ToFu

An open-source python/cython library for synthetic tomography diagnostics on tokamaks

Laura S. Mendoza[1], Didier Vezinet[2]

Euroscipy 2019, Bilbao, España

[1]INRIA Grand-Est, TONUS Team, Strasbourg, France

[2]CEA, Cadarache, France

Table of contents

Context

Context: energy needs vs resources and climate change




Worldwide growing energy needs (population, standards of living...)

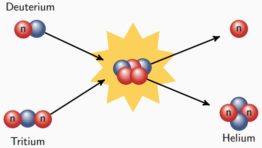
$\Rightarrow$ high pressure on environment (degrading, changing, exhausting...)

$\Rightarrow$ decrease consumption + alternative production means

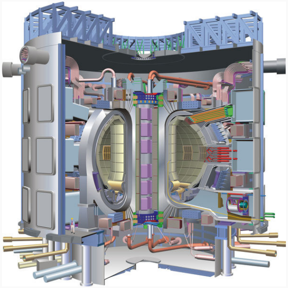
$\Rightarrow$ relatively clean, safe, mass-production means with large resources would be welcome in the mix

Context: Controlled fusion and magnetic confinement

D-T Fusion reaction



- Gas $>$ 100 Million$^\circ$K composed of positive ions and negative electrons: plasma
- Confinement using electromagnetic fields
- break-even not obtained yet



In a nutshell: toroidal vacuum vessel, filled with H plasma

Tomography diagnostics

Tokamak diagnostics to measure plasma quantities

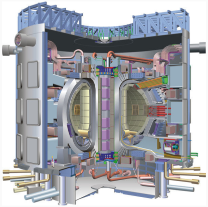
Diagnostics

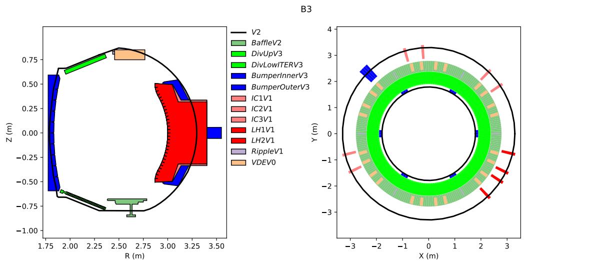
Set of instruments to measure plasma quantities, for understanding, control, optimization.

e.g: magnetic field, neutrons, **emitted light**, temperature, density...

$\Rightarrow$ pinhole cameras (1D or 2D) for measuring light in various wavelengths

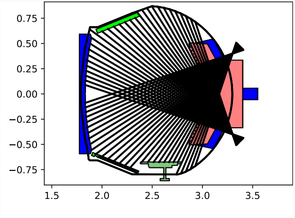
A tokamak as a poloidal + horizontal projections





Tomography diagnostics - numerical context

$$M_i(t) = \iiint\limits_{V_i} \overrightarrow{\varepsilon(x,t)} \cdot \vec{n}\, \Omega_i \, dV$$



- **Direct problem** (synthetic diagnostic):

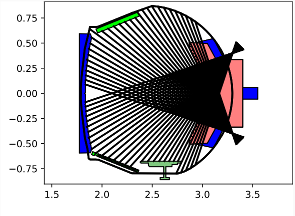
  Simulated emissivity $\longrightarrow$ integrated measurements

- **Inverse problem** (tomography):

  Integrated measurements $\longrightarrow$ Reconstructed emissivity

Tomography diagnostics - numerical context

$$M_i(t) = \iiint_{V_i} \overrightarrow{\varepsilon(x,t)} \cdot \vec{n}\, \Omega_i \, dV$$



- **Direct problem** (synthetic diagnostic):

  Simulated emissivity $\longrightarrow$ measurements

  Spatial integration

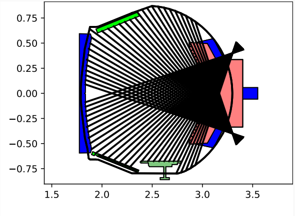
- **Inverse problem** (tomography):

  Integrated measurements $\longrightarrow$ Reconstructed emissivity

  Mesh and basis functions construction, spatial integration, data filtering, inversion routines, etc.

