wakis:
3D Electromagnetic Time- Domain Wake and Impedance Solver

ABP-CEI Meeting, 25th April 2024

Elena de la Fuente, Lorenzo Giacomel, Giovanni Iadarola, Carlo Zannini, Manuel Cotelo (UPM)

☺ Special thanks to: Chiara Antuono, Giovanni Rumolo, Benoit Salvant, Leonardo Sito, Jean Luc Vay (Berkeley Lab) and yet again to Lorenzo Giacomel

Outline

CERN IFN-GV

Outline

Beam Coupling Impedance $Z(f)$

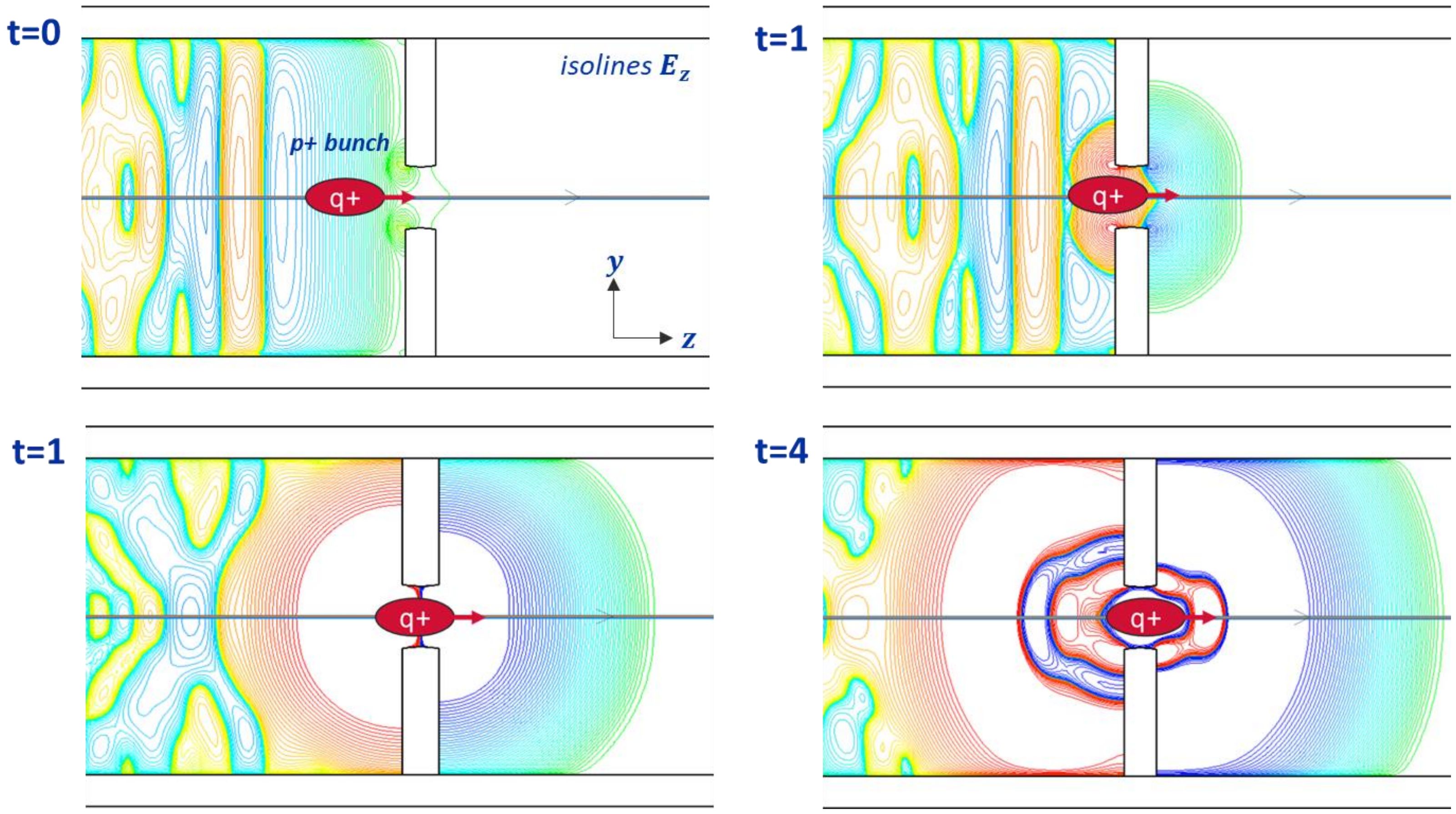


Fig: 3D Time domain simulation of a smooth pipe with obstacle with a passing proton beam (1-4) with CST®

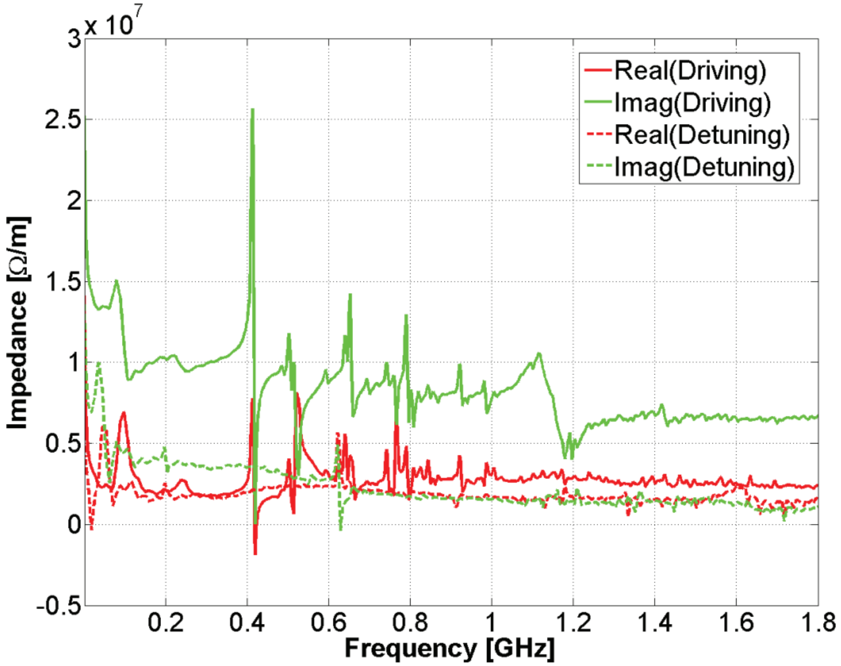
Wakefields are generated as the particle beam traverses the different accelerator devices and 'perceives' **discontinuities:**

- in the geometry
- or the electromagnetic properties (ε, μ, σ...)

These wakefields will affect the trailing particles/bunches. The ***beam-coupling impedance*** is the frequency-dependent property of each accelerator device, used to quantify the wakefields' effects.

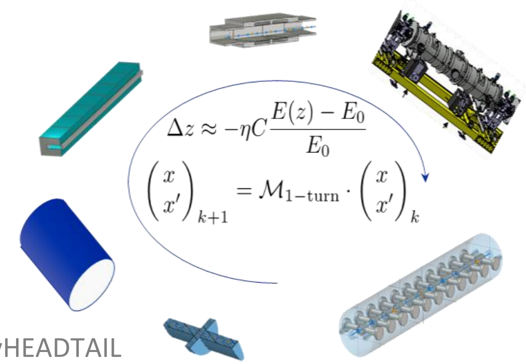
Beam Coupling Impedance $Z(f)$ (II)

- The impedance of all relevant accelerator components is gathered in each accelerator's impedance model

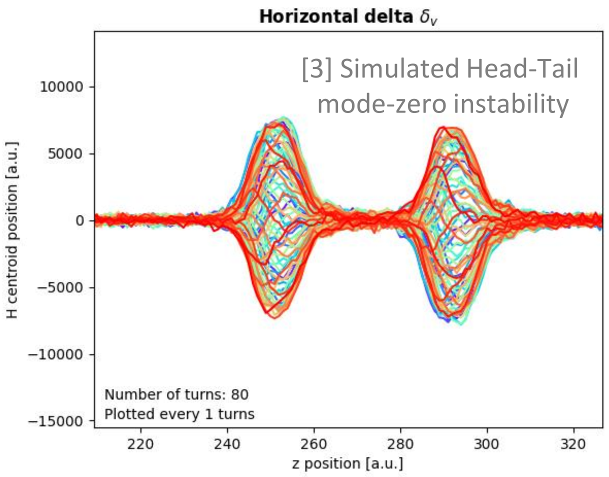


[1] SPS Transverse vertical impedance model.
C. Zannini

- And used in beam-dynamics codes (pyHeadTail, Xsuite) to predict beam behaviour

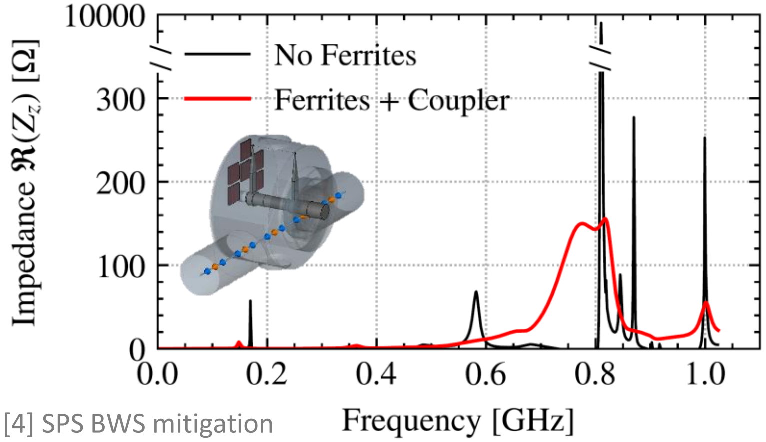


[2] PyHEADTAIL concept schematic.
K. Li & G. Rumolo

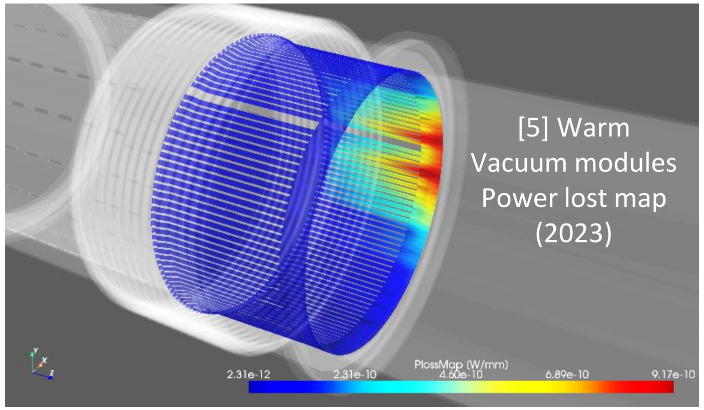


[3] Simulated Head-Tail mode-zero instability

- Or assess beam-induced heating of individual accelerator components and propose mitigation solutions



[4] SPS BWS mitigation solution (2023)



[5] Warm Vacuum modules Power lost map (2023)

How to obtain $Z(f)$?

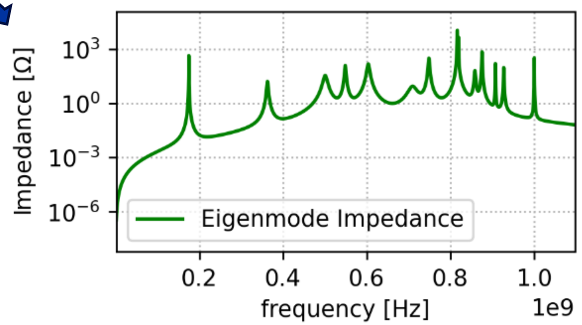
Directly in frequency domain (FD)

- **Eigenmode solver (CST®):**

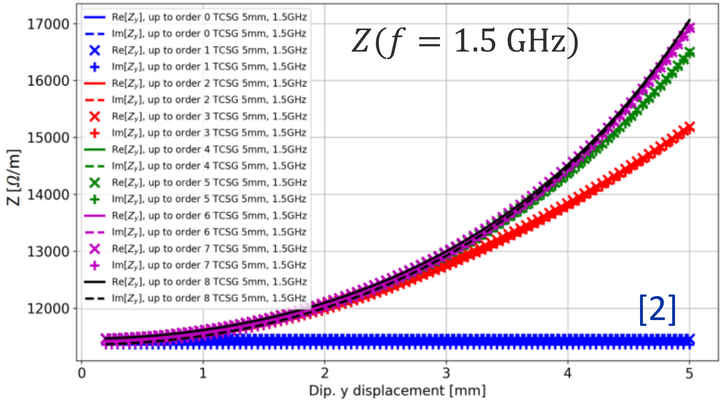
Mode	Frequency
1	8.02724981695
2	10.1781072579
3	10.3554337227
4	12.5109337728
5	14.7562186034

f, R_s, Q

Reconstructed $Z(f)$



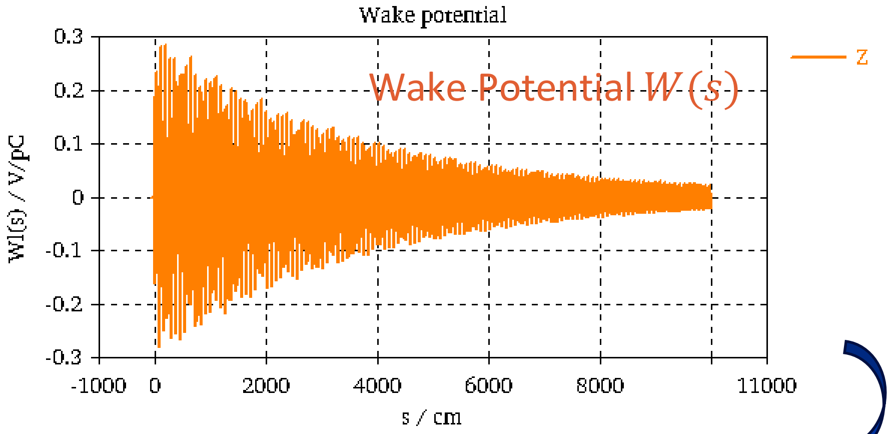
- **Resistive wall impedance: (IW2D)** [1]

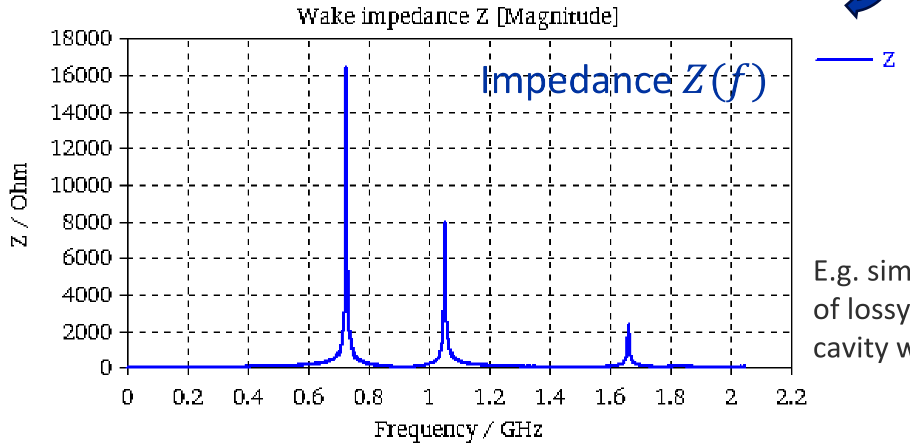


E.g. vertical TCSG CFC collimator vertical dipolar impedance at $f = 1.5$ GHz vs. vertical source offset

From the time domain (TD)

- **Wakefield solver (CST®):**





E.g. simulation of lossy pillbox cavity with CST®

[1] N. Mounet. The LHC Transverse Coupled-Bunch Instability, PhD thesis 5305 (EPFL, 2012)
[2] N. Mounet. ImpedanceWake2D BE-ABP-CEI 22/04/2021

How to obtain $Z(f)$?

Commercial / not open-source codes

Directly in frequency domain (FD)

- **Eigenmode solver (CST®):**

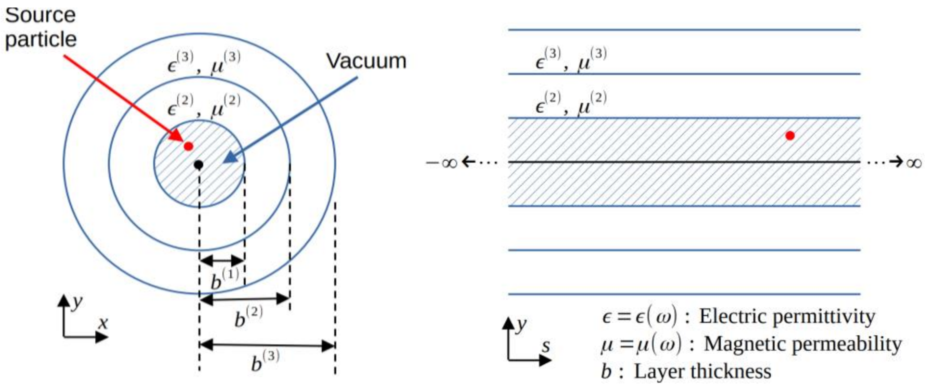
$$rot\,\vec{E} = j\omega\mu\vec{H}$$
$$rot\,\vec{H} = j\omega\left(\varepsilon + \frac{\sigma}{j\omega}\right)\vec{E}$$

Eigenvalue problem

$$rot\,\frac{1}{\underline{\underline{\mu}}}\,rot\,\vec{E} = \underline{\underline{\varepsilon}}\,\omega^2\vec{E}$$

Fundamental EM modes of 3D loss-less resonant structures without excitation + Q-factor postprocessing

- **Resistive wall impedance: (IW2D@CERN)** [1]



From the time domain (TD)

- **Wakefield solver (CST®):**

3D time domain Maxwell equations (integral form)

$$\oint_{\partial A} \boldsymbol{E}\cdot d\boldsymbol{s} = -\iint_A \frac{\partial \boldsymbol{B}}{\partial t}\cdot d\boldsymbol{A}$$
$$\oint_{\partial A} \boldsymbol{H}\cdot d\boldsymbol{s} = -\iint_A \left(\frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J}\right)\cdot d\boldsymbol{A}$$
$$\oiint_{\partial V} \boldsymbol{B}\cdot d\boldsymbol{A} = 0$$
$$\oiint_{\partial V} \boldsymbol{D}\cdot d\boldsymbol{A} = \iiint_V \boldsymbol{\rho}\, d\boldsymbol{V}$$

with beam excitation $\boldsymbol{J} = v_z\lambda(z)\boldsymbol{e_z}$

+ wake potential and impedance computation:

$$\boldsymbol{W}(r_1, r_2, s) = \frac{1}{q_1}\int_{-\infty}^{\infty} dz \left[\vec{E}(r_1, r_2, z, t) + c\vec{e_z}\times\vec{B}(r_1, r_2, z, t)\right]_{t=\frac{(s+z)}{c}}$$

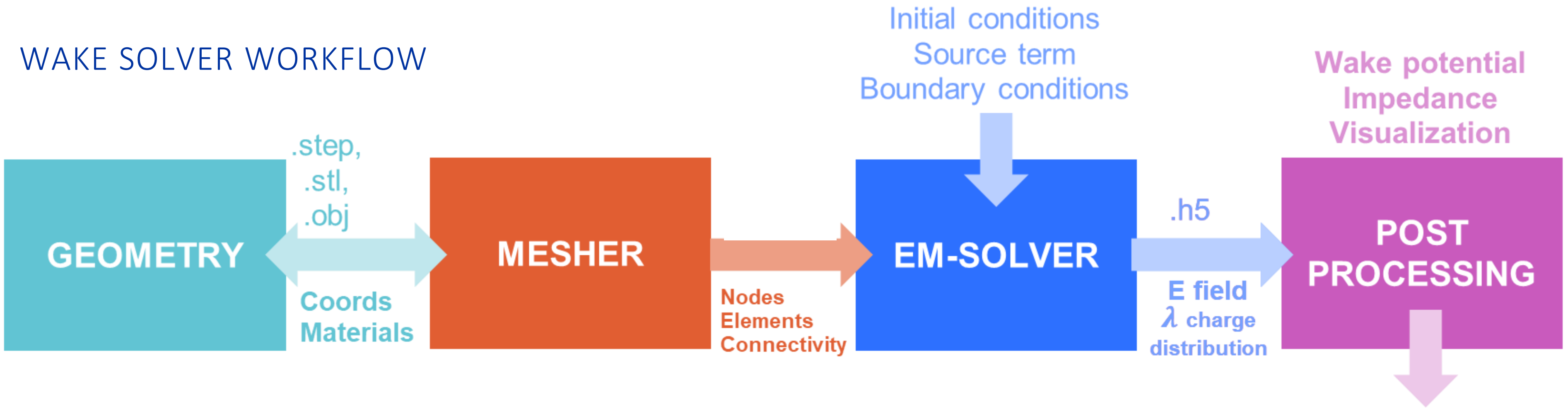
$$\boldsymbol{Z}_{||}(\omega) = -\frac{\int_{-\infty}^{\infty} W(s)e^{-i\omega s}ds}{\int_{-\infty}^{\infty} c\lambda(s)e^{-i\omega s}ds}$$ (FT + deconvolution)

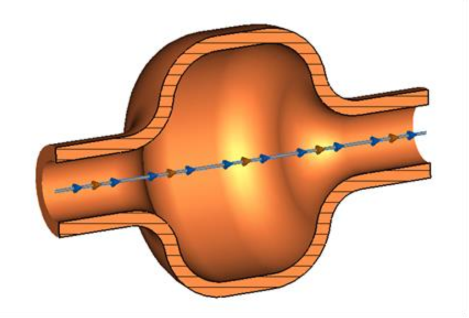
[1] N. Mounet. The LHC Transverse Coupled-Bunch Instability, PhD thesis 5305 (EPFL, 2012)

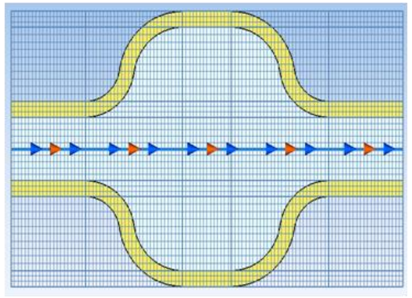
Project objective

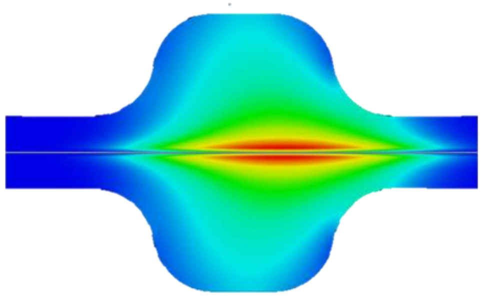
Develop an **open-source Wakefield Solver** (i.e., 3D electromagnetic time-domain) at **CERN**

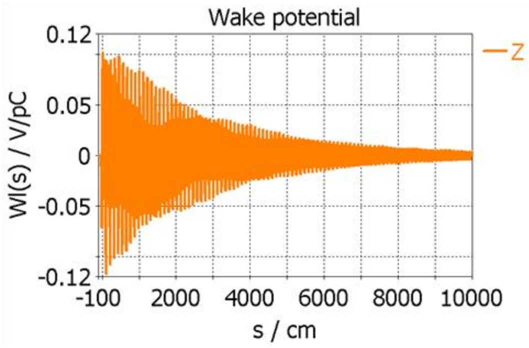
WAKE SOLVER WORKFLOW

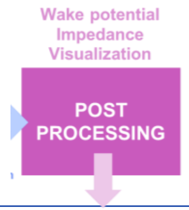












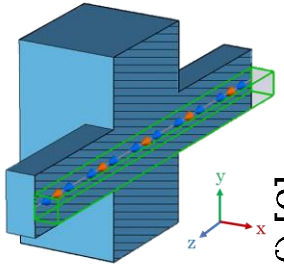
PhD motivation

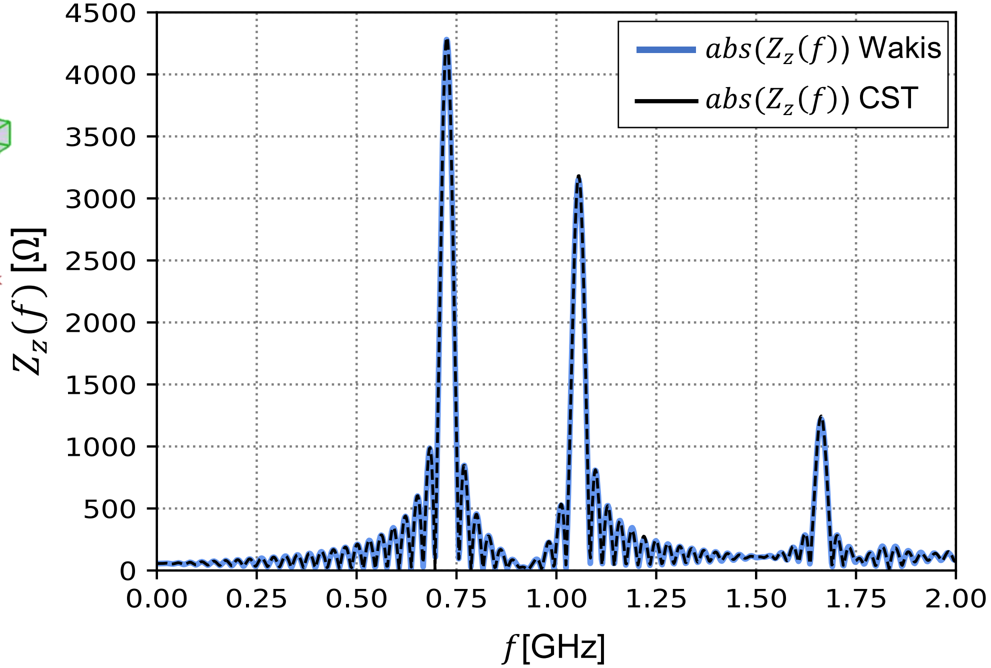
Trainee | PhD

1st year	2nd year	1st year	2nd year	3rd year

(v0.1) Developed wakis package:

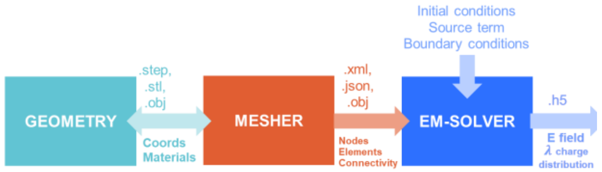
- Computation of wake potential and impedance from pre-computed fields
- Longitudinal $Z_{||}$ and transverse $Z_{\perp}$ (dipolar & quadrupolar)
- Benchmarked with CST® Wakefield fields

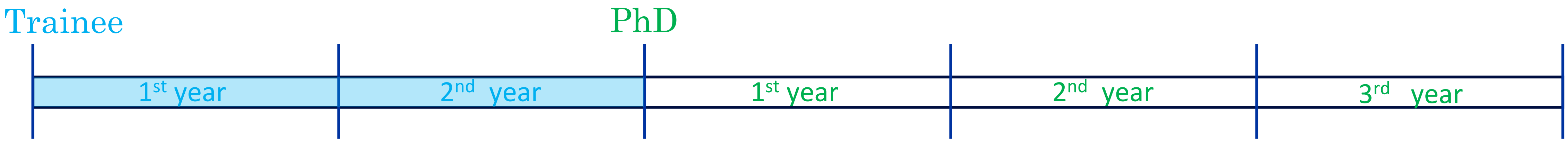




[1] *Progress and challenges of an in-house wake/impedance solver*. ABP information meeting, 28th April 2022. https://indico.cern.ch/event/1154158/

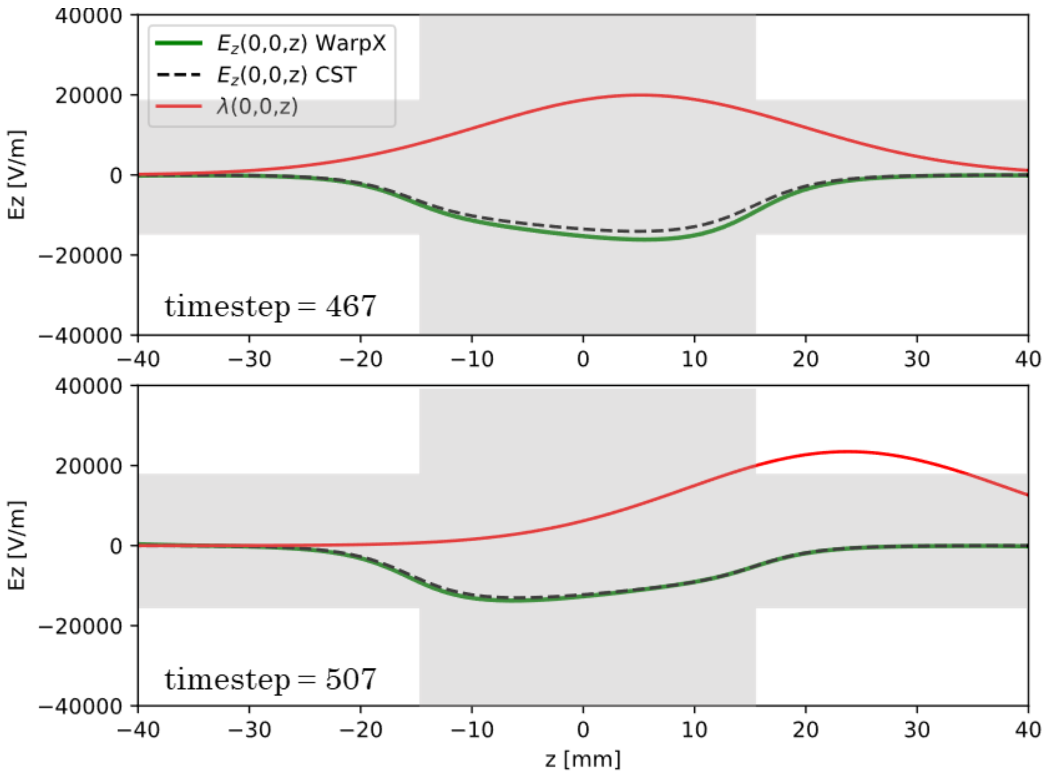
PhD motivation (II)





(v0.2) Explored open-source EM solvers to couple with `wakis`:

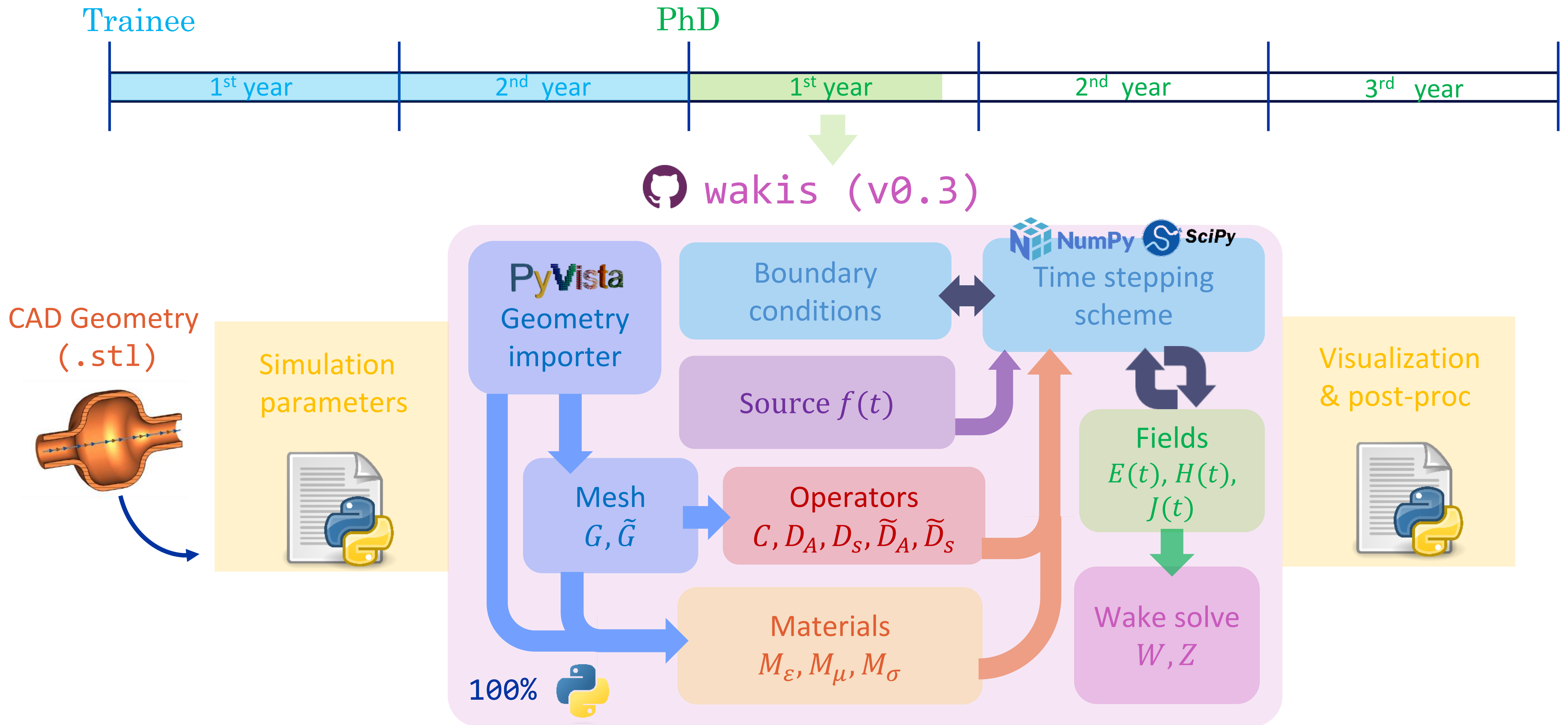
- **WarpX** was the most promising option: PIC+FDTD, mesh refinement (AMR), PML boundaries, python API
- Found very good agreement for pillbox cavity, AMReX developed importer for more complex geometries
- WarpX at the time, did not include material tensors other than vacuum/PEC. Found limitations in beam injection & PML behaviour too, C++ core, complexity…



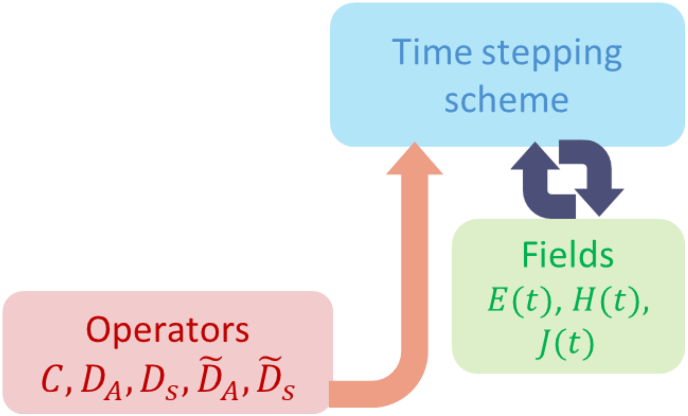
[2] *Progress on in-house wake solver "wakis"*. ABP-CEI meeting, 3rd August 202
https://indico.cern.ch/event/1283485/

PhD status

(v.0.3) Decided to explore CST®'s numerical method, the **Finite Integration Technique (FIT)** in a python mockup ☺



Outline



Time stepping scheme
Operators
$C, D_A, D_s, \tilde{D}_A, \tilde{D}_s$
Fields
$E(t), H(t),$
$J(t)$

Finite Integration Technique

Maxwell Equations (Integral form)

$$\oint_{\partial A} \boldsymbol{E} \cdot \boldsymbol{ds} = -\iint_A \frac{\partial \boldsymbol{B}}{\partial t} \cdot \boldsymbol{dA}$$

$$\oint_{\partial A} \boldsymbol{H} \cdot \boldsymbol{ds} = -\iint_A \left(\frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J}\right) \cdot \boldsymbol{dA}$$

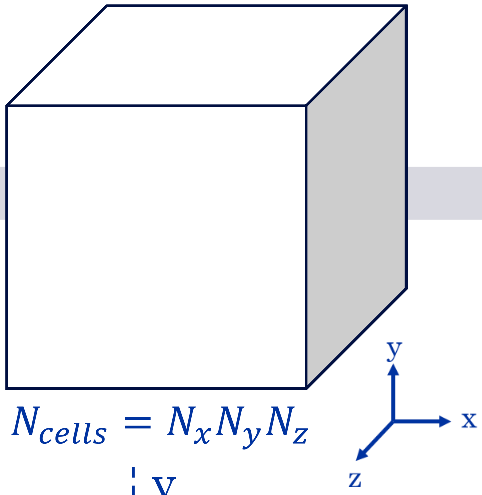
$$\oiint_{\partial V} \boldsymbol{B} \cdot \boldsymbol{dA} = \boldsymbol{0}$$

$$\oiint_{\partial V} \boldsymbol{D} \cdot \boldsymbol{dA} = \iiint_V \boldsymbol{\rho}\, \boldsymbol{dV}$$

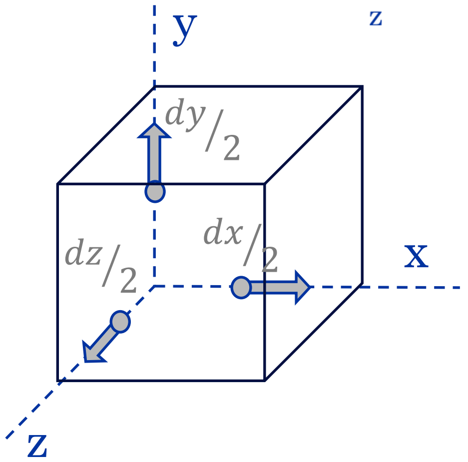
$$\boldsymbol{D} = \underline{\underline{\varepsilon}}\boldsymbol{E}, \quad \boldsymbol{B} = \underline{\underline{\mu}}\boldsymbol{H}, \quad \mathbf{J} = \underline{\underline{\sigma}}\boldsymbol{E} + \boldsymbol{\rho v}$$

1st approximation Domain discretization

dx, dy, dz



$N_{cells} = N_x N_y N_z$



Maxwell Grid Equations*

$$\boldsymbol{C}\boldsymbol{D_s}\boldsymbol{e} = -\boldsymbol{D_A}\frac{\partial \boldsymbol{b}}{\partial t}$$

$$\widetilde{\boldsymbol{C}}\widetilde{\boldsymbol{D}}_{\boldsymbol{s}}\boldsymbol{h} = \widetilde{\boldsymbol{D}}_{\boldsymbol{A}}\left(\frac{\partial \boldsymbol{d}}{\partial t} + \boldsymbol{j}\right)$$

$$\boldsymbol{S}\boldsymbol{D_A}\boldsymbol{b} = \boldsymbol{0}$$

$$\widetilde{\boldsymbol{S}}\widetilde{\boldsymbol{D}}_{\boldsymbol{A}}\left(\frac{\partial \boldsymbol{d}}{\partial t} + \boldsymbol{j}\right) = \boldsymbol{0}$$

$$\boldsymbol{d} = \widetilde{\boldsymbol{D}}_{\boldsymbol{\varepsilon}}\boldsymbol{e}, \quad \mathbf{b} = \boldsymbol{D_{\mu}}\boldsymbol{h}, \quad \mathbf{j} = \widetilde{\boldsymbol{D}}_{\boldsymbol{\sigma}}\boldsymbol{e} + \boldsymbol{j_{src}}$$

- Operators
- Grid areas and lengths
- Materials

*Formulation by T. Weiland: Wakefields and Impedances , 1991

Finite Integration Technique (II)

Maxwell Grid Equations

$$\boldsymbol{C}\boldsymbol{D}_s\boldsymbol{e} = -\boldsymbol{D}_A\frac{\partial \boldsymbol{b}}{\partial t}$$

$$\tilde{\boldsymbol{C}}\tilde{\boldsymbol{D}}_s\boldsymbol{h} = \tilde{\boldsymbol{D}}_A\left(\frac{\partial \boldsymbol{d}}{\partial t} + \boldsymbol{j}\right)$$

$$\boldsymbol{d} = \tilde{\boldsymbol{D}}_\varepsilon\boldsymbol{e}, \quad \mathbf{b} = \boldsymbol{D}_\mu\boldsymbol{h},$$

$$\mathbf{j} = \tilde{\boldsymbol{D}}_\sigma\boldsymbol{e} + \boldsymbol{j}_{src}$$

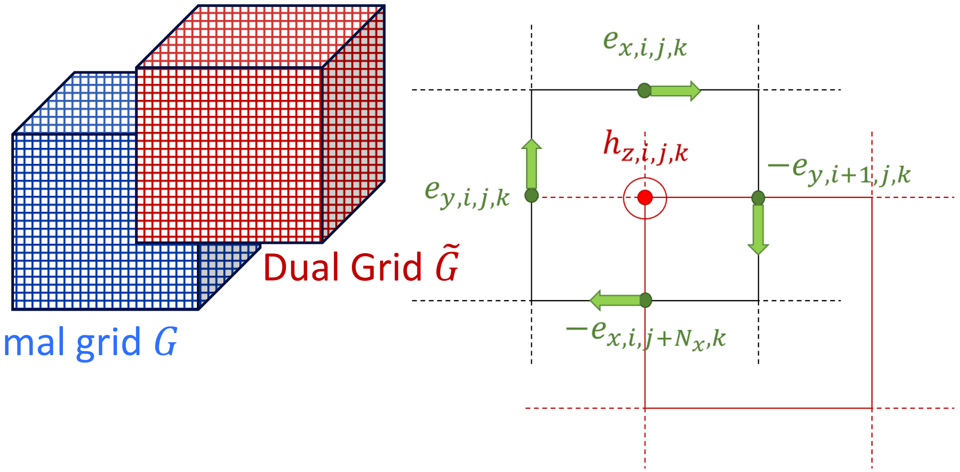
$$\boldsymbol{S}\boldsymbol{D}_A\boldsymbol{b} = 0$$

$$\tilde{\boldsymbol{S}}\tilde{\boldsymbol{D}}_A\left(\frac{\partial \boldsymbol{d}}{\partial t} + \boldsymbol{j}\right) = 0$$

2nd approximation
Evolution in timestep Δt

$$\frac{\partial \mathrm{a}}{\partial t} = \frac{\mathrm{a}^{n+1} - \mathrm{a}^{n}}{\Delta \mathrm{t}}$$

Domain discretization
With Yee grid



Time-stepping scheme (leapfrog)

$$\boldsymbol{h}^{n+1} = \boldsymbol{h}^{n} - \Delta t\,\tilde{\boldsymbol{D}}_s\boldsymbol{D}_\mu^{-1}\boldsymbol{D}_A^{-1}\boldsymbol{C}\boldsymbol{e}^{n+0.5}$$

$$\boldsymbol{e}^{n+1.5} = \boldsymbol{e}^{n+0.5} + \Delta t\,\boldsymbol{D}_s\tilde{\boldsymbol{D}}_\varepsilon^{-1}\tilde{\boldsymbol{D}}_A^{-1}\tilde{\boldsymbol{C}}\boldsymbol{h}^{n} - \tilde{\boldsymbol{D}}_\varepsilon^{-1}\boldsymbol{j}_{src}^{n}$$

- ✓ 2nd order convergence in time
- ✓ Can pre-compute most of the quantities → speed
- ✓ No matrix inversion needed during time loop

Gauss Laws (for E and H) considered satisfied* $\forall\ t$ for no charges j_{ext}

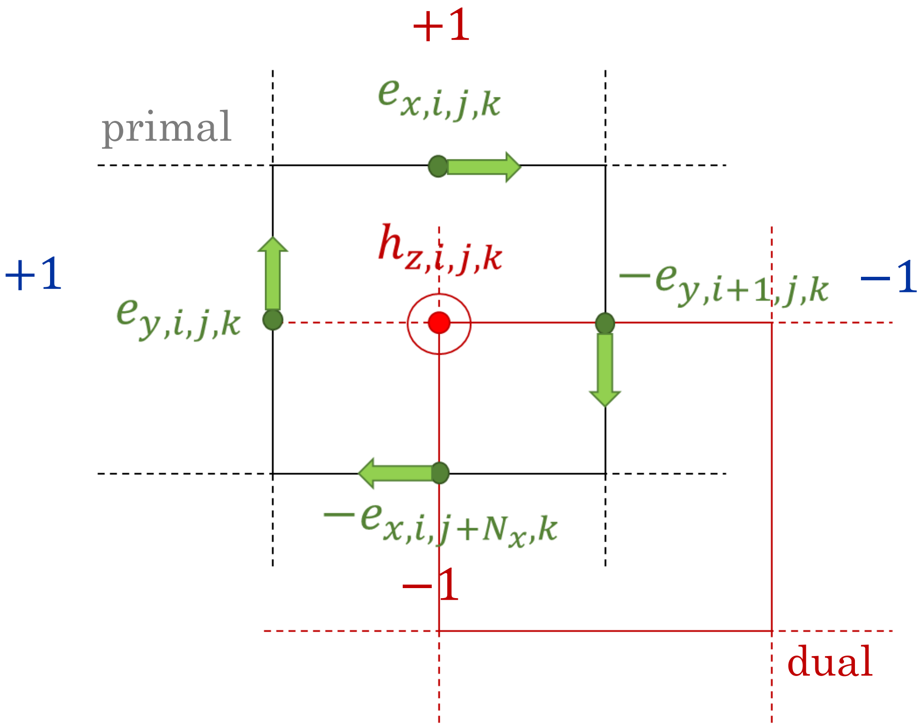
*Raymond Rumpf, *Electromagnetic and Photonic Simulation for the Beginner*, Artech, 2022.

FIT (III): Operators C, $\tilde{C}$

Time-stepping scheme in vacuum

$$\boldsymbol{h}^{n+1} = \boldsymbol{h}^{n} - \Delta t \boldsymbol{\mu}_0^{-1} \widetilde{\boldsymbol{D}}_s \boldsymbol{D}_A^{-1} \boldsymbol{C} \boldsymbol{e}^{n+0.5}$$

$$\boldsymbol{e}^{n+1.5} = \boldsymbol{e}^{n+0.5} + \Delta t \boldsymbol{\varepsilon}_0^{-1} \boldsymbol{D}_s \widetilde{\boldsymbol{D}}_A^{-1} \widetilde{\boldsymbol{C}} \boldsymbol{h}^{n} - \boldsymbol{\varepsilon}_0^{-1} \boldsymbol{j}^{n}$$



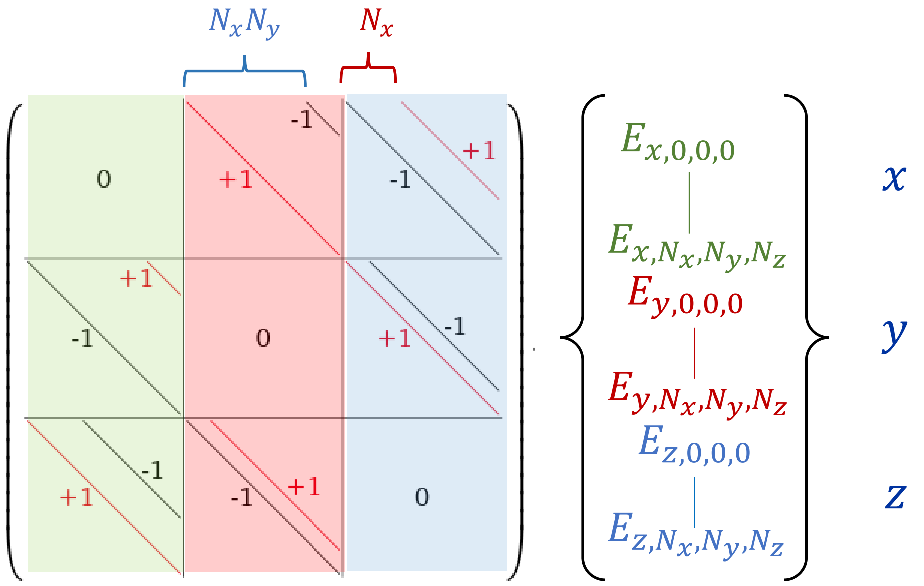
The **curl operator** C on primal grid and $\tilde{C}$ on the dual grid, with $\tilde{C} = C^t$ is a $3N_{cells} \times 3N_{cells}$ sparse matrix made of bands of $+1$ and -1

By modifying the columns or rows of C, one can add **boundary conditions**:

- Perfect Electric Conductor (PEC): columns to zero at ijk of boundary cells
- Perfect Magnetic Conductor (PMC): rows to zero at ijk of boundary cells

Sparcity of C

$$\mathbf{C} = \begin{pmatrix} 0 & -\mathbf{P}_z & \mathbf{P}_y \\ \mathbf{P}_z & 0 & -\mathbf{P}_x \\ -\mathbf{P}_y & \mathbf{P}_x & 0 \end{pmatrix} =$$



Size: $3N_{cells} \times 3N_{cells}$

Lexicographic indexation

FIT (IV): Diagonal matrices $D_A, D_s, \tilde{D}_A, \tilde{D}_s$

Time-stepping scheme vacuum

$$\boldsymbol{h}^{n+1} = \boldsymbol{h}^{n} - \Delta t \boldsymbol{\mu}_0^{-1} \tilde{\boldsymbol{D}}_s \boldsymbol{D}_A^{-1} \boldsymbol{C} \boldsymbol{e}^{n+0.5}$$

$$\boldsymbol{e}^{n+1.5} = \boldsymbol{e}^{n+0.5} + \Delta t \boldsymbol{\varepsilon}_0^{-1} \boldsymbol{D}_s \tilde{\boldsymbol{D}}_A^{-1} \tilde{\boldsymbol{C}} \boldsymbol{h}^{n} - \boldsymbol{\varepsilon}_0^{-1} \boldsymbol{j}^{n}$$

The diagonal matrices contain the **grid** information:

- $D_A^{-1} = diag\left\{A_{x,0.0,0}^{-1}, \dots, A_{x,N_x.N_y,N_z}^{-1}, A_{y,0.0,0}^{-1}, \dots, A_{y,N_x.N_y,N_z}^{-1}, A_{z,0.0,0}^{-1}, \dots, A_{z,N_x.N_y,N_z}^{-1}\right\}$
- $D_s = diag\{l_{x,0.0,0}, \dots, l_{x,N_x.N_y,N_z}, l_{y,0.0,0}, \dots, l_{y,N_x.N_y,N_z}, l_{z,0.0,0}, \dots, l_{z,N_x.N_y,N_z}\}$

Where (i.e. x-direction):

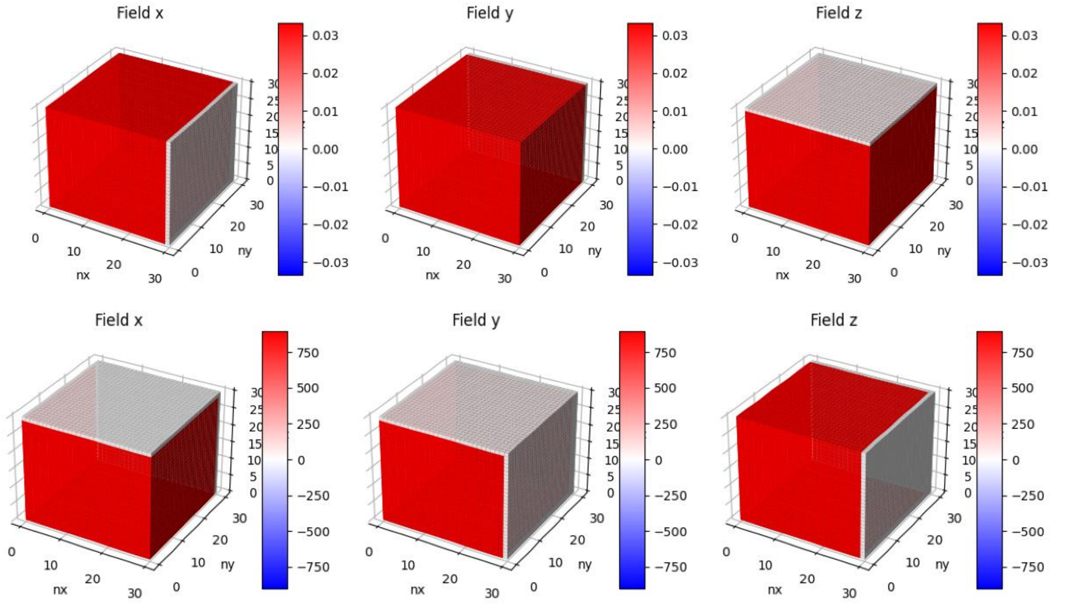
- $l_{x,i.j,k} = x_{i+1.j,k} - x_{i,j,k}$
- $A_{x,i.j,k} = l_{y\ i.j,k} \times l_{z\ i.j,k}$

- $\tilde{\boldsymbol{D}}_A$ **and** $\tilde{\boldsymbol{D}}_s$ are analogous for the **dual grid**. Due to the Yee staggering, some of the components are zero:

- If not set to zero, but equal to D_A and D_s at the opposite end → **Periodic boundary conditions**

$\tilde{D}_s$

$\tilde{D}_A$



*generated with wakis built-in 3D plotting

FIT (V): Fields E, H, J

Time-stepping scheme vacuum

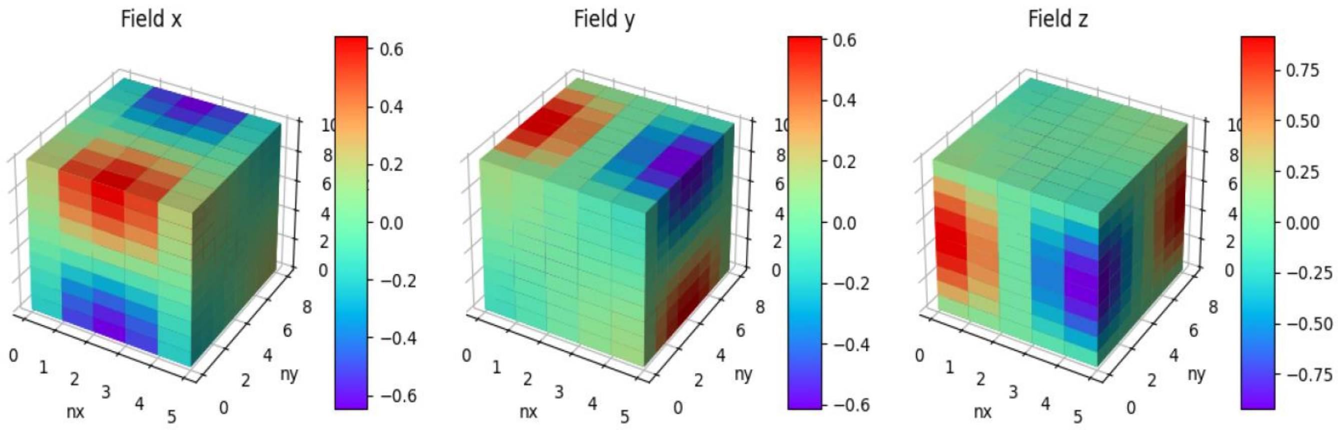
$$h^{n+1} = h^n - \Delta t \mu_0^{-1} \tilde{D}_s D_A^{-1} C e^{n+0.5}$$

$$e^{n+1.5} = e^{n+0.5} + \Delta t \varepsilon_0^{-1} D_s \tilde{D}_A^{-1} \tilde{C} h^n - \varepsilon_0^{-1} j^n$$

The fields are stored in memory as 3 (x, y, z) **3d** numpy matrices `np.zeros`(N_x, N_y, N_z). In the time-stepping, they must be converted to **1d arrays** with lexicographic indexing:

$$n = 1 + (i-1) + (j-1)N_x + (k-1)N_x N_y$$

Class `Field` has custom **magic methods** (`__getitem__`, `__setitem__`, `__mul__`, …) to handle 3d to collapsed conversion, operations and built-in **visualization methods** (`inspect()`, `inspect3d()`)



**E.g., E field in a coarse-meshed cube resonator generated with `wakis` built-in 3D plotting `solver.E.inspect3d()`*

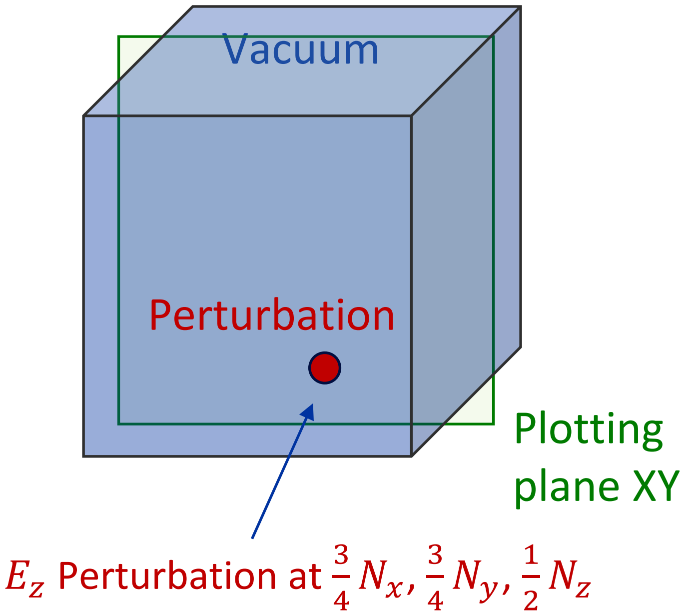
`field.py`

`class Field`

$$\begin{Bmatrix} E_{x,0,0,0} \\ \vdots \\ E_{x,N_x,N_y,N_z} \\ E_{y,0,0,0} \\ \vdots \\ E_{y,N_x,N_y,N_z} \\ E_{z,0,0,0} \\ \vdots \\ E_{z,N_x,N_y,N_z} \end{Bmatrix} \begin{matrix} x \\ \\ y \\ \\ z \end{matrix}$$

Lexicographic indexation

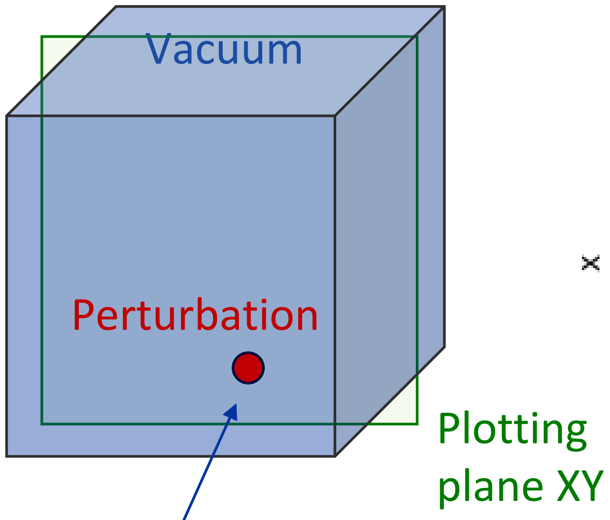
Example: Perturbation in free-space with BC



examples/script_noeb_fit.py

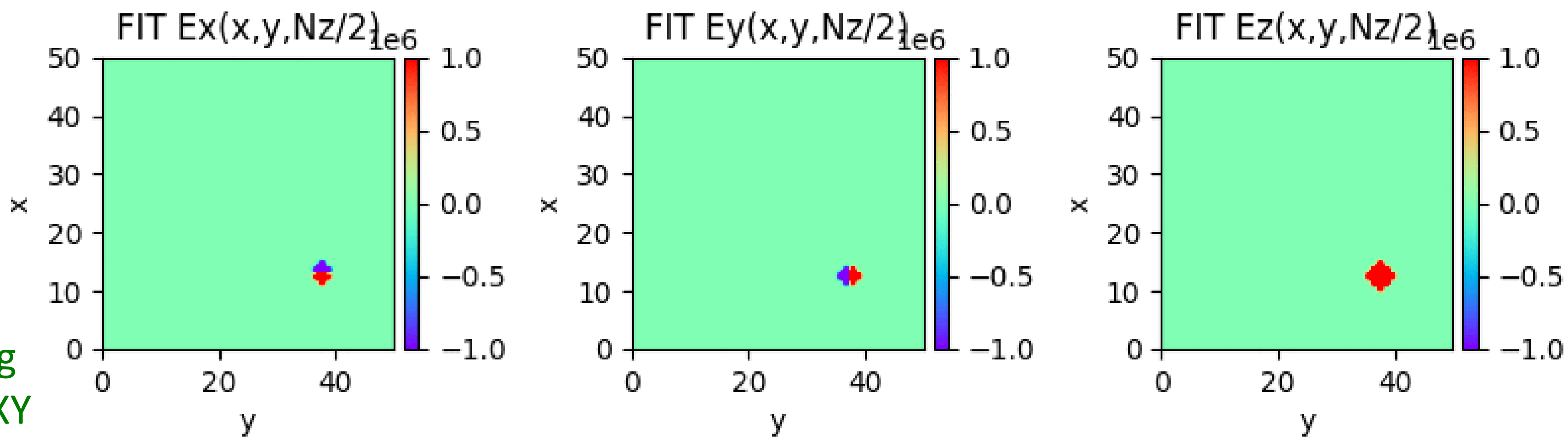
A static field perturbation in E_z generates a spheric wavefront that is reflected for PEC BCs or re-enters the domain for Periodic BCs

Example: Perturbation in free-space with BC



E_z Perturbation at $\frac{3}{4}N_x, \frac{3}{4}N_y, \frac{1}{2}N_z$

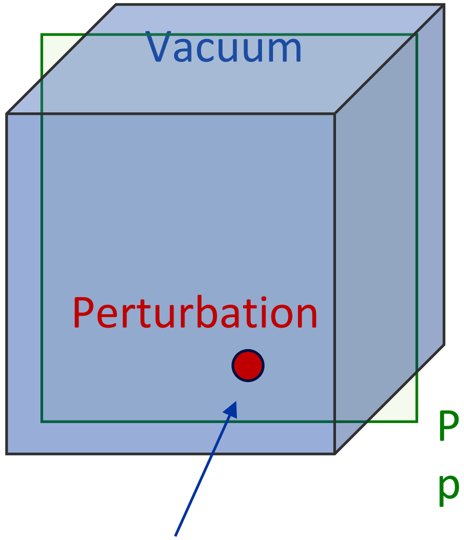
Boundary conditions (BCs): All Periodic



examples/script_noeb_fit.py

A static field perturbation in E_z generates a spheric wavefront that is reflected for PEC BCs or re-enters the domain for Periodic BCs

Example: Perturbation in free-space with BC



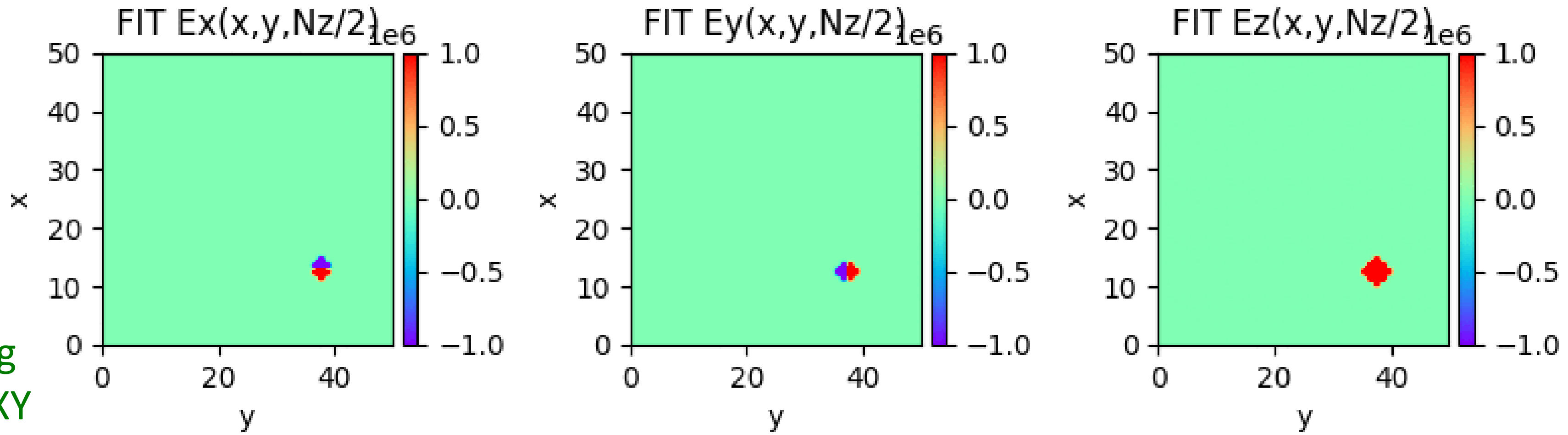
Plotting plane XY

E_z Perturbation at $\frac{3}{4}N_x, \frac{3}{4}N_y, \frac{1}{2}N_z$

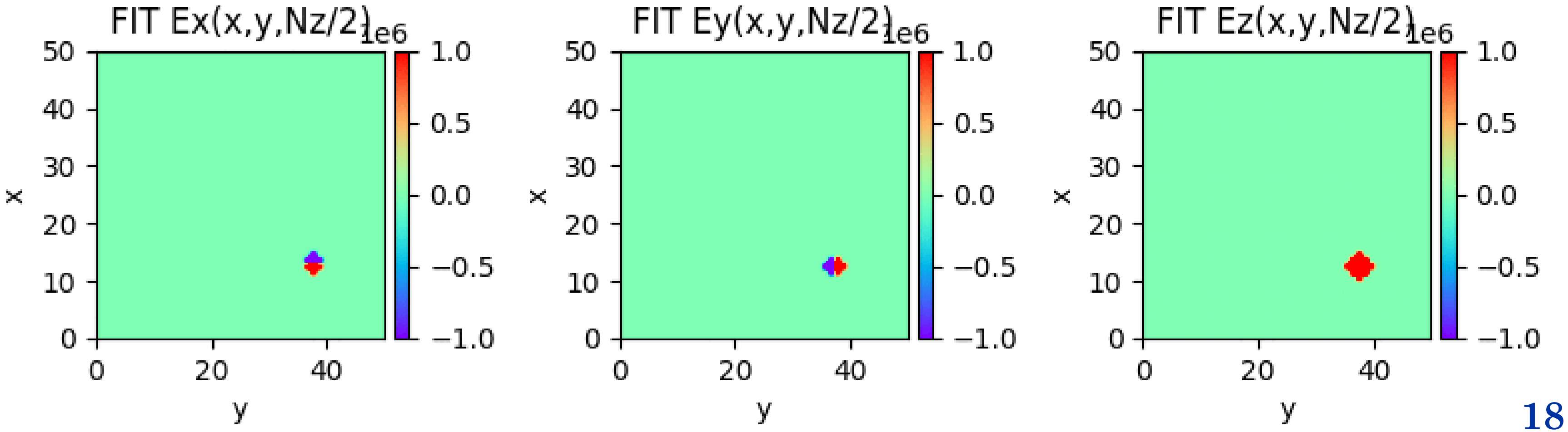
examples/script_noeb_fit.py

A static field perturbation in E_z generates a spheric wavefront that is reflected for PEC BCs or re-enters the domain for Periodic BCs

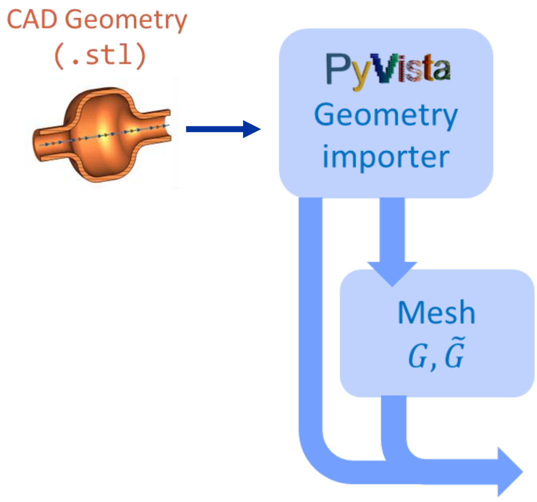
Boundary conditions (BCs): All Periodic



Boundary conditions (BCs): all PEC



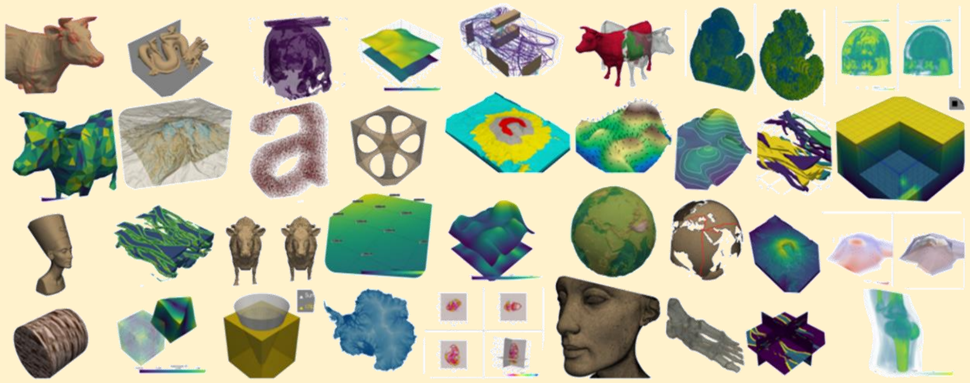
Outline



Importing geometry with PyVista

PyVista

3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK)

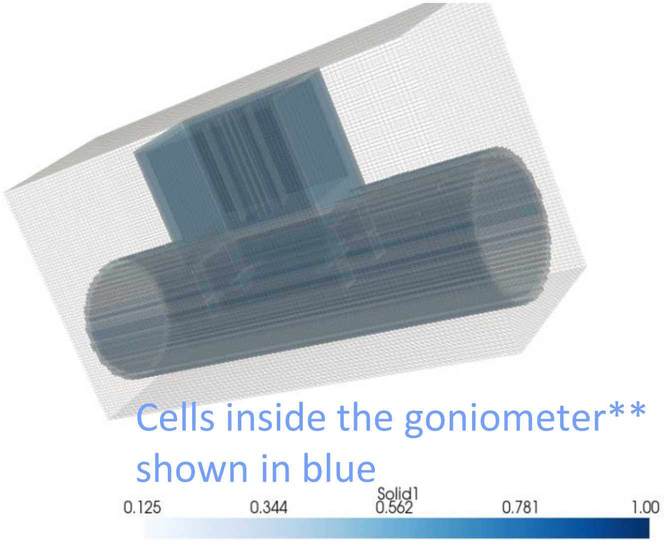


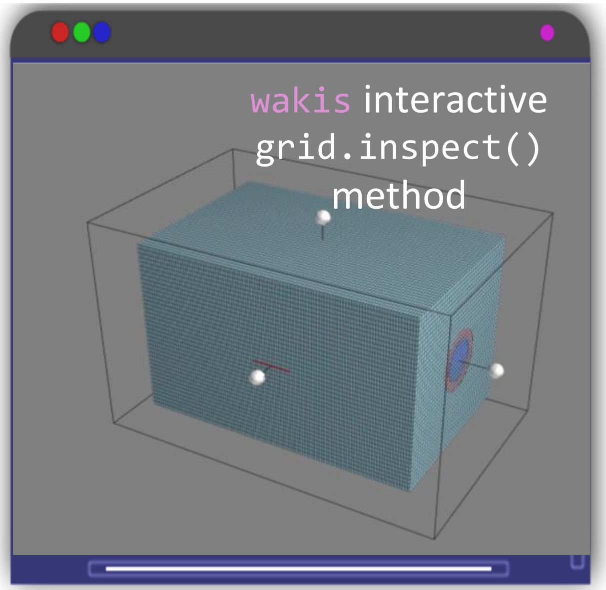
https://github.com/pyvista/pyvista

https://docs.pyvista.org/version/stable/

Using **PyVista** capabilities, `wakis` can:

- Import CAD geometry: `stl= pv.read(stl_file)`
- Generate a grid: `grid = pv.StructuredGrid(X, Y, Z)`
- Find the cells that are inside the geometry with advanced collision filters* → mask for material properties
- State-of-the-art interactive 3d plotting

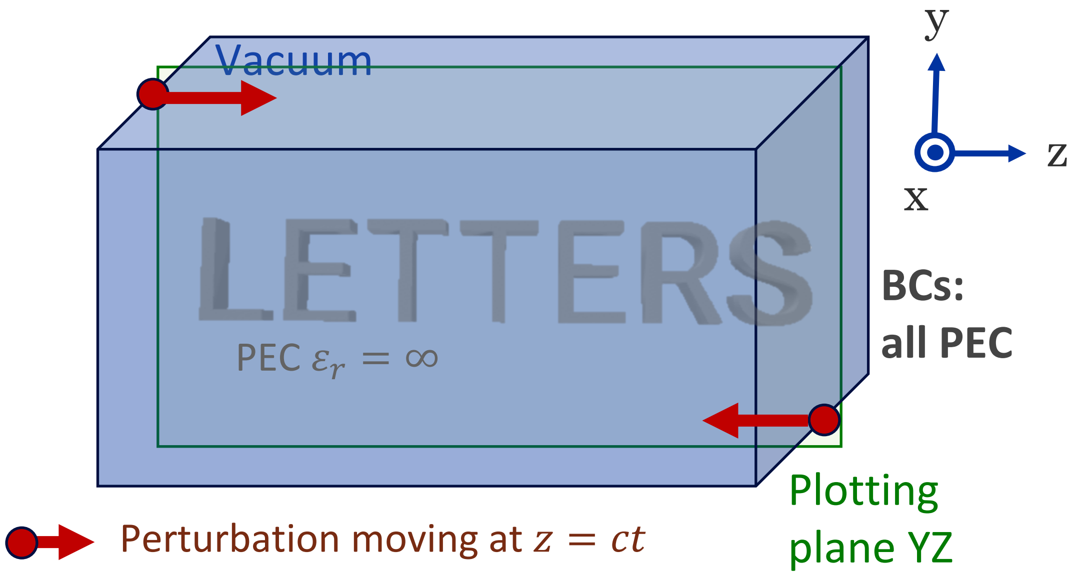




*Pyvista extract_cells_inside_surface filter

** Simplified goniometer geometry by C. Antuono

Example: Perturbation with imported STL

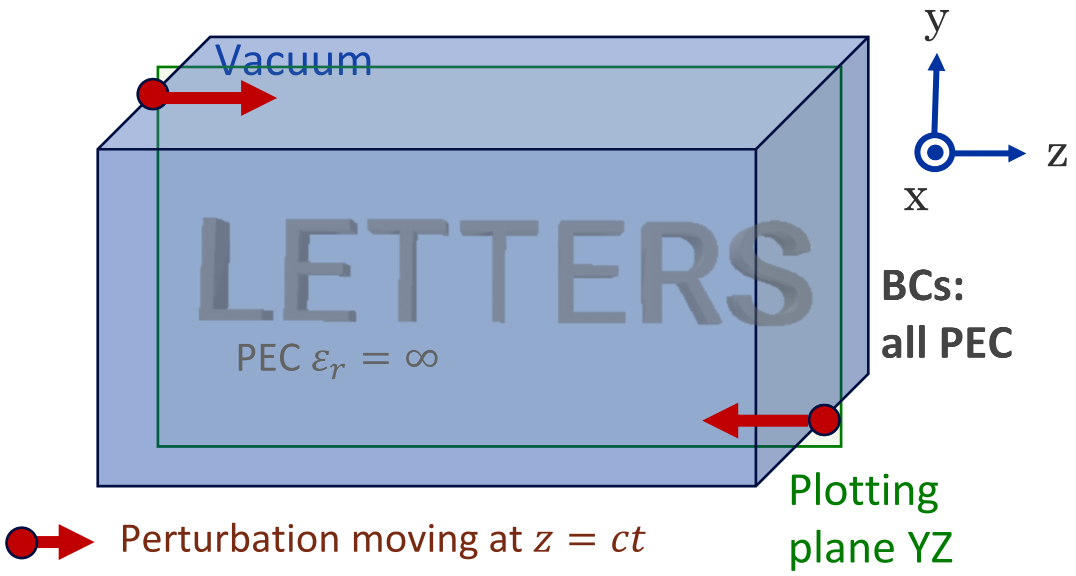


examples/script_letters_fit.py

A field perturbation in E_z moving at $z = ct$ excites the field in the vacuum domain revealing the PEC imported geometry

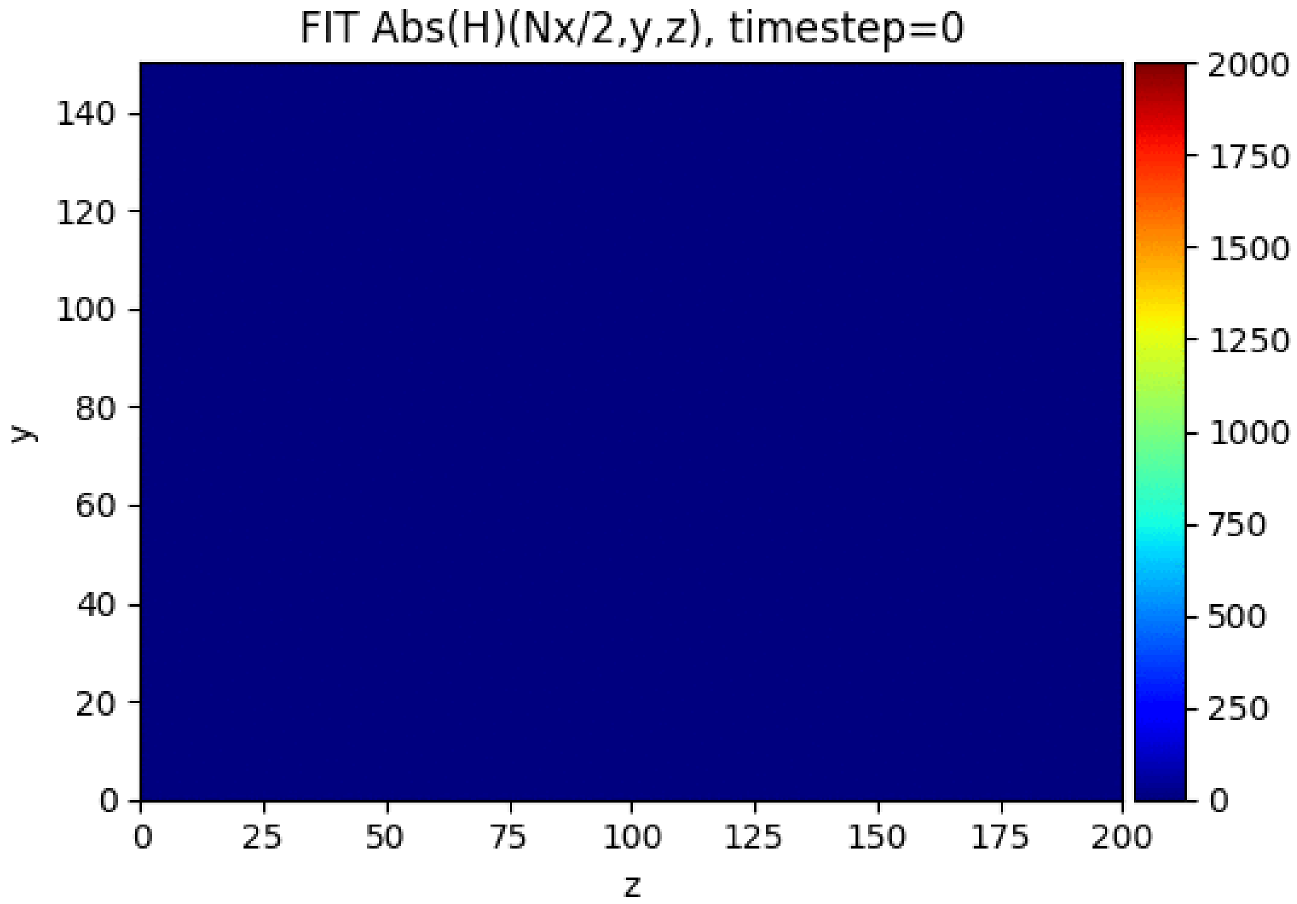
*STL files generated with online tool: https://text2stl.mestres.fr/

Example: Perturbation with imported STL



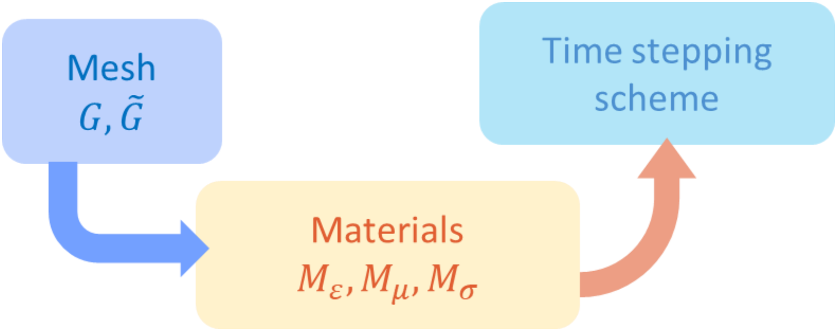
examples/script_letters_fit.py

A field perturbation in E_z moving at $z = ct$ excites the field in the vacuum domain revealing the PEC imported geometry



*STL files generated with online tool: https://text2stl.mestres.fr/

Outline


Mesh
$G, \tilde{G}$
Materials
$M_\varepsilon, M_\mu, M_\sigma$
Time stepping scheme

Material tensors for ε, μ

Time-stepping scheme with ε, μ

$$\boldsymbol{h}^{n+1} = \boldsymbol{h}^{n} - \Delta t\, \widetilde{\boldsymbol{D}}_s \boldsymbol{D}_\mu^{-1} \boldsymbol{D}_A^{-1} \boldsymbol{C} \boldsymbol{e}^{n+0.5}$$

$$\boldsymbol{e}^{n+1.5} = \boldsymbol{e}^{n+0.5} + \Delta t \boldsymbol{D}_s \widetilde{\boldsymbol{D}}_\varepsilon^{-1} \widetilde{\boldsymbol{D}}_A^{-1} \widetilde{\boldsymbol{C}} \boldsymbol{h}^{n} - \widetilde{\boldsymbol{D}}_\varepsilon^{-1} \boldsymbol{j}^{n}$$

The material matrices are considered **diagonal* tensors**:

- User can specify ***relative*** permitivity/permeability: $\varepsilon_r = {}^{\varepsilon}/_{\varepsilon_0}$
- The matrix will be built with a background material $\varepsilon_{r,bg}, \mu_{r,bg}$
- The material associated with each imported stl geometry will overwrite the background material in the tensor (numpy + pyvista)
- User can also specify anisotropic materials by defining $\varepsilon_{r,x}, \varepsilon_{r,y}, \varepsilon_{r,z}, \ldots$

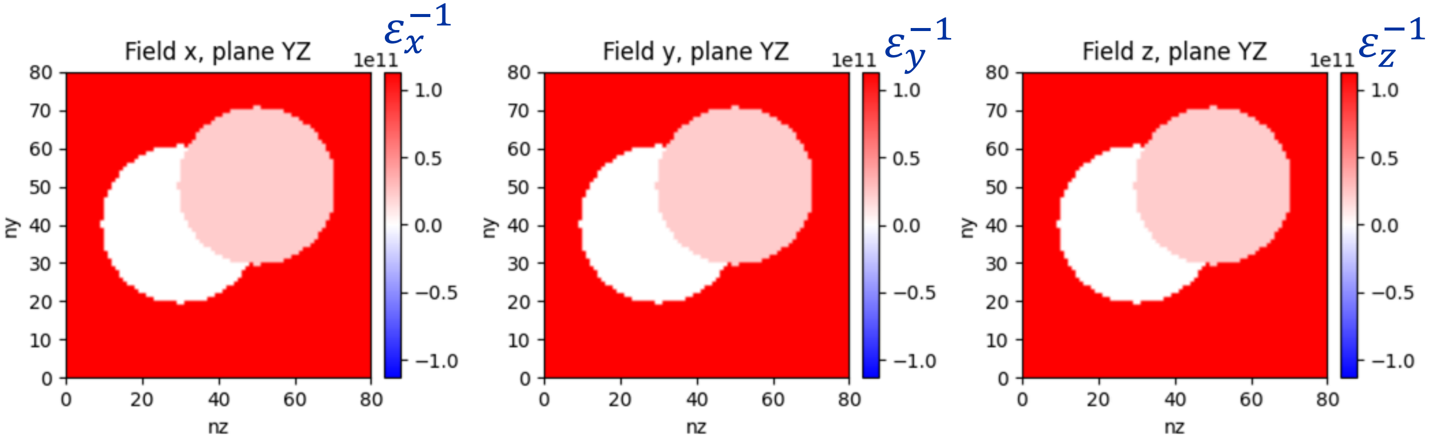
$$D_\varepsilon^{-1} = diag\left\{\varepsilon_{x,0.0,0}^{-1}, \ldots, \varepsilon_{x,N_x.N_y,N_z}^{-1}, \varepsilon_{y,0.0,0}^{-1}, \ldots, \varepsilon_{y,N_x.N_y,N_z}^{-1}, \varepsilon_{z,0.0,0}^{-1}, \ldots, \varepsilon_{z,N_x.N_y,N_z}^{-1}\right\} \quad 3 \times N_{cells}$$

$$D_\mu^{-1} = diag\left\{\mu_{x,0.0,0}^{-1}, \ldots, \mu_{x,N_x.N_y,N_z}^{-1}, \mu_{y,0.0,0}^{-1}, \ldots, \mu_{y,N_x.N_y,N_z}^{-1}, \mu_{z,0.0,0}^{-1}, \ldots, \mu_{z,N_x.N_y,N_z}^{-1}\right\} \quad 3 \times N_{cells}$$

examples/test/test_materials.py

A PEC sphere $\varepsilon^{-1} = 0$ and a Dielectric sphere $\varepsilon^{-1} = (5\varepsilon_0)^{-1}$ imported in a vacuum background $\varepsilon^{-1} = \varepsilon_0^{-1}$.

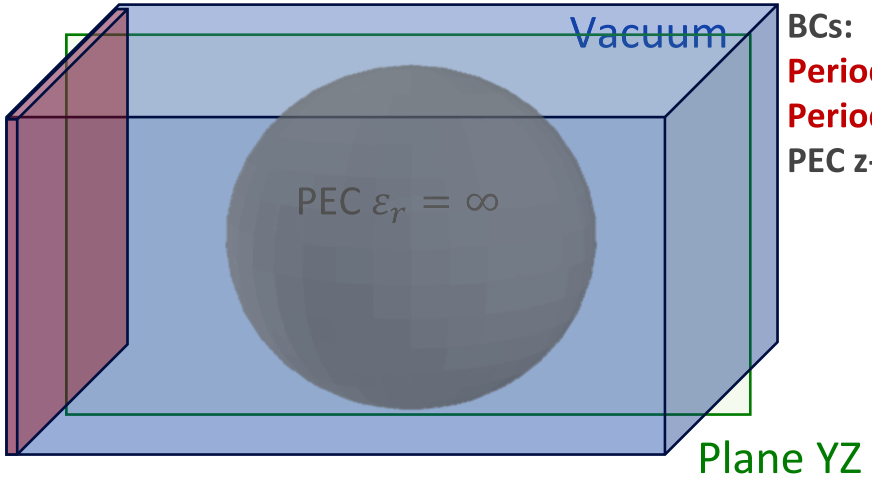
User can inspect the material tensors by:
`solver.ieps.inspect(plane='YZ')`



*This approximation is only not valid for Gyrotropic materials

examples/script_planewave_fit.py

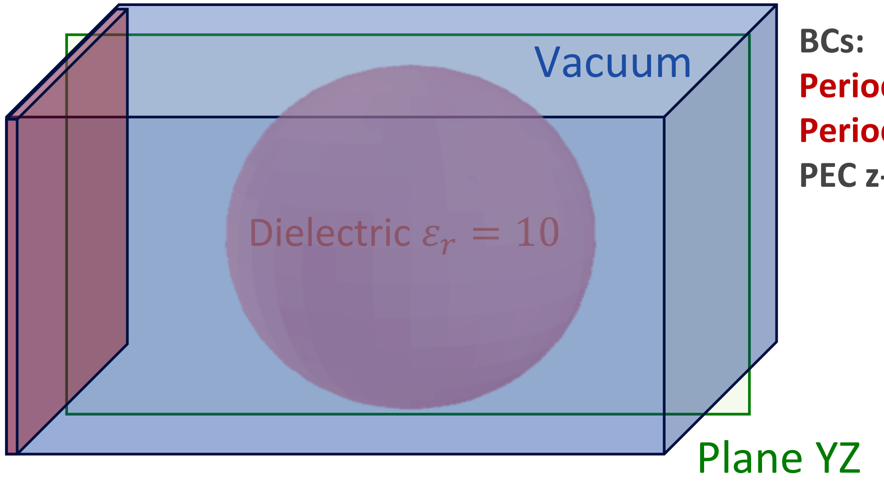
Example: Planewave interacting with sphere



Plane wave port

BCs:
Periodic x-/x+
Periodic y-/y+
PEC z-/z+

Special thanks to Prof. M. Cotelo (UPM) for the help with the planewave port setup and validation

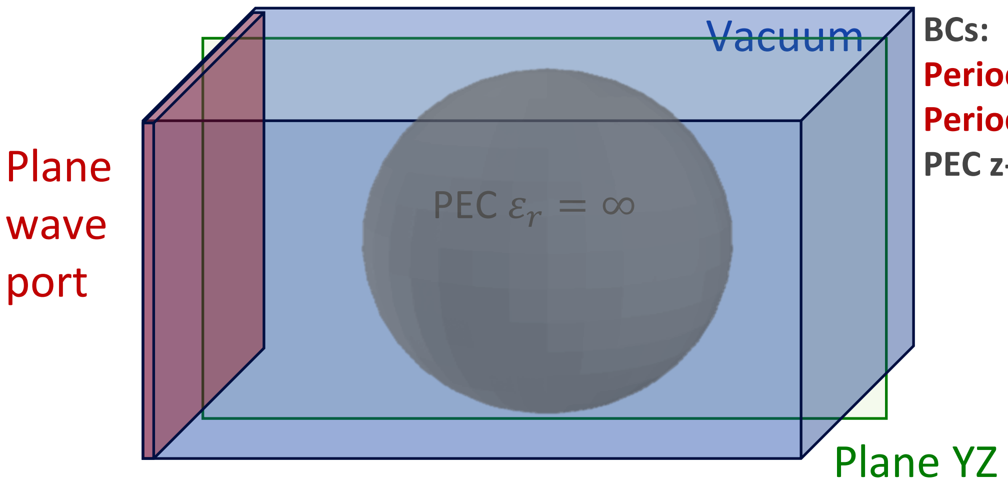


Plane wave port

BCs:
Periodic x-/x+
Periodic y-/y+
PEC z-/z+

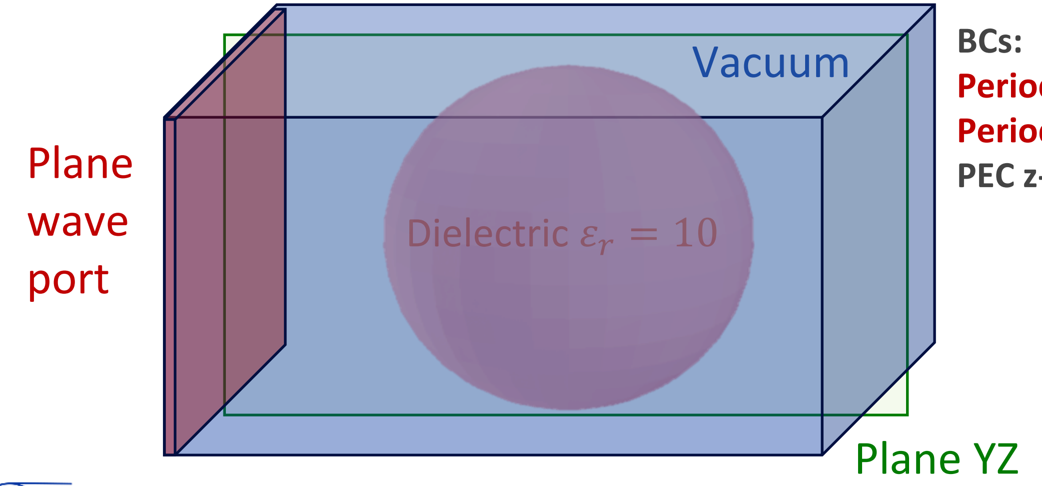
examples/script_planewave_fit.py

Example: Planewave interacting with sphere



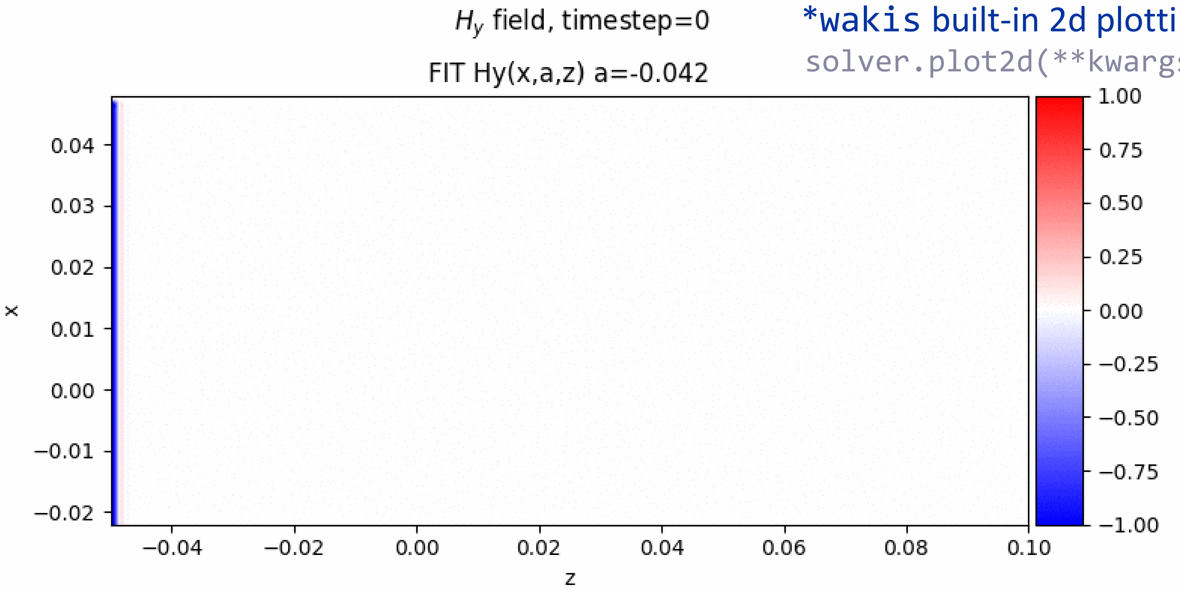
BCs:
Periodic x-/x+
Periodic y-/y+
PEC z-/z+

Special thanks to Prof. M. Cotelo (UPM) for the help with the planewave port setup and validation

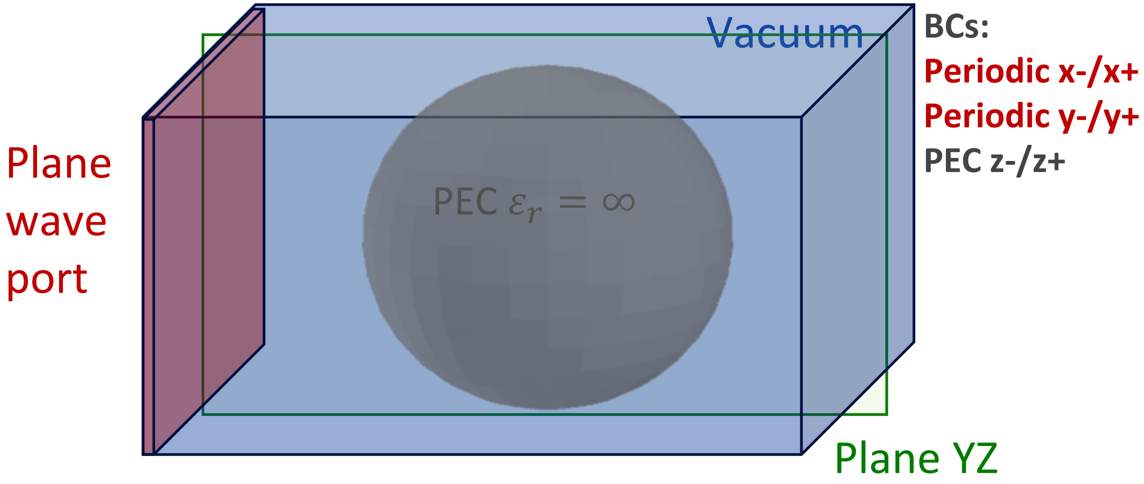


BCs:