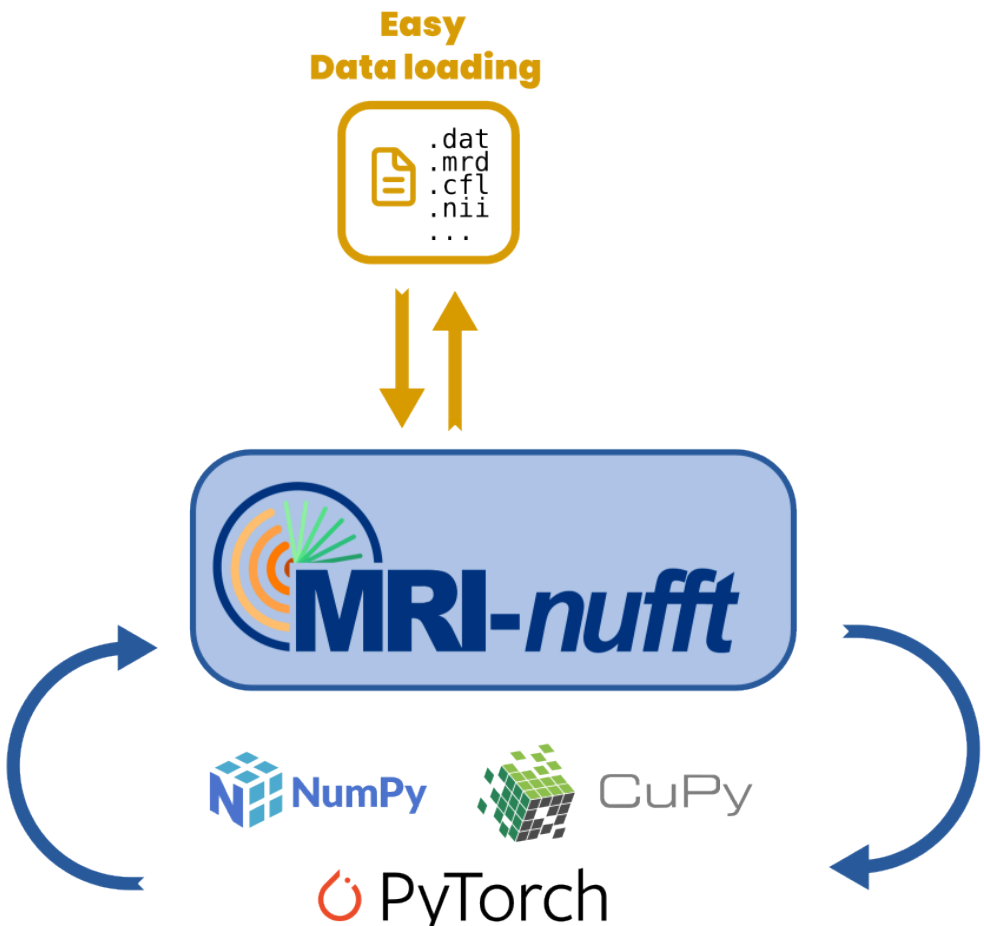
Introducing MRI-NUFFT

- Most MR Toolboxes focus on Cartesian sampling
- Non-Cartesian: Great potential
  - ✓ Freedom of sampling locations
  - ✓ Increased sampling efficacy
  - ✓ Robustness to motion
- But …
  - ✗ Require expensive Non-Uniform Fourier Transform
  - ✗ Off-Resonance effects
  - ✗ (Re)-design non-Cartesian trajectories for sequences
  - ✗ MR System constraints

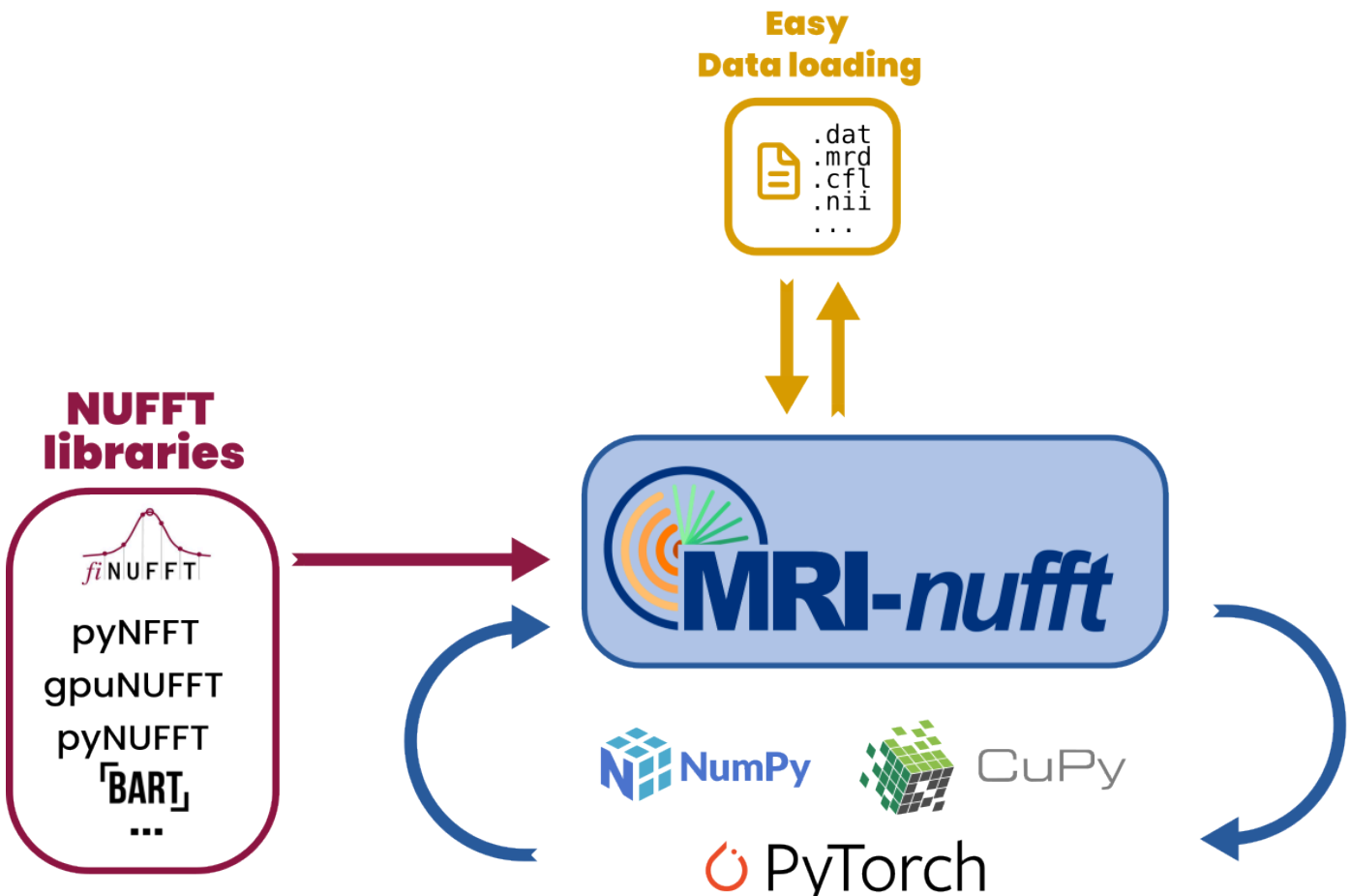
Let's fix this with:

MRI-nufft

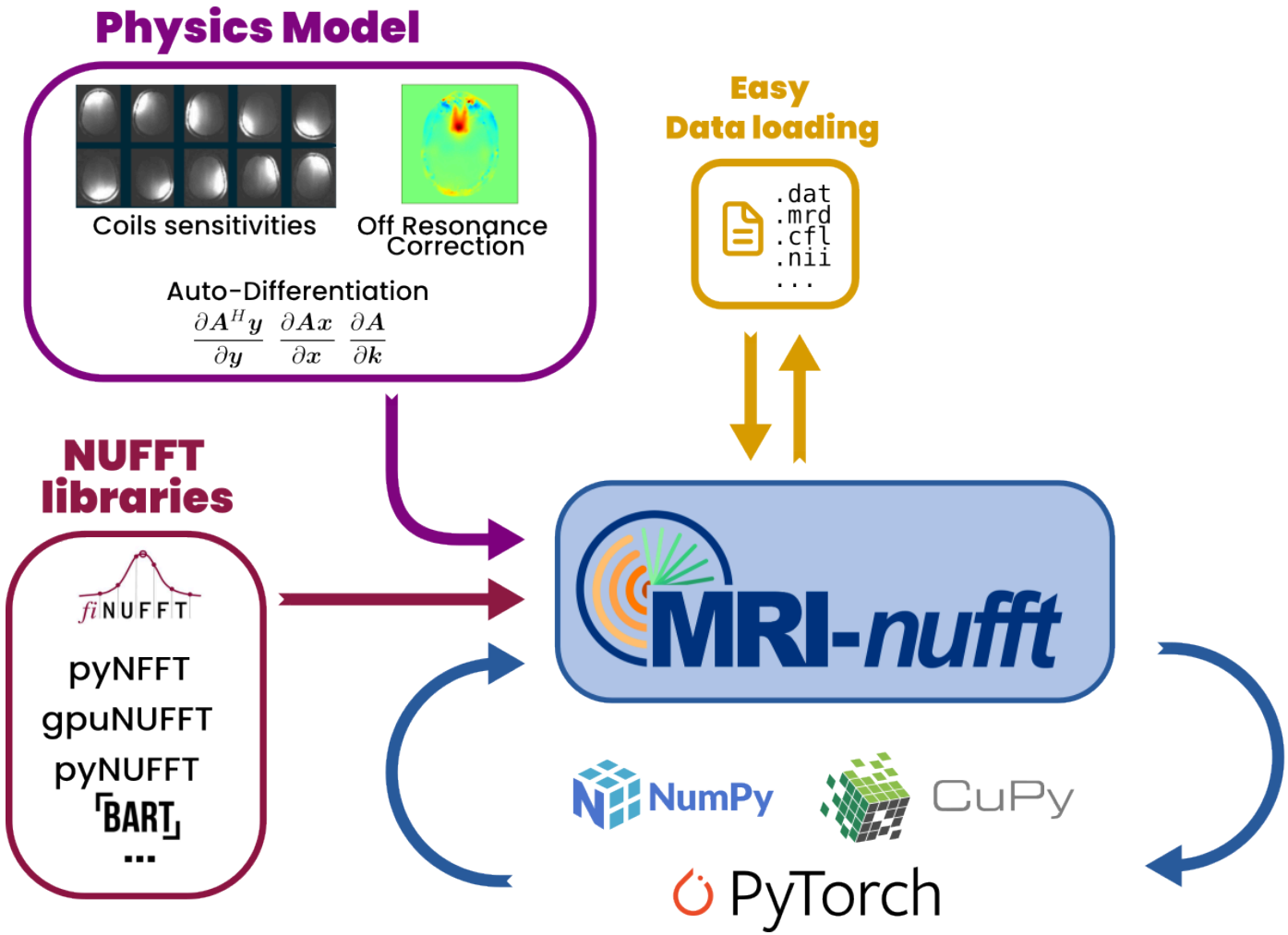
MRI-NUFFT in a Nutshell



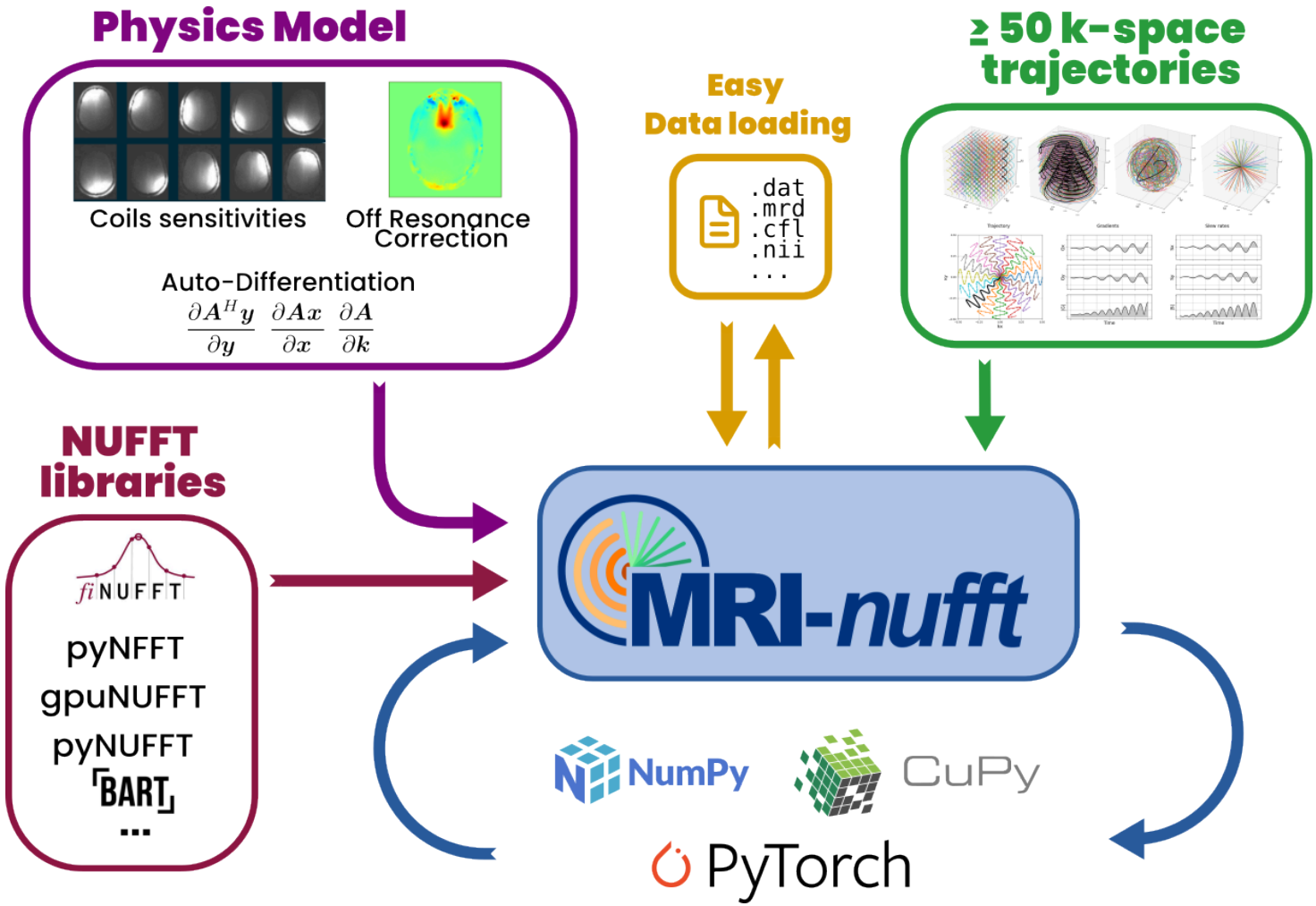
MRI-NUFFT in a Nutshell



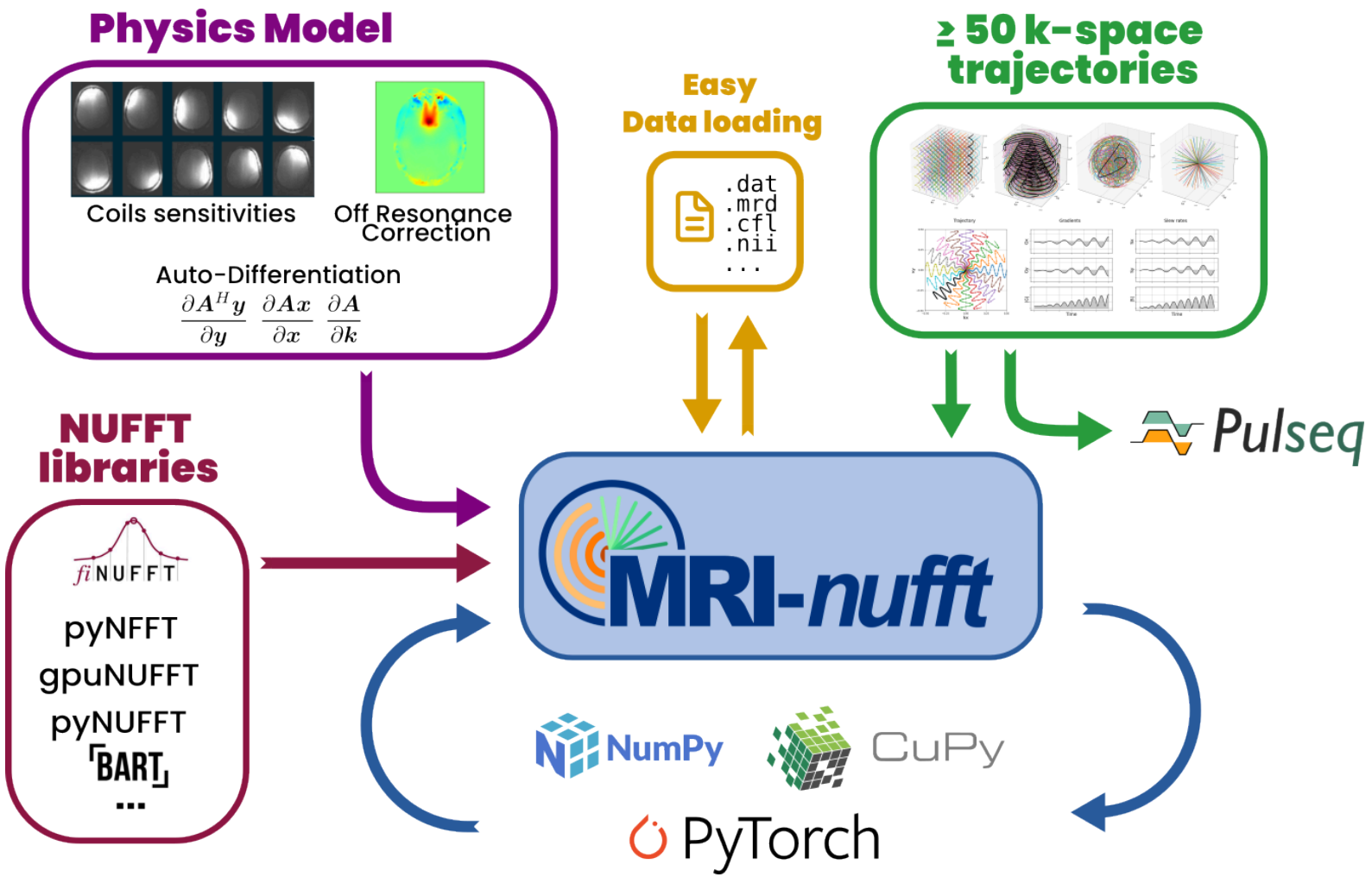
MRI-NUFFT in a Nutshell



MRI-NUFFT in a Nutshell



MRI-NUFFT in a Nutshell



2. Mathematics for non-Cartesian MRI

From FFT to NUFFT

$$y(t) = \int_{\mathbb{R}^d} x(\boldsymbol{u}) e^{-2\imath\pi \boldsymbol{k}(t)\cdot\boldsymbol{u}} d\boldsymbol{u}$$

$$\forall i = 1...M, j = 1...N \qquad y_i = \sum_j x(\boldsymbol{u}_j) e^{-2\imath\pi \boldsymbol{k}_i\cdot\boldsymbol{u}_j}$$

From FFT to NUFFT

$$y(t) = \int_{\mathbb{R}^d} x(\boldsymbol{u}) e^{-2\imath\pi \boldsymbol{k}(t)\cdot\boldsymbol{u}} d\boldsymbol{u}$$

$$\forall i = 1...M, j = 1...N \qquad y_i = \sum_j x(\boldsymbol{u}_j) e^{-2\imath\pi \boldsymbol{k}_i\cdot\boldsymbol{u}_j}$$

- In Cartesian settings, *uniform sampling*: $k_i = i \cdot \delta k$, $u_j = j \cdot \delta u$
  - FFT: $\mathcal{O}(N \log N)$

From FFT to NUFFT

$$y(t) = \int_{\mathbb{R}^d} x(\boldsymbol{u}) e^{-2\imath\pi \boldsymbol{k}(t)\cdot\boldsymbol{u}} d\boldsymbol{u}$$

$$\forall i = 1...M, j = 1...N \qquad y_i = \sum_j x(\boldsymbol{u}_j) e^{-2\imath\pi \boldsymbol{k}_i\cdot\boldsymbol{u}_j}$$

- In Cartesian settings, *uniform sampling*: $k_i = i \cdot \delta k$, $u_j = j \cdot \delta u$
  - FFT: $\mathcal{O}(N \log N)$
- Non Cartesian Settings: *non-uniform sampling* in k-space: $k_i \neq i \cdot \delta k$,

From FFT to NUFFT

$$y(t) = \int_{\mathbb{R}^d} x(\boldsymbol{u}) e^{-2\imath\pi \boldsymbol{k}(t)\cdot\boldsymbol{u}} d\boldsymbol{u}$$

$$\forall i = 1...M, j = 1...N \qquad y_i = \sum_j x(\boldsymbol{u}_j) e^{-2\imath\pi \boldsymbol{k}_i\cdot\boldsymbol{u}_j}$$

- In Cartesian settings, *uniform sampling*: $k_i = i \cdot \delta k$, $u_j = j \cdot \delta u$
  - FFT: $\mathcal{O}(N \log N)$
- Non Cartesian Settings: *non-uniform sampling* in k-space: $k_i \neq i \cdot \delta k$,
  - Greedy: Discrete Fourier Transform : $\mathcal{O}(MN)$

From FFT to NUFFT

$$y(t) = \int_{\mathbb{R}^d} x(\boldsymbol{u}) e^{-2\imath\pi \boldsymbol{k}(t)\cdot\boldsymbol{u}} d\boldsymbol{u}$$

$$\forall i = 1...M, j = 1...N \qquad y_i = \sum_j x(\boldsymbol{u}_j) e^{-2\imath\pi \boldsymbol{k}_i\cdot\boldsymbol{u}_j}$$

- In Cartesian settings, *uniform sampling*: $k_i = i \cdot \delta k$, $u_j = j \cdot \delta u$
  - FFT: $\mathcal{O}(N \log N)$
- Non Cartesian Settings: *non-uniform sampling* in k-space: $k_i \neq i \cdot \delta k$,
  - Greedy: Discrete Fourier Transform : $\mathcal{O}(MN)$
  - Smarter: Non-Uniform Fast Fourier Transform $\mathcal{O}(\alpha N \log(N) + M/\log(\varepsilon))$
    1. Interpolation (ε-precise Kernel) to an oversampled grid (α)
    2. FFT
    3. Apodization correction and cropping FOV

Fessler&Sutton, IEEE TSP, 2003.
Barnett et al., 2019
Shih et al., 2021

Non-Cartesian MRI reconstruction

- Inverse problem formulation for MRI

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}\in\mathbb{C}^N} \frac{1}{2}\sum_{\ell=1}^{L} \|\mathcal{F}_\Omega \mathcal{S}_\ell \boldsymbol{x} - \boldsymbol{y}_\ell\|_2^2 + \lambda g(\boldsymbol{x})$$

Non-Cartesian MRI reconstruction

- Inverse problem formulation for MRI

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}\in\mathbb{C}^N} \frac{1}{2}\sum_{\ell=1}^{L} \|\mathcal{F}_\Omega \mathcal{S}_\ell \boldsymbol{x} - \boldsymbol{y}_\ell\|_2^2 + \lambda g(\boldsymbol{x})$$

“Priors and Networks are temporary, …

… Forward model is for ever“

Non-Cartesian MRI reconstruction

- Inverse problem formulation for MRI

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}\in\mathbb{C}^N} \frac{1}{2}\sum_{\ell=1}^{L}\|\mathcal{F}_\Omega \mathcal{S}_\ell \boldsymbol{x} - \boldsymbol{y}_\ell\|_2^2 + \lambda g(\boldsymbol{x})$$

“Priors and Networks are temporary, …

… Forward model is for ever“

Non-Cartesian MRI reconstruction

- Inverse problem formulation for MRI

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}\in\mathbb{C}^N} \frac{1}{2}\sum_{\ell=1}^{L}\|\mathcal{F}_\Omega \mathcal{S}_\ell \boldsymbol{x} - \boldsymbol{y}_\ell\|_2^2 + \lambda g(\boldsymbol{x})$$

> *“Priors and Networks are temporary, …*
>
> *… Forward model is for ever“*

MRI-NUFFT provides *efficient* implementation for the forward model operator:

- based on the NUFFT backend of your choice.
- Handles CPU/GPU transfers and array conversion (`torch/numpy/cupy`)
- `autograd` support $\frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \boldsymbol{x}}$, $\frac{\partial \boldsymbol{A}^{\boldsymbol{H}}\boldsymbol{y}}{\partial \boldsymbol{y}}$, $\frac{\partial \boldsymbol{A}_\theta \boldsymbol{x}}{\partial \theta}$, …

Non-Cartesian MRI reconstruction

- Inverse problem formulation for MRI

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}\in\mathbb{C}^N} \frac{1}{2}\sum_{\ell=1}^{L}\|\mathcal{F}_\Omega \mathcal{S}_\ell \boldsymbol{x} - \boldsymbol{y}_\ell\|_2^2 + \lambda g(\boldsymbol{x})$$

> *"Priors and Networks are temporary, …*
>
> *… Forward model is for ever"*

MRI-NUFFT provides *efficient* implementation for the forward model operator:

- based on the NUFFT backend of your choice.
- Handles CPU/GPU transfers and array conversion (`torch/numpy/cupy`)
- `autograd` support $\frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \boldsymbol{x}}, \frac{\partial \boldsymbol{A}^H\boldsymbol{y}}{\partial \boldsymbol{y}}, \frac{\partial \boldsymbol{A}_\theta\boldsymbol{x}}{\partial \theta}, \ldots$

```python
from mrinufft import get_operator
nufft = get_operator("cufinufft", samples, shape, smaps=smaps,
density="pipe")
ksp = nufft.op(image)          # y = Ax
adjoint = nufft.adj_op(ksp)    # x = A^H y
pinv = nufft.pinv_solver(ksp) # x = A^† y, using CG
```

Extended Fourier Model

- Multi-coil, subspace representations, …

Daval-Frérot et al., MRM, 2022.; *Sutton et al.*, IEEE TMI, 2003.

Extended Fourier Model

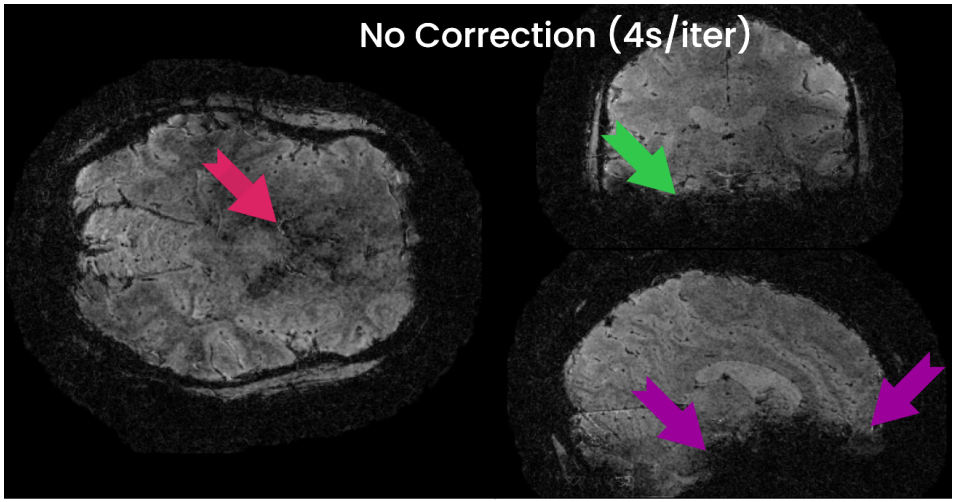
- Multi-coil, subspace representations, …
- Density Compensation (`voronoi`, `pipe`, …)

Daval-Frérot et al., MRM, 2022.; *Sutton et al.*, IEEE TMI, 2003.

Extended Fourier Model

- Multi-coil, subspace representations, …
- Density Compensation (`voronoi`, `pipe`, …)
- Off-Resonance

$$\boldsymbol{y}_\ell(t) = \int_{FOV} \boldsymbol{x}(\boldsymbol{r}) \mathcal{S}_\ell(\boldsymbol{r}) e^{-\Delta\omega(\boldsymbol{r})t} e^{-2i\pi \boldsymbol{k}(t)\cdot\boldsymbol{r}} \mathrm{d}\boldsymbol{r}$$



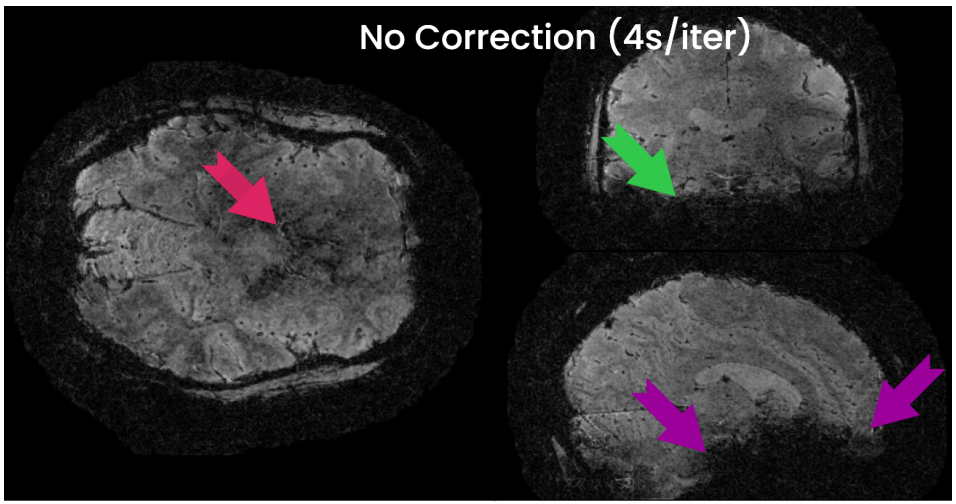
Daval-Frérot et al., MRM, 2022.; *Sutton et al.*, IEEE TMI, 2003.

Extended Fourier Model

- Multi-coil, subspace representations, …
- Density Compensation (`voronoi`, `pipe`, …)
- Off-Resonance correction via bilinearization (MTI, MFI)

$$\boldsymbol{y}_\ell(t) = \int_{FOV} \boldsymbol{x}(\boldsymbol{r}) \mathcal{S}_\ell(\boldsymbol{r}) e^{-\Delta\omega(\boldsymbol{r})t} e^{-2i\pi \boldsymbol{k}(t)\cdot\boldsymbol{r}} \mathrm{d}\boldsymbol{r}$$

$$\boldsymbol{y}_\ell \approx \sum_{p=1}^{P} \boldsymbol{b}_p \sum_{\ell=1}^{L} \mathcal{F}_\Omega \left(\boldsymbol{c}_p \mathcal{S}_\ell \boldsymbol{x} \right)$$



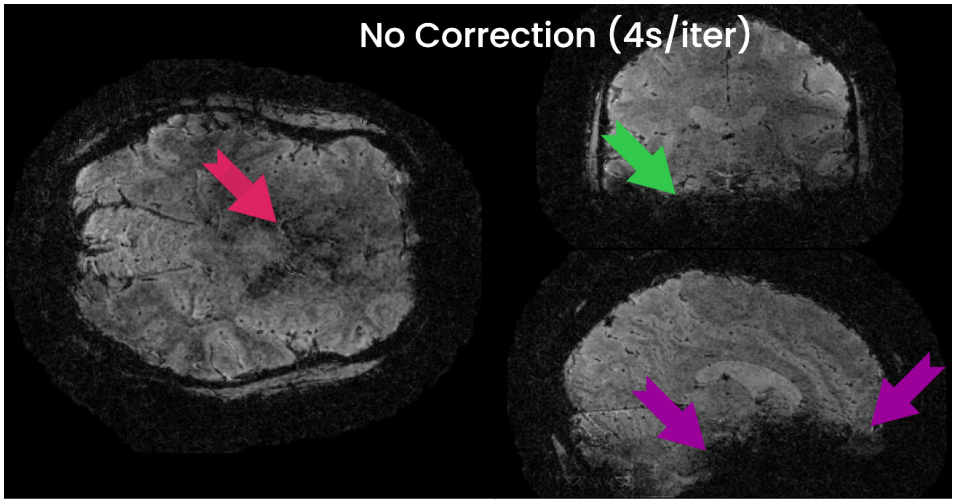
Daval-Frérot et al., MRM, 2022.; *Sutton et al.*, IEEE TMI, 2003.

Extended Fourier Model

- Multi-coil, subspace representations, …
- Density Compensation (`voronoi`, `pipe`, …)
- Off-Resonance
  correction via bilinearization (MTI, MFI, **SVD**)

$$\boldsymbol{y}_\ell(t) = \int_{FOV} \boldsymbol{x}(\boldsymbol{r}) \mathcal{S}_\ell(\boldsymbol{r}) e^{-\Delta\omega(\boldsymbol{r})t} e^{-2i\pi \boldsymbol{k}(t)\cdot\boldsymbol{r}} \mathrm{d}\boldsymbol{r}$$

$$\min_{\substack{\boldsymbol{B}\in\mathbb{C}^{M\times P} \\ \boldsymbol{C}\in\mathbb{C}^{P\times N}}} \| e^{-\Delta\omega(\boldsymbol{r})t} - \boldsymbol{B}\boldsymbol{C} \|_2^2 \Longrightarrow \boldsymbol{y}_\ell \approx \sum_{p=1}^{P} \boldsymbol{b}_p \sum_{\ell=1}^{L} \mathcal{F}_\Omega \left(\boldsymbol{c}_p \mathcal{S}_\ell \boldsymbol{x} \right)$$



Daval-Frérot et al., MRM, 2022.; *Sutton et al.*, IEEE TMI, 2003.

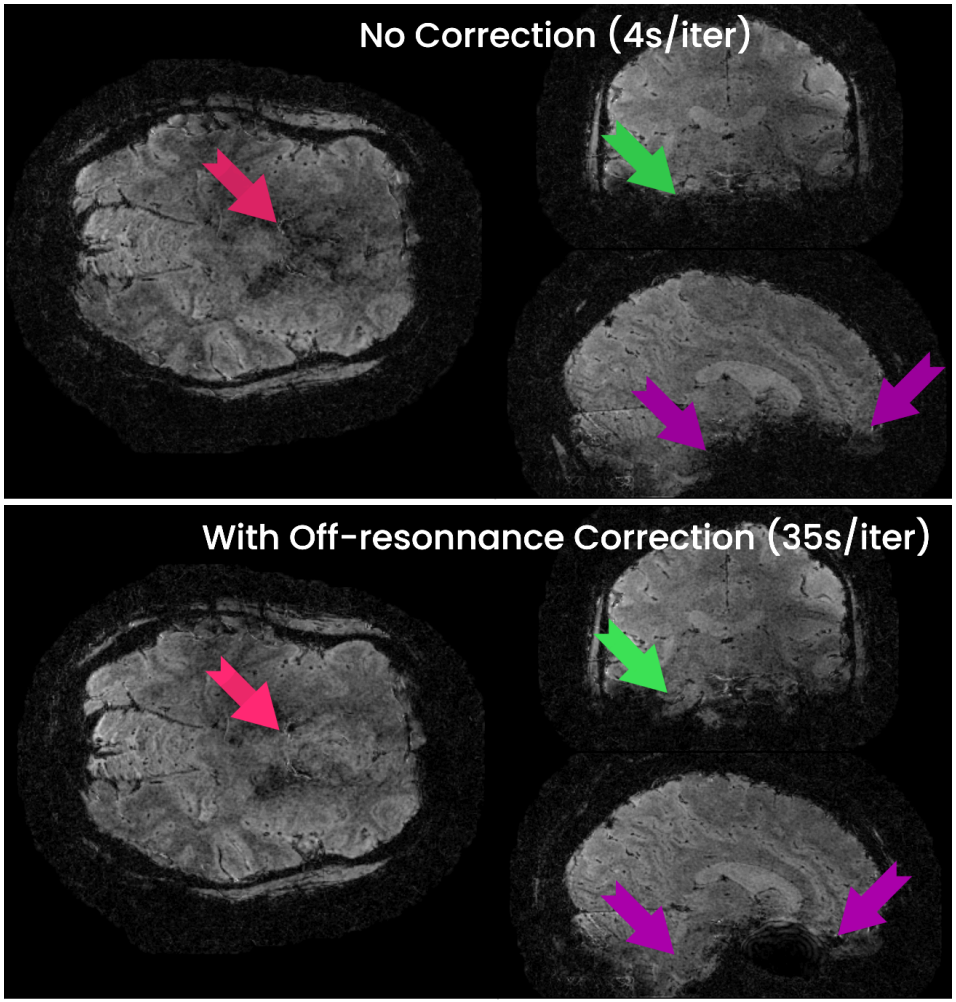
Extended Fourier Model

- Multi-coil, subspace representations, …
- Density Compensation (`voronoi`, `pipe`, …)
- Off-Resonance correction via bilinearization (MTI, MFI, **SVD**)

$$\boldsymbol{y}_\ell(t) = \int_{FOV} \boldsymbol{x}(\boldsymbol{r}) \mathcal{S}_\ell(\boldsymbol{r}) e^{-\Delta\omega(\boldsymbol{r})t} e^{-2i\pi \boldsymbol{k}(t)\cdot\boldsymbol{r}} \mathrm{d}\boldsymbol{r}$$

$$\min_{\substack{\boldsymbol{B}\in\mathbb{C}^{M\times P}\\ \boldsymbol{C}\in\mathbb{C}^{P\times N}}} \| e^{-\Delta\omega(\boldsymbol{r})t} - \boldsymbol{B}\boldsymbol{C} \|_2^2 \Longrightarrow \boldsymbol{y}_\ell \approx \sum_{p=1}^{P} \boldsymbol{b}_p \sum_{\ell=1}^{L} \mathcal{F}_\Omega \left(\boldsymbol{c}_p \mathcal{S}_\ell \boldsymbol{x} \right)$$

```
nufft_orc = nufft.with_off_resonance_correction(
    field_map=field_map,
    readout_time=readout_time,
    interpolators=20)
adj_orc = nufft_orc.adj_op(ksp) # x = A^H_{Δω} y
pinv_orc = nufft_orc.pinv_solver(ksp) # x = A^†_{Δω} y
```



Daval-Frérot et al., MRM, 2022.; *Sutton et al.*, IEEE TMI, 2003.

Physics operators for Deep Learning

MRI-NUFFT Physics operator:

$$\boldsymbol{y} = f(\mathcal{S}, \Omega, \Delta\omega, \boldsymbol{x}) = A(\mathcal{S}, \Omega, \Delta\omega)\boldsymbol{x}$$

Physics operators for Deep Learning

MRI-NUFFT Physics operator:

$$\boldsymbol{y} = f(\mathcal{S}, \Omega, \Delta\omega, \boldsymbol{x}) = A(\mathcal{S}, \Omega, \Delta\omega)\boldsymbol{x}$$

- In deep-learning application we rely on reverse-mode autodifferentiation
  - 😵 VJP computations, complex derivatives and Wirtinger calculus

$$\langle \boldsymbol{v} \mid \frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \boldsymbol{x}} \rangle \qquad \langle \boldsymbol{v} \mid \frac{\partial \boldsymbol{A}^{\boldsymbol{H}}\boldsymbol{y}}{\partial \boldsymbol{y}} \rangle \qquad \langle \boldsymbol{v} \mid \frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \Omega} \rangle \qquad \langle \boldsymbol{v} \mid \frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \mathcal{S}} \rangle \qquad \dots$$

Physics operators for Deep Learning

MRI-NUFFT Physics operator:

$$\boldsymbol{y} = f(\mathcal{S}, \Omega, \Delta\omega, \boldsymbol{x}) = A(\mathcal{S}, \Omega, \Delta\omega)\boldsymbol{x}$$

- In deep-learning application we rely on reverse-mode autodifferentiation
  - 😵 VJP computations, complex derivatives and Wirtinger calculus

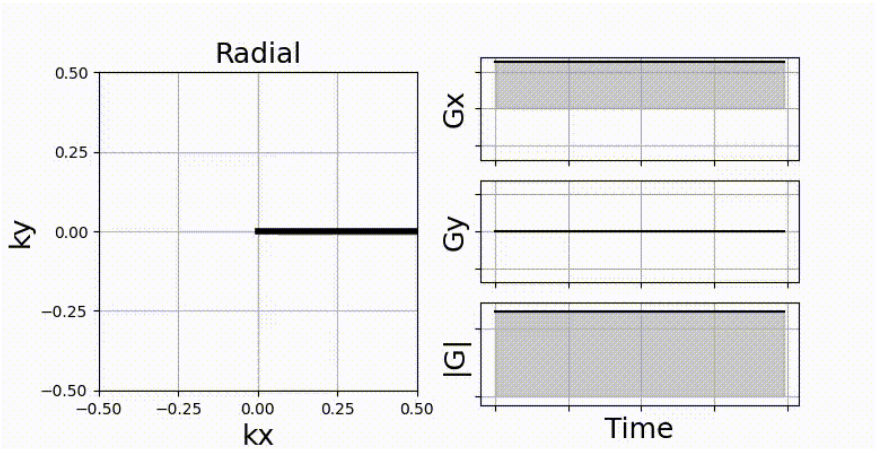
$$\langle \boldsymbol{v} \mid \frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \boldsymbol{x}} \rangle \qquad \langle \boldsymbol{v} \mid \frac{\partial \boldsymbol{A}^{\boldsymbol{H}}\boldsymbol{y}}{\partial \boldsymbol{y}} \rangle \qquad \langle \boldsymbol{v} \mid \frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \Omega} \rangle \qquad \langle \boldsymbol{v} \mid \frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \mathcal{S}} \rangle \qquad \dots$$

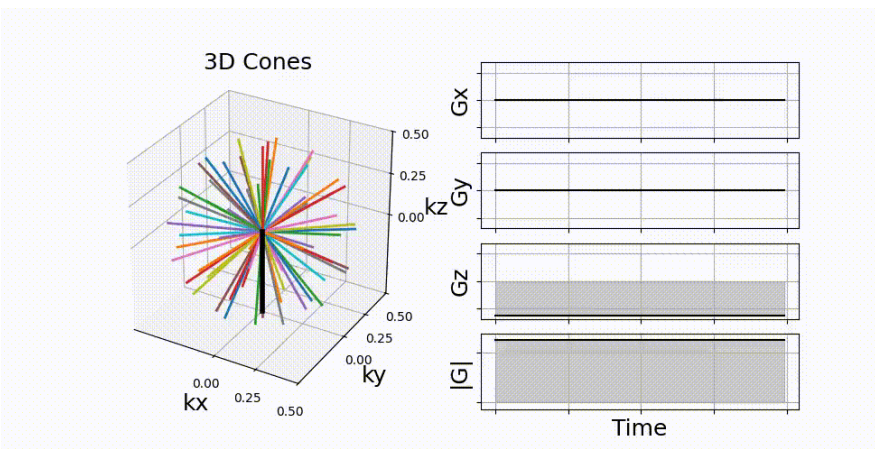
  - 📄 *Wang&Fessler*, IEEE TCI, 2023.
  - 🥳 Efficient implementation in MRI-NUFFT
  - 🚧 WIP on $\frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \Delta\omega}$

3. Trajectories and Gradients

Trajectories & Gradients

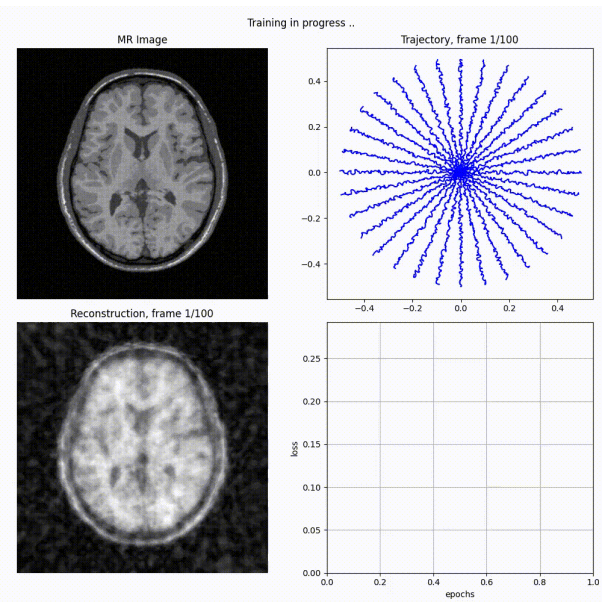
- Large collection of non-Cartesian trajectories
  - 2D and 3D, fully parameterized

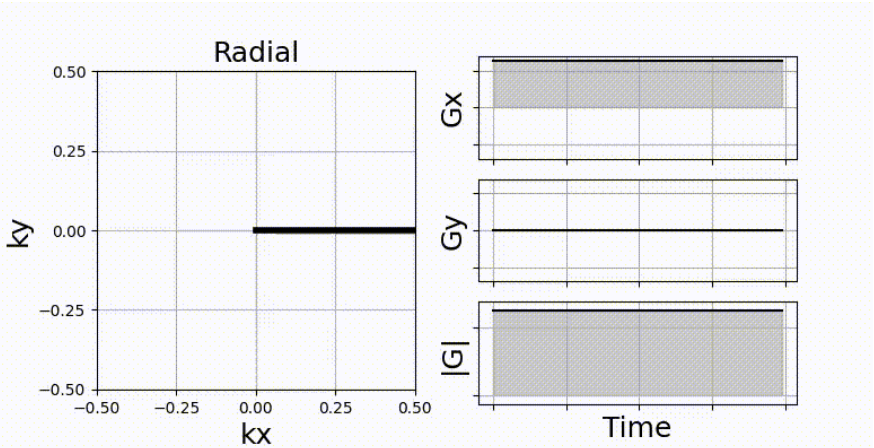


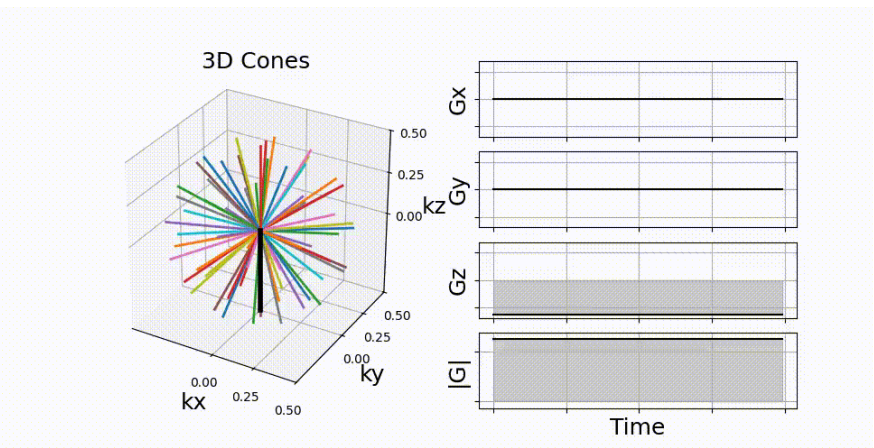


Trajectories & Gradients

- Large collection of non-Cartesian trajectories
  - 2D and 3D, fully parameterized
  - Hardware compliance enforcement.
  - You can even learn trajectories in a deep Learning setting with autodifferentiation !

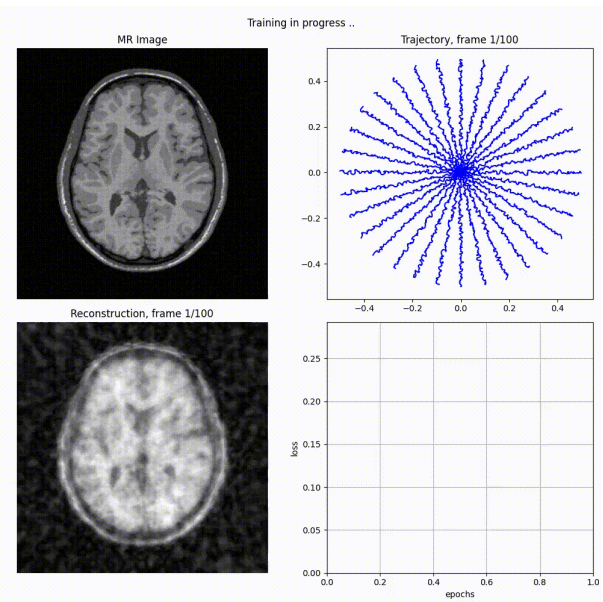






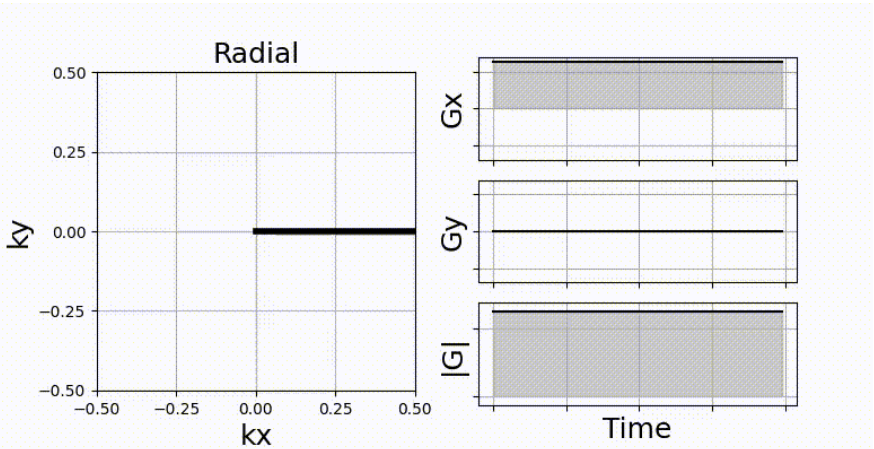
Trajectories & Gradients

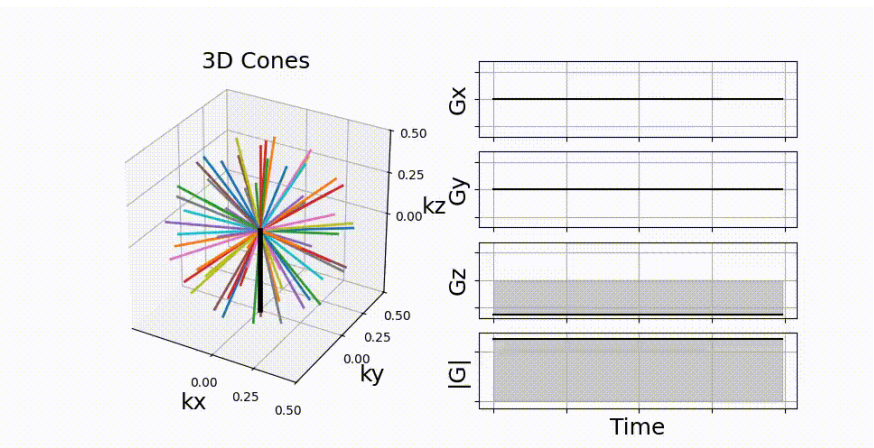
- Large collection of non-Cartesian trajectories
  - 2D and 3D, fully parameterized
  - Hardware compliance enforcement.
  - You can even learn trajectories in a deep Learning setting with autodifferentiation !



Live Demo !

Open in Colab





Harware compliant trajectories: Projection

Trajectories Constraints

Set of admissible trajectories

$$\mathcal{Q} = \left\{ \begin{array}{c} \forall i = \{1, ..., N_c\}, \boldsymbol{k}_i \in \Omega^{N_s} \\ \|\boldsymbol{k}\|_\infty \leq \frac{1}{2}, \|\dot{\boldsymbol{k}}\|_{2,\infty} \leq G_{\max}, \|\ddot{\boldsymbol{k}}\|_{2,\infty} \leq S_{\max} \\ \boldsymbol{A_i k_i} = \boldsymbol{b}_i \end{array} \right\}$$

Harware compliant trajectories: Projection

Trajectories Constraints

Set of admissible trajectories

$$\mathcal{Q} = \left\{ \begin{array}{c} \forall i = \{1, ..., N_c\}, \boldsymbol{k}_i \in \Omega^{N_s} \\ \|\boldsymbol{k}\|_\infty \leq \frac{1}{2}, \|\dot{\boldsymbol{k}}\|_{2,\infty} \leq G_{\max}, \|\ddot{\boldsymbol{k}}\|_{2,\infty} \leq S_{\max} \\ \boldsymbol{A_i k_i} = \boldsymbol{b}_i \end{array} \right\}$$

Given an existing trajectory $\boldsymbol{k} \in \Omega^{N_c \times N_s}$:

$$\boldsymbol{k}' = \arg\min_{\boldsymbol{k}' \in \mathcal{Q}} \| \boldsymbol{k} - \boldsymbol{k}' \|_2$$

Solved with a Primal Dual Algorithm *Chauffert et al.*, IEEE TMI, 2016; *Chaithya et al.*, IEEE TMI, 2022.

Harware compliant trajectories: Projection

Trajectories Constraints

Set of admissible trajectories

$$\mathcal{Q} = \left\{ \begin{array}{c} \forall i = \{1, ..., N_c\}, \boldsymbol{k}_i \in \Omega^{N_s} \\ \|\boldsymbol{k}\|_\infty \leq \frac{1}{2}, \|\dot{\boldsymbol{k}}\|_{2,\infty} \leq G_{\max}, \|\ddot{\boldsymbol{k}}\|_{2,\infty} \leq S_{\max} \\ \boldsymbol{A_i k_i} = \boldsymbol{b}_i \end{array} \right\}$$

Given an existing trajectory $\boldsymbol{k} \in \Omega^{N_c \times N_s}$:

$$\boldsymbol{k}' = \arg\min_{\boldsymbol{k}' \in \mathcal{Q}} \| \boldsymbol{k} - \boldsymbol{k}' \|_2$$

Solved with a Primal Dual Algorithm *Chauffert et al.*, IEEE TMI, 2016; *Chaithya et al.*, IEEE TMI, 2022.

```python
from mrinufft.trajectories import Acquisition, SIEMENS
acq = Acquisition(
    hardware = SIEMENS.TERRAX,
    TE=0.050, TR=0.100, FA=12,
    fov=(.192,.192,.129), img_size=(64,64,43))
new_traj = project_trajectory(traj, acq=acq)
```

Harware compliant trajectories: Projection

Trajectories Constraints

Set of admissible trajectories

$$\mathcal{Q} = \left\{ \begin{array}{c} \forall i = \{1, ..., N_c\}, \boldsymbol{k}_i \in \Omega^{N_s} \\ \|\boldsymbol{k}\|_\infty \leq \frac{1}{2}, \|\dot{\boldsymbol{k}}\|_{2,\infty} \leq G_{\max}, \|\ddot{\boldsymbol{k}}\|_{2,\infty} \leq S_{\max} \\ \boldsymbol{A_i k_i} = \boldsymbol{b}_i \end{array} \right\}$$

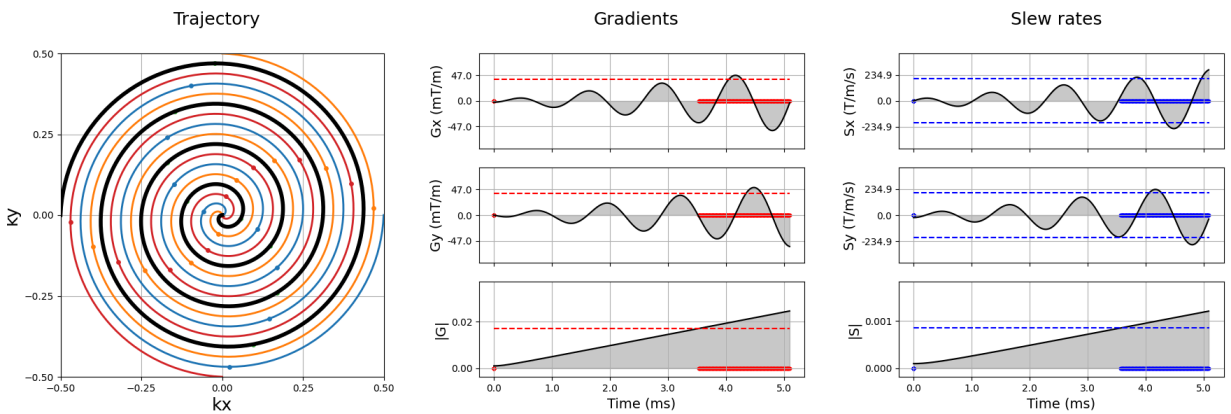
Given an existing trajectory $\boldsymbol{k} \in \Omega^{N_c \times N_s}$:

$$\boldsymbol{k}' = \arg\min_{\boldsymbol{k}' \in \mathcal{Q}} \| \boldsymbol{k} - \boldsymbol{k}' \|_2$$

Solved with a Primal Dual Algorithm Chauffert et al., IEEE TMI, 2016; *Chaithya et al.*, IEEE TMI, 2022.

```
from mrinufft.trajectories import Acquisition, SIEMENS
acq = Acquisition(
    hardware = SIEMENS.TERRAX,
    TE=0.050, TR=0.100, FA=12,
    fov=(.192,.192,.129), img_size=(64,64,43))
new_traj = project_trajectory(traj, acq=acq)
```

No Projection



Harware compliant trajectories: Projection

Trajectories Constraints

Set of admissible trajectories

$$\mathcal{Q} = \left\{ \begin{array}{c} \forall i = \{1, ..., N_c\}, \boldsymbol{k}_i \in \Omega^{N_s} \\ \|\boldsymbol{k}\|_\infty \leq \frac{1}{2}, \|\dot{\boldsymbol{k}}\|_{2,\infty} \leq G_{\max}, \|\ddot{\boldsymbol{k}}\|_{2,\infty} \leq S_{\max} \\ \boldsymbol{A_i k_i} = \boldsymbol{b}_i \end{array} \right\}$$

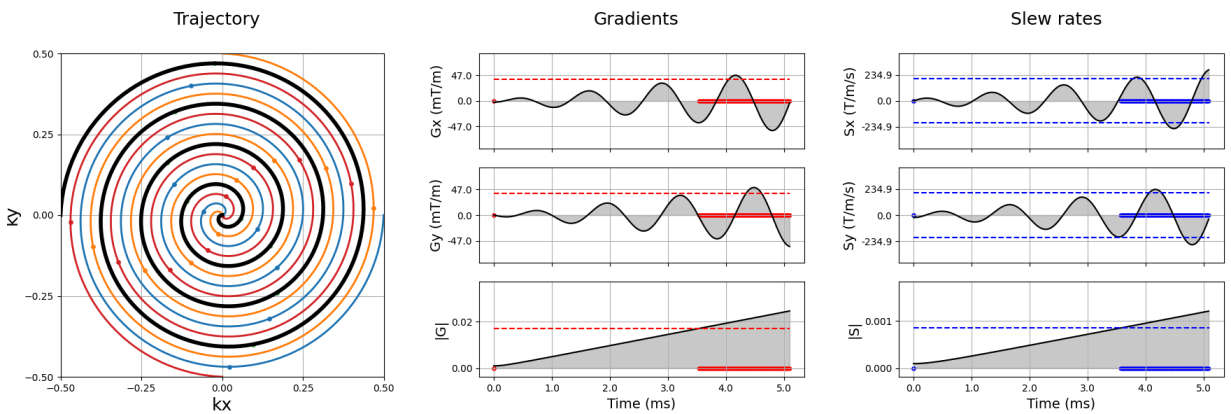
Given an existing trajectory $\boldsymbol{k} \in \Omega^{N_c \times N_s}$:

$$\boldsymbol{k}' = \arg\min_{\boldsymbol{k}' \in \mathcal{Q}} \| \boldsymbol{k} - \boldsymbol{k}' \|_2$$

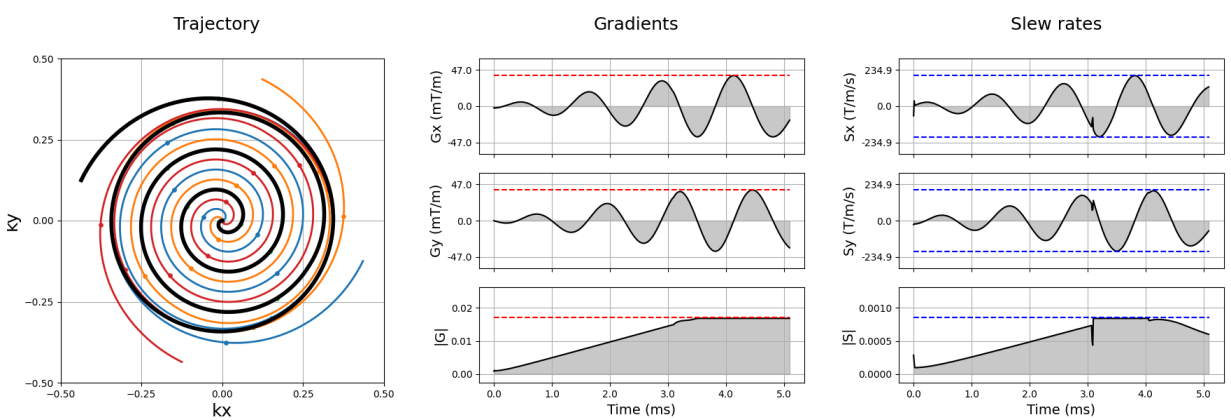
Solved with a Primal Dual Algorithm Chauffert et al., IEEE TMI, 2016; *Chaithya et al.*, IEEE TMI, 2022.

```
from mrinufft.trajectories import Acquisition, SIEMENS
acq = Acquisition(
    hardware = SIEMENS.TERRAX,
    TE=0.050, TR=0.100, FA=12,
    fov=(.192,.192,.129), img_size=(64,64,43))
new_traj = project_trajectory(traj, acq=acq)
```

No Projection



With Projection



Optimal Prephasors and Rewinders

- Non-Cartesian trajectories need to start at an arbitrary point in k-space, but after RF, spins are “resting at the center”

Optimal Prephasors and Rewinders

- Non-Cartesian trajectories need to start at an arbitrary point in k-space, but after RF, spins are “resting at the center”
- Start the non-Cartesian gradients with ADC-off segments: **Prephasors**

Optimal Prephasors and Rewinders

- Non-Cartesian trajectories need to start at an arbitrary point in k-space, but after RF, spins are “resting at the center”
- Start the non-Cartesian gradients with ADC-off segments: **Prephasors**
- At the end, we want to crush any remaining magnetization: **Spoilers**

Optimal Prephasors and Rewinders

- Non-Cartesian trajectories need to start at an arbitrary point in k-space, but after RF, spins are “resting at the center”
- Start the non-Cartesian gradients with ADC-off segments: **Prephasors**
- At the end, we want to crush any remaining magnetization: **Spoilers**

Design Gradient Waveforms

Given $\|\dot{\boldsymbol{k}}\|_\infty \leq G_{\max}$, $\|\ddot{\boldsymbol{k}}\|_\infty \leq S_{\max}$, $\Delta\boldsymbol{k}$, $\boldsymbol{g}_{start}, \boldsymbol{g}_{end}$

1. What is the minimum of raster time points required?
2. Given this number of points, what is the path to take?
3. Make the trajectory as smooth as possible
   - minimize the slew rate OR
   - minimize curvature of gradients $\|\dddot{k}\|$

Linear and Quadratic Programming

Linear Programming

Given a vector-size N , find $\boldsymbol{g} \in [-G_{\max}, G_{\max}]^N$ s.t.

$\boldsymbol{A}_{\text{eq}}\boldsymbol{g} = \boldsymbol{b}_{\text{eq}}$ $\boldsymbol{A}_{\text{ub}}\boldsymbol{g} \leq \boldsymbol{b}_{\text{ub}}$

$$\boldsymbol{A}_{eq} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 1 \end{bmatrix}, \quad \boldsymbol{b}_{eq} = \begin{bmatrix} \Delta \boldsymbol{k} \\ \boldsymbol{g}_{\text{start}} \\ \boldsymbol{g}_{\text{end}} \end{bmatrix}$$

$$\boldsymbol{A}_{\text{ub}} = \left[\begin{array}{ccc} -1 & 1 & 0 \dots \\ 0 & -1 & 1 \dots \\ \hline 1 & -1 & 0 \dots \\ 0 & 1 & -1 \dots \end{array}\right], \quad \boldsymbol{b}_{\text{ub}} = \begin{bmatrix} S_{\max}\Delta T \\ \vdots \\ S_{\max}\Delta T \end{bmatrix}$$

Linear and Quadratic Programming

Linear Programming

Given a vector-size N , find $\boldsymbol{g} \in [-G_{\max}, G_{\max}]^N$ s.t.

$\boldsymbol{A}_{\text{eq}}\boldsymbol{g} = \boldsymbol{b}_{\text{eq}}$ $\boldsymbol{A}_{\text{ub}}\boldsymbol{g} \leq \boldsymbol{b}_{\text{ub}}$

$$\boldsymbol{A}_{eq} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 1 \end{bmatrix}, \quad \boldsymbol{b}_{eq} = \begin{bmatrix} \Delta \boldsymbol{k} \\ \boldsymbol{g}_{\text{start}} \\ \boldsymbol{g}_{\text{end}} \end{bmatrix}$$

$$\boldsymbol{A}_{\text{ub}} = \left[\begin{array}{rrrr} -1 & 1 & 0 & \dots \\ 0 & -1 & 1 & \dots \\ \hline 1 & -1 & 0 & \dots \\ 0 & 1 & -1 & \dots \end{array}\right], \quad \boldsymbol{b}_{\text{ub}} = \begin{bmatrix} S_{\max}\Delta T \\ \vdots \\ S_{\max}\Delta T \end{bmatrix}$$

Quadratic Programming

Smooth gradient $\rightarrow$ Minimize the curvature

$$\min_{\boldsymbol{g}\in\mathbb{R}^N} \sum_{i=0}^{N-1} \left(g_{i+1} - 2g_i + g_{i-1}\right)^2 \text{ s.t. } g_n \in \mathcal{Q}$$

$$\min_{\boldsymbol{g}\in\mathbb{R}^N} \frac{1}{2}\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g} + \boldsymbol{q}^T\boldsymbol{g} + \boldsymbol{c} \text{ s.t. } \boldsymbol{l} \leq \boldsymbol{A}\boldsymbol{g} \leq \boldsymbol{u}$$

Linear and Quadratic Programming

Linear Programming

Given a vector-size N , find $\boldsymbol{g} \in [-G_{\max}, G_{\max}]^N$ s.t.

$\boldsymbol{A}_{\text{eq}}\boldsymbol{g} = \boldsymbol{b}_{\text{eq}}$ $\boldsymbol{A}_{\text{ub}}\boldsymbol{g} \leq \boldsymbol{b}_{\text{ub}}$

$$\boldsymbol{A}_{eq} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 1 \end{bmatrix}, \quad \boldsymbol{b}_{eq} = \begin{bmatrix} \Delta \boldsymbol{k} \\ \boldsymbol{g}_{\text{start}} \\ \boldsymbol{g}_{\text{end}} \end{bmatrix}$$

$$\boldsymbol{A}_{\text{ub}} = \left[\begin{array}{ccc:c} -1 & 1 & 0 & \dots \\ 0 & -1 & 1 & \dots \\ \hdashline 1 & -1 & 0 & \dots \\ 0 & 1 & -1 & \dots \end{array}\right], \quad \boldsymbol{b}_{\text{ub}} = \begin{bmatrix} S_{\max}\Delta T \\ \vdots \\ S_{\max}\Delta T \end{bmatrix}$$

Quadratic Programming

Smooth gradient $\rightarrow$ Minimize the curvature

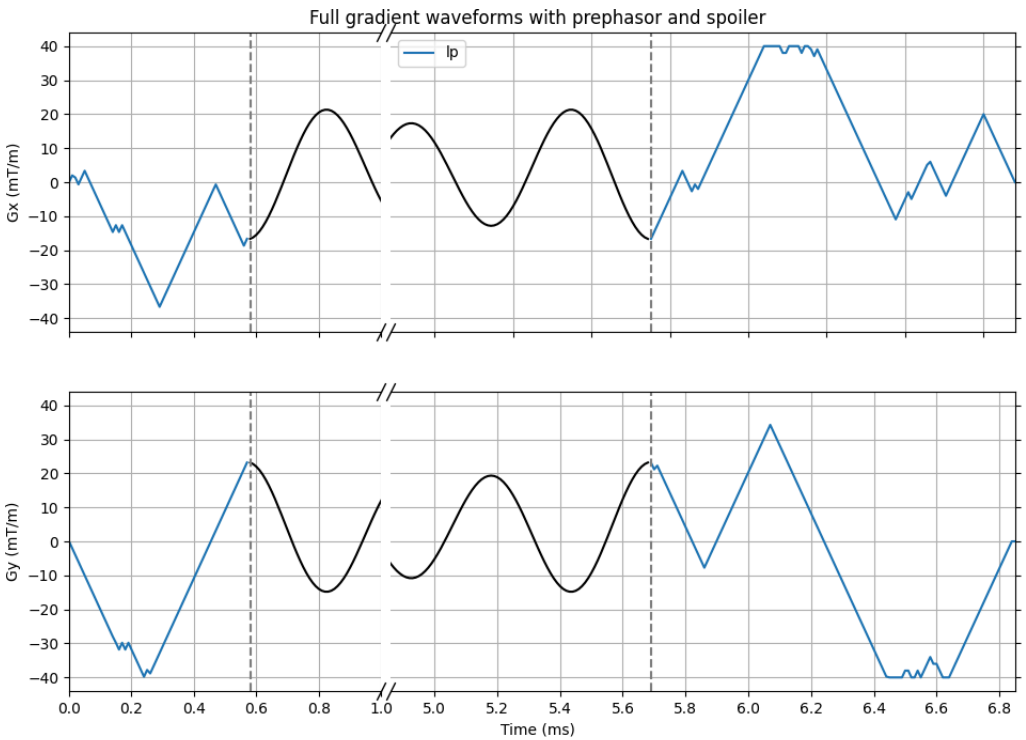
$$\min_{\boldsymbol{g}\in\mathbb{R}^N} \sum_{i=0}^{N-1} \left(g_{i+1} - 2g_i + g_{i-1}\right)^2 \text{ s.t. } g_n \in \mathcal{Q}$$

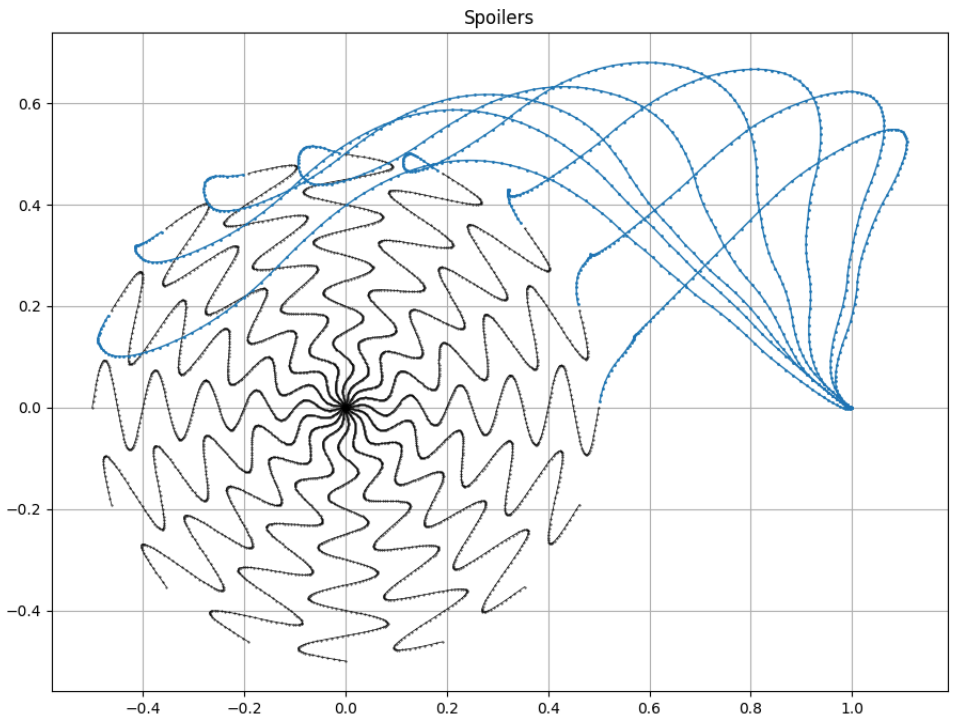
$$\min_{\boldsymbol{g}\in\mathbb{R}^N} \frac{1}{2}\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g} + \boldsymbol{q}^T\boldsymbol{g} + \boldsymbol{c} \text{ s.t. } \boldsymbol{l} \leq \boldsymbol{A}\boldsymbol{g} \leq \boldsymbol{u}$$

- There exist an N_0 s.t. $\forall N \geq N_0, \Delta k \in \left\{\sum_n^N g_n \mid g_n \in \mathcal{Q}\right\}$
- Finding N_0 is done with binary search
  - Is it feasibile $\Leftrightarrow$ Solve it
  - Engineer trick: quantize the constraints for multi-shot setting ($N_{\text{shots}} \times 3 \rightsquigarrow 100$-ish cases)
  - Same mini-max N for all shots (same TR !)

Connecting the dots

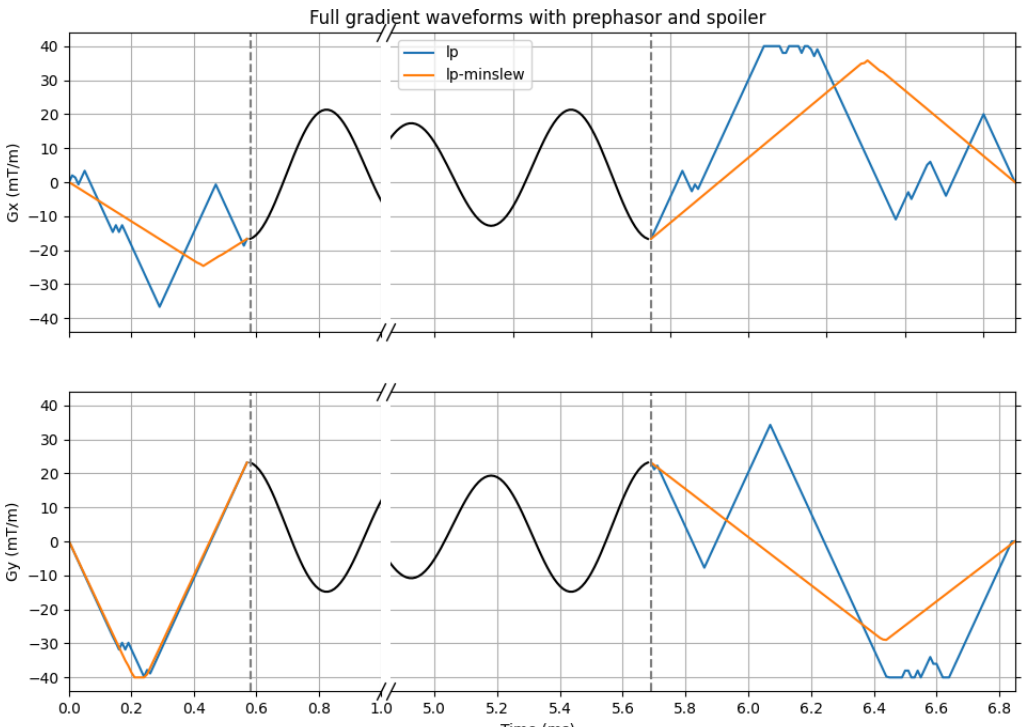
Linear Programming

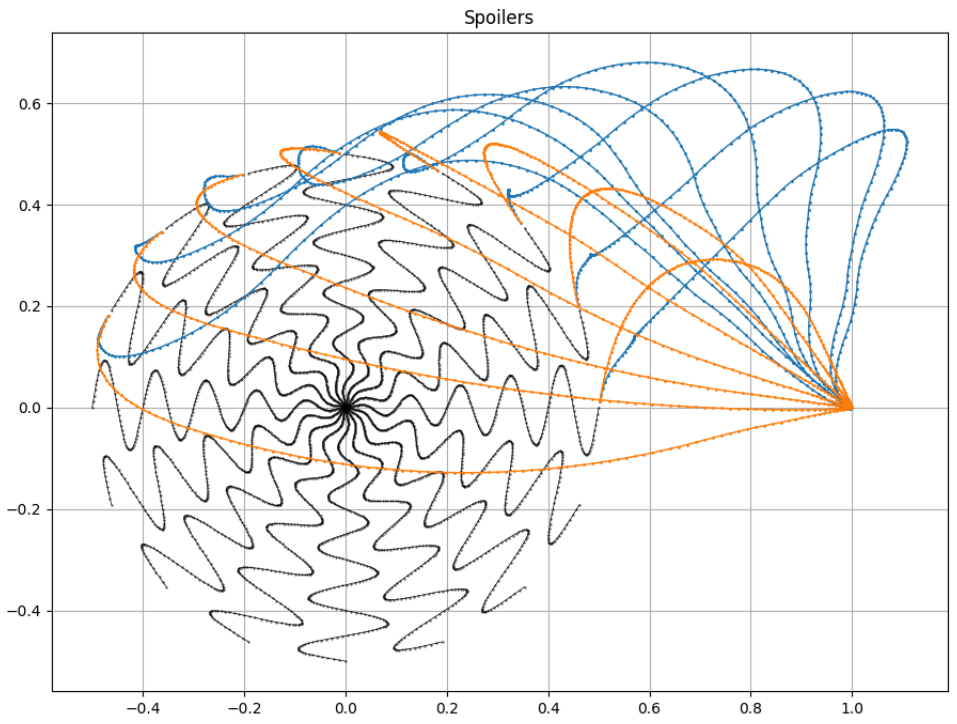




Connecting the dots

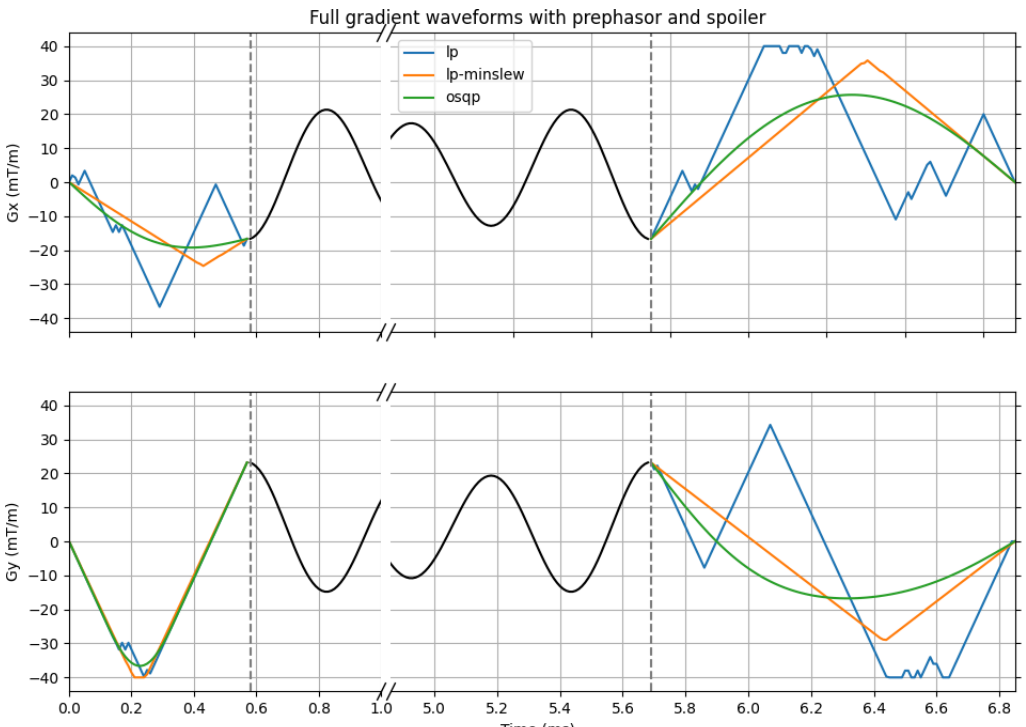
Linear Programming + Mini-Max Slew Rate (binary searched again)

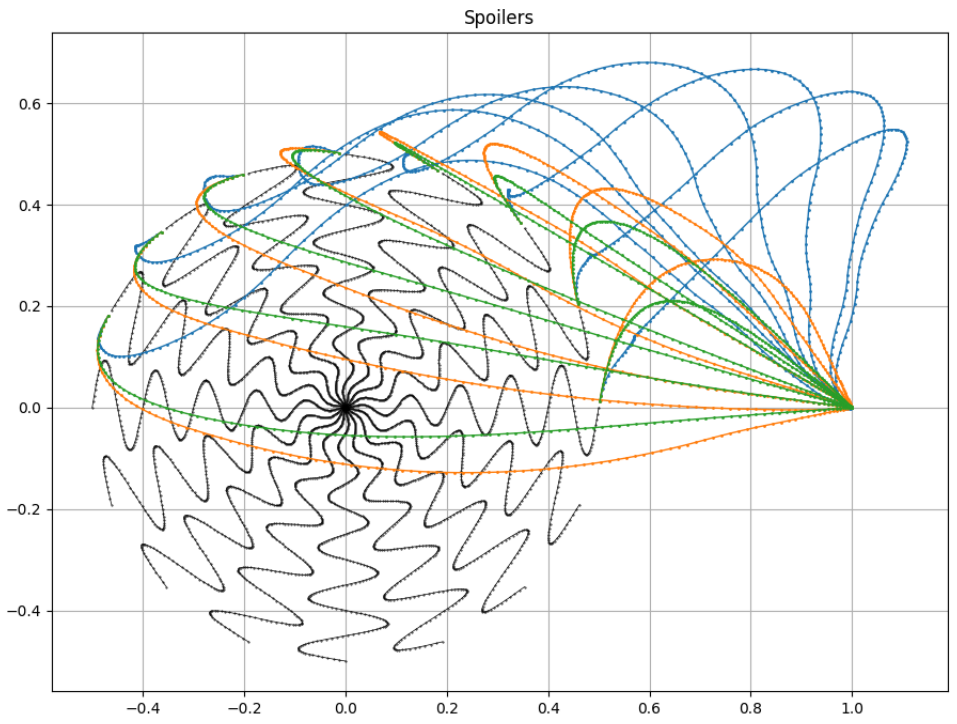




Connecting the dots

Quadratic Programming





4. The toolbox part

Integration with other libraries

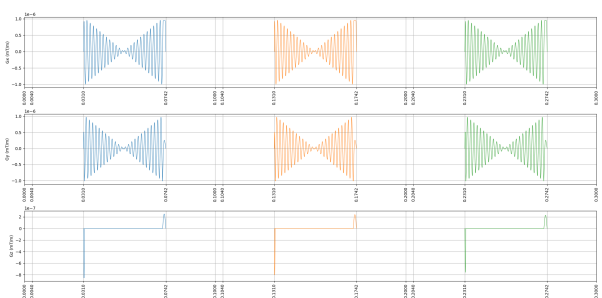
- $\geq$ 20 Projects depends on MRI-NUFFT[1]
- Integration with external packages

[1] https://github.com/mind-inria/mri-nufft/network/dependents

Integration with other libraries

- $\geq$ 20 Projects depends on MRI-NUFFT[1]
- Integration with external packages
  - (Py)Pulseq

```python
from mrinufft.trajectories import (pulseq_gre,
    read_pulseq_traj)
traj = ...
seq = pulseq_gre(traj, acq=acq)
seq.write("spiral_gre.seq")
traj_read = read_pulseq_traj(seq)
```

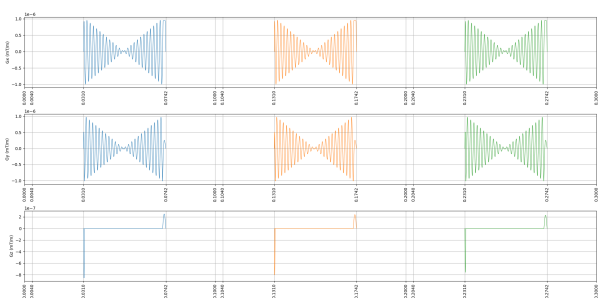


[1] https://github.com/mind-inria/mri-nufft/network/dependents

Integration with other libraries

- $\geq$ 20 Projects depends on MRI-NUFFT[1]
- Integration with external packages
  - (Py)Pulseq
  - DeepInverse

```python
from mrinufft.trajectories import (pulseq_gre,
    read_pulseq_traj)
traj = ...
seq = pulseq_gre(traj, acq=acq)
seq.write("spiral_gre.seq")
traj_read = read_pulseq_traj(seq)
```



Example for integration with `deepinv`

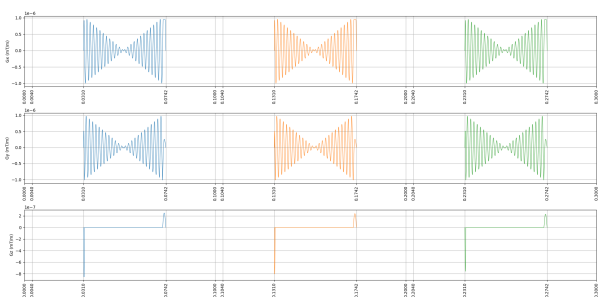
```python
from mrinufft import get_operator
from deepinv.optim import WaveletPrior, L2, optim_builder
# MRI-NUFFT setup
fourier_op = get_operator("cufinufft",samples_loc,shape=shape)
y = fourier_op.op(ground_truth) + noise  # y = Ax + ε
```

[1] https://github.com/mind-inria/mri-nufft/network/dependents

Integration with other libraries

- $\geq$ 20 Projects depends on MRI-NUFFT[1]
- Integration with external packages
  - (Py)Pulseq
  - DeepInverse

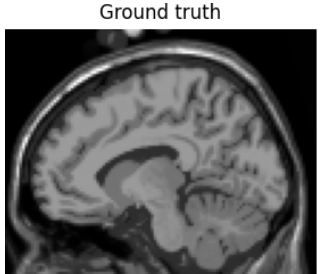
```python
from mrinufft.trajectories import (pulseq_gre,
    read_pulseq_traj)
traj = ...
seq = pulseq_gre(traj, acq=acq)
seq.write("spiral_gre.seq")
traj_read = read_pulseq_traj(seq)
```



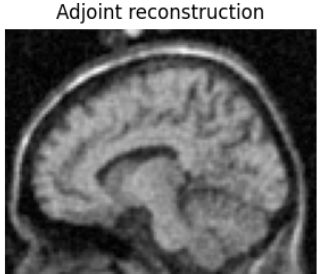
Example for integration with `deepinv`

```python
from mrinufft import get_operator
from deepinv.optim import WaveletPrior, L2, optim_builder
# MRI-NUFFT setup
fourier_op = get_operator("cufinufft",samples_loc,shape=shape)
y = fourier_op.op(ground_truth) + noise  # y = Ax + ε

# DeepInverse setup
physics = fourier_op.make_deepinv_phy()
wavelet = WaveletPrior(wv="sym8", is_complex=True, p=1) # ‖Ψ · ‖_1
wavelet_fista = optim_builder(
    iteration="FISTA",
    prior=wavelet, # swap with ANY prior !
    data_fidelity=L2(),
    params_algo={"lambda": 0.1}
)

# Reconstruction
x_pinv  = physics.A_dagger(y) # arg min_x ‖Ax − y‖_2^2
x_wavelet = wavelet_fista(y, physics) # arg min_x ‖Ax − y‖_2^2 + λ‖Ψx‖_1^1
```

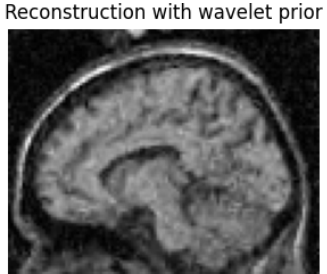
Ground truth



Adjoint reconstruction



Reconstruction with wavelet prior



[1] https://github.com/mind-inria/mri-nufft/network/dependents

Conclusion

You can use MRI-NUFFT for:

- **Fast and efficient NUFFT** operators for non-Cartesian MRI reconstruction

Conclusion

You can use MRI-NUFFT for:

- **Fast and efficient NUFFT** operators for non-Cartesian MRI reconstruction
  - To be used inside your reconstruction libraries

Conclusion

You can use MRI-NUFFT for:

- **Fast and efficient NUFFT** operators for non-Cartesian MRI reconstruction
  - To be used inside your reconstruction libraries
- Testing a **large collection** of ready-to-use **non-Cartesian trajectories**

Conclusion

You can use MRI-NUFFT for:

- **Fast and efficient NUFFT** operators for non-Cartesian MRI reconstruction
  - To be used inside your reconstruction libraries
- Testing a **large collection** of ready-to-use **non-Cartesian trajectories**
- Augmenting, and adapt/constraint these trajectories.

Conclusion

You can use MRI-NUFFT for:

- **Fast and efficient NUFFT** operators for non-Cartesian MRI reconstruction
  - To be used inside your reconstruction libraries
- Testing a **large collection** of ready-to-use **non-Cartesian trajectories**
- Augmenting, and adapt/constraint these trajectories.
- Acquire with `(Py)Pulseq`, Reconstruct with Deepinv, or any other (Python) toolbox!

Conclusion

You can use MRI-NUFFT for:

- **Fast and efficient NUFFT** operators for non-Cartesian MRI reconstruction
  - To be used inside your reconstruction libraries
- Testing a **large collection** of ready-to-use **non-Cartesian trajectories**
- Augmenting, and adapt/constraint these trajectories.
- Acquire with `(Py)Pulseq`, Reconstruct with Deepinv, or any other (Python) toolbox!

https://github.com/mind-inria/mri-nufft

https://mind-inria.github.io/mri-nufft

Comby et al., (2025). *MRI-NUFFT: Doing non-Cartesian MRI has never been easier*. Journal of Open Source Software, 10(108), 7743, https://doi.org/10.21105/joss.07743

Check out Our *awesome* **open-source** software:

MRI-nufft

SNAKE

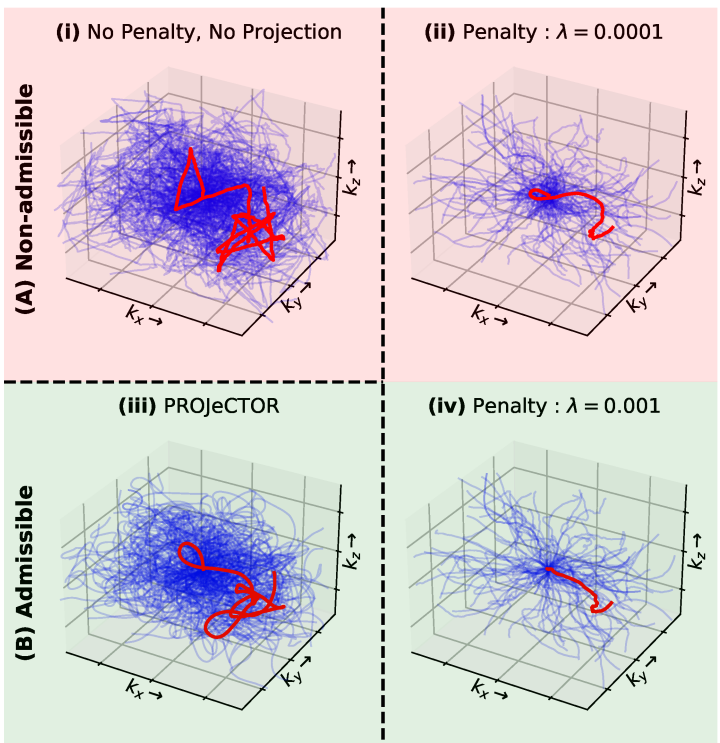
Thank you for your attention

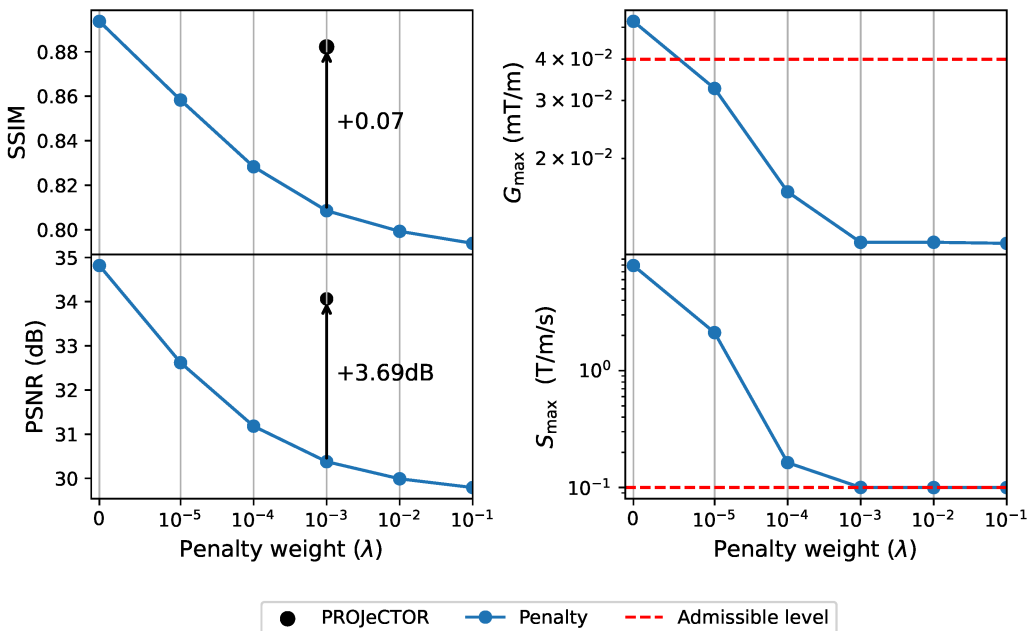
pierre-antoine.comby@cea.fr

Neurospin, CEA, Paris-Saclay University

Projection vs penalty for trajectory design

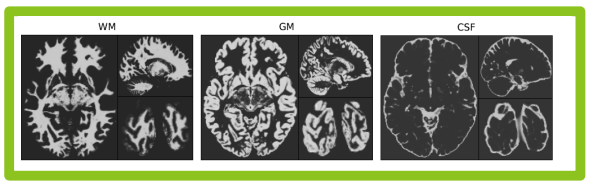
- Context: Learning the trajectory $\partial \frac{A}{\partial} \Omega$ (+Joint reconstruction network)
- *Radhakrishna&Ciuciu*, Bioeng., 2023.





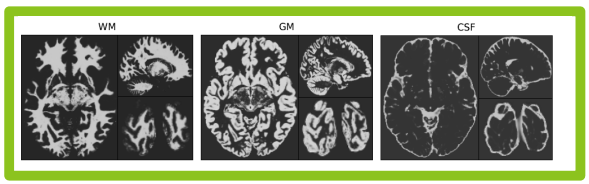
SNAKE fMRI k-space Simulator

Brain, B_0, Activated Regions, ...

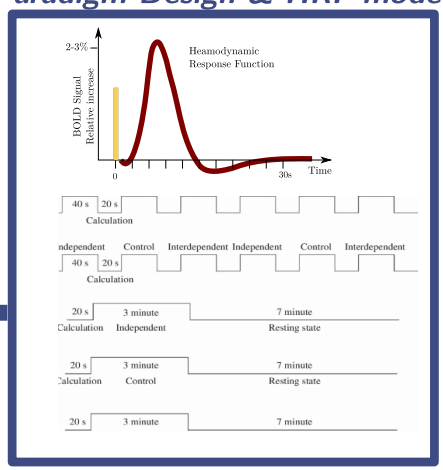


SNAKE fMRI k-space Simulator

Brain, B_0, Activated Regions, ...



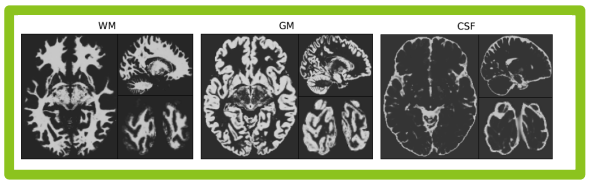
Paradigm Design & HRF model



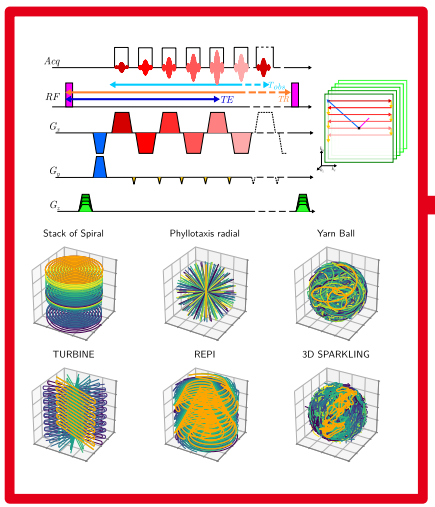
SNAKE

SNAKE fMRI k-space Simulator

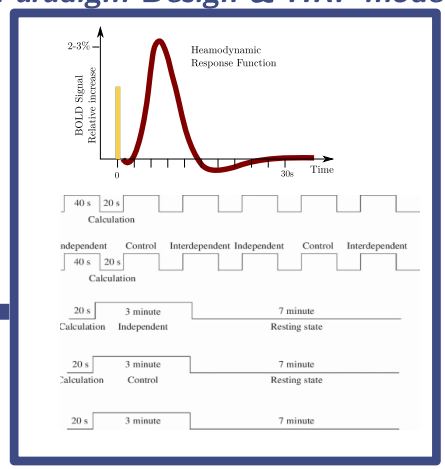
Brain, B_0, Activated Regions, ...



Acquisition Setting

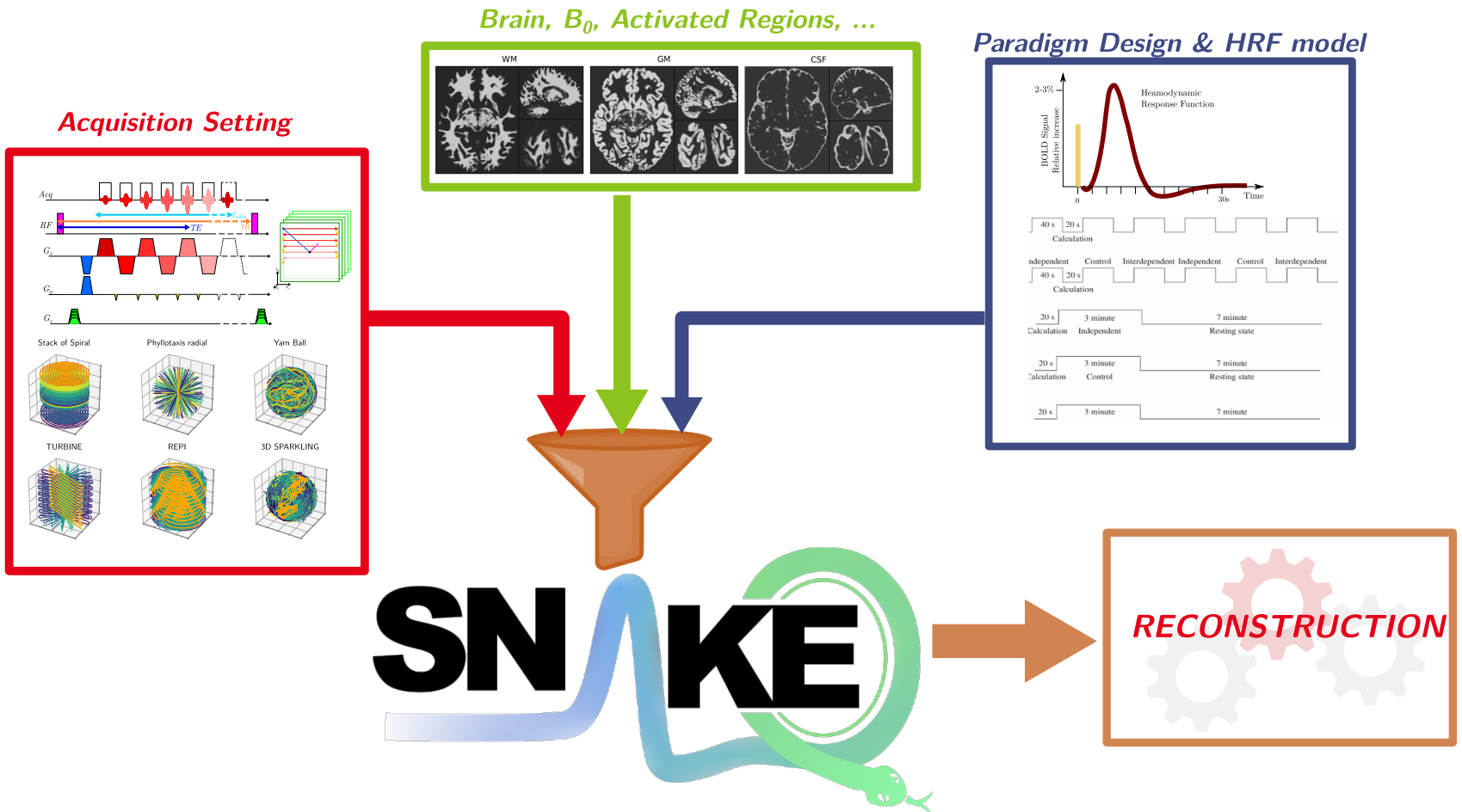


Paradigm Design & HRF model



SNAKE

SNAKE fMRI k-space Simulator

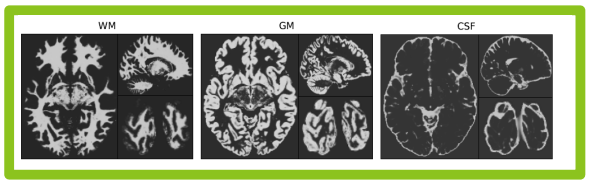


SNAKE fMRI k-space Simulator

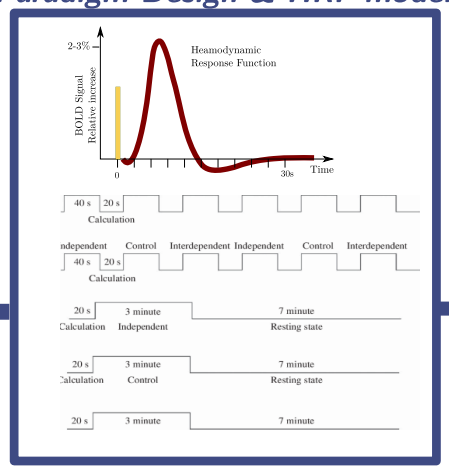
Learn more

Comby et al., ImagNeuro, 2025.

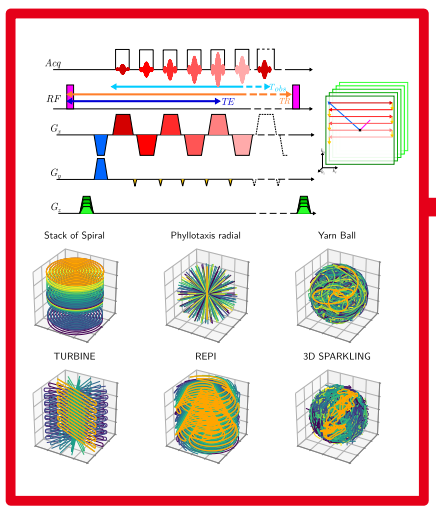
Brain, B_0, Activated Regions, ...



Paradigm Design & HRF model



Acquisition Setting



SNAKE

RECONSTRUCTION

STATISTICAL ANALYSIS VALIDATION