Tomography diagnostics - numerical context

$$M_i(t) = \iiint_{V_i} \overrightarrow{\varepsilon(x,t)} \cdot \vec{n}\, \Omega_i \, dV$$



- **Direct problem** (synthetic diagnostic):

  Simulated emissivity $\longrightarrow$ measurements

  Spatial integration

- **Inverse problem** (tomography):

  Integrated measurements $\longrightarrow$ Reconstructed emissivity

  Mesh and basis functions construction, spatial integration, data filtering, inversion routines, etc.

Tomography is **ill-posed**, very sensitive to errors, noise and bias

$\longrightarrow$ Reputation for low reproducibility / reliability

The ToFu package

Motivation: “current” state

In the fusion community, codes for tomography diagnostic are often:

- developed by physicists (with little programming experience)
- in Matlab (or IDL)
- written from scratch, re-done by new students
- not distributed (few users), rarely documented

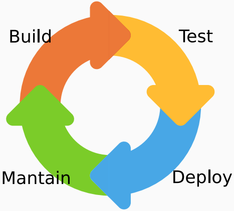
... which means

- waste of resources: time, man-power
- low traceability, reproducibility
- low standardization, unclear assumptions / methods

A code for Tomography for Fusion

Develop a common tool:

- Generic (geometry independent)
- Portable (Python)
- Optimized / parallelized
- Documented online
- Continuous integration



ToFu[1,2,3] = **To**mography for **Fu**sion

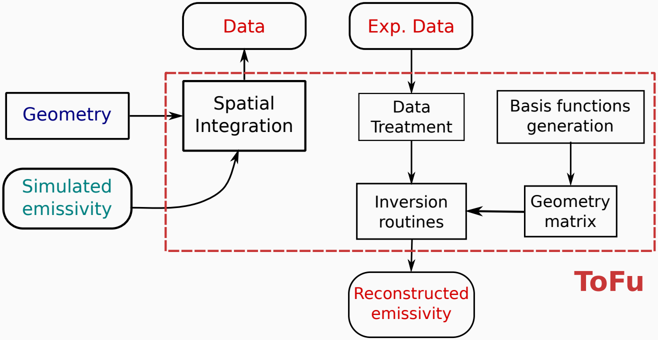
[1] repository: https://github.com/ToFuProject/tofu
[2] documentation: https://tofuproject.github.io/tofu/index.html
[3] D Vezinet et al. “Non-monotonic growth rates of sawtooth precursors evidenced with a new method on ASDEX Upgrade”. In: *Nuclear Fusion* 8 (2016).

More about Tofu

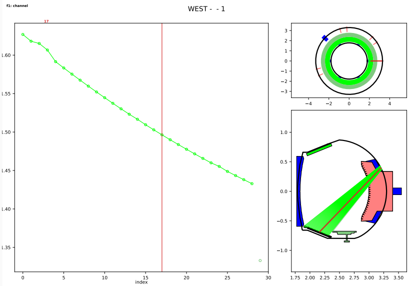
- Created in 2014
- Open Source: **MIT license**
- Python 2.7 and **Python** 3 + **Cython**
- Continuous integration: **Travis** CI
- **conda**, **pip**
- Two (main) developers:
  - Didier Vezinet (creator, Physics)
  - Laura S. Mendoza (since 06.2018, Applied Maths)
- Contributors:
  - Jorge Morales
  - Florian Le Bourdais
  - Arpan Khandelwal

Travis-CI passing | Anaconda Cloud 1.4.0 | downloads 55 total | Last updated 02 May 2019 | Platforms linux-64 | license MIT | Install with conda
codecov 55% | pypi package 1.4.0

Tofu's structure

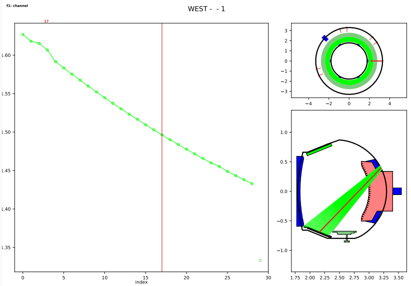


1D Camera

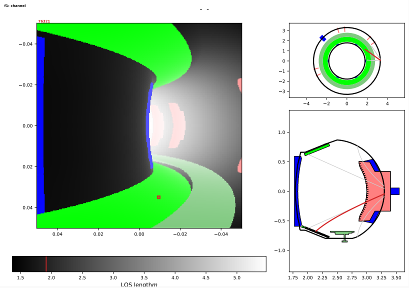


tofu.geom: modeling of simplified geometry

1D Camera

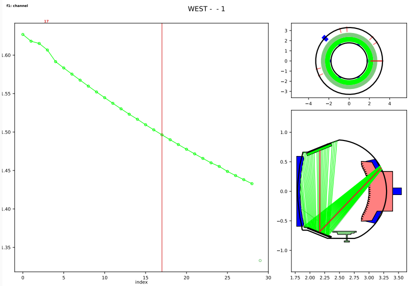


2D Camera

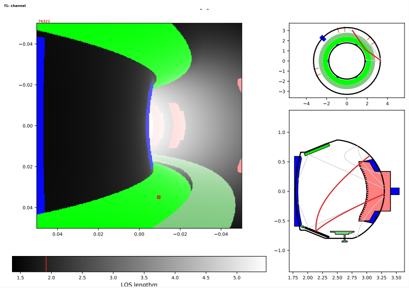


tofu.geom: handle basic reflexions

1D Camera with reflexions



2D Camera with reflexions



What ToFu can do

ToFu can:

- Modeling of simplified geometry
- 3D modeling of a 1D camera
- 3D modeling of a 2D camera
- Handle basic reflexions
- Computing synthetic signals
- ...and soon:
    - meshing and basis functions
    - tomographic inversion
    - dust particle trajectory tracking
    - faster Matplotlib + PyQtGraph visualization
    - magnetic field line tracing
    - data visualization and statistical tools (pandas)

Demo

Demo

Code Optimization

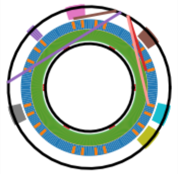
Geometry reconstruction: ray-tracing techniques

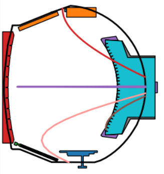
To reconstruct emissivity we need to take account:

- Up to hundreds of structural elements in vessel
- Scale of the vessel: 10^4 bigger than smaller structural detail

$\Rightarrow$ Geometry defined with minimal data polygon (R, Z) extruded along φ

$\Rightarrow$ Symmetry of vessel along φ



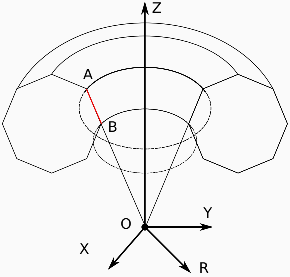


Optimization of ray-tracing algorithm

- Description of geometry:
  - Vessel and structures: set of 2D polygon $\mathcal{P}_j = \cup_{i=1}^{n} \overline{\mathrm{A}_i\mathrm{B}_i}$
  - Extruded along $[\varphi_{min}, \varphi_{max}]$
  - Detectors defined as set of rays (of origin D and direction u)

  $\Rightarrow$ Light memory-wise

$\Rightarrow$ Equivalent to: set of truncated cones (frustums) of generatrix A_iB_i



Ray-tracing algorithm on fusion device $\longrightarrow$ Computation of cone-Ray intersection

$$\exists(q,k) \in [0;1] \times [0;\infty[, \quad \begin{cases} R - R_A = q(R_B - R_A) \\ Z - Z_A = q(Z_B - Z_A) \\ DM = ku \end{cases}$$

Optimization of ray-tracing algorithm

Cone-Ray intersection algorithm:

- Main steps:
  - Test intersection **bounding-box**
  - **special cases** (ray direction, segment, etc.)
  - General case: solution of a **quadratic equation**
- Pre-computation of variables
- Core functions in **Cython**
- Parallelization (**prange** loops)

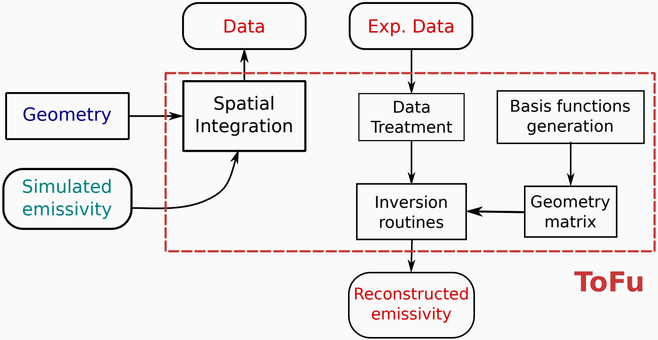
Optimization of ray-tracing algorithm

Cone-Ray intersection algorithm:

- Main steps:
  - Test intersection **bounding-box**
  - **special cases** (ray direction, segment, etc.)
  - General case: solution of a **quadratic equation**
- Pre-computation of variables
- Core functions in **Cython**
- Parallelization (**prange** loops)

Nb LOS	10^3	10^4	10^5	10^6	
original	$3.26 \cdot 10^1$	$3.10 \cdot 10^2$	$3.20 \cdot 10^3$	$3.17 \cdot 10^4$	(8h48)
optimized	$2.58 \cdot 10^{-2}$	$2.72 \cdot 10^{-1}$	2.74	$2.66 \cdot 10^1$	($<$ 30s)
32 threads	$1.36 \cdot 10^{-2}$	$4.66 \cdot 10^{-2}$	$3.64 \cdot 10^{-1}$	2.92	

Tofu's structure



Optimization of spatial integration routines

Integration of **user-defined function** along a LOS:

- Integration of a python function **func** defined by user by:
  - **numpy**.sum (quad: **midpoint**)
  - Cython based sum (quad: **midpoint**)
  - **Scipy**.integrate.simps
  - **Scipy**.integrate.romb
- Optional optimizations:
  - calls to **func**: avoid Cython-Python conversion, user-defined
  - memory: fine resolutions, high number of LOS
  - hybrid: compromise

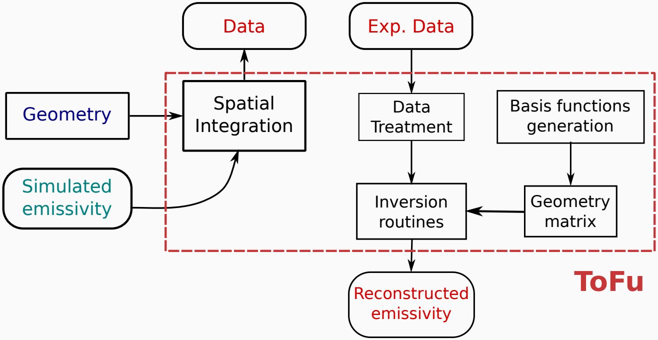
Optimization of spatial integration routines

LOS	10	10^2	10^3	10^4
original	0.46	2.24	18.1	x
memory	0.9	8.9	96	945 (6Gb)
calls	0.207	0.53	4.32	x
hybrid	0.08	0.44	4.2	40.3 (32Gb)

- Space resolution: 10^{-3}
- Number of time steps: 10^3
- Integration method: **sum** (Cython or numpy) on midpoint

What’s next

Tofu's structure



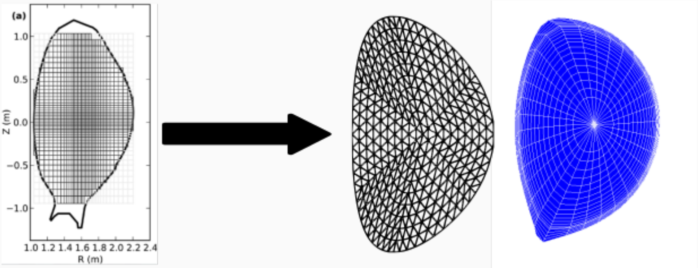
On geometry discretization: meshing

Several options for poloidal cut meshing:

- Cartesian mesh
- Polar mesh
- Adaptive polar mesh
- Hexagonal mesh
- Triangular mesh

For basis functions:

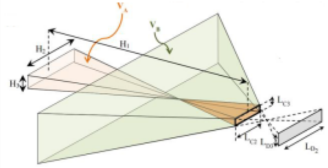
- Lagrange polynomials
- B-splines
- NURBS
- Box-splines



Tofu's main algorithms

Geometry:

- Finite beam width (LOS $\Rightarrow$ VOS)
- More advanced reflexions



Meshing and Inversions:

- Meshing and Basis functions with visualization
- Multiple Inversion-Regularization and visualization

Thank you for your attention!

B(asis)-Splines basis*

B-Splines of degree d are defined by the **recursion** formula:

$$B_j^{d+1}(x) = \frac{x - x_j}{x_{j+d} - x_j} B_j^d(x) + \frac{x_{j+1} - x}{x_{j+d+1} - x_{j+1}} B_{j+1}^d(x) \qquad (1)$$

Some important properties about B-splines:

- Piece-wise polynomials of degree d $\Rightarrow$ **smoothness**
- Compact support $\Rightarrow$ **sparse matrix system**
- Partition of unity $\sum_j Bj(x) = 1, \forall x$ $\Rightarrow$ **conservation laws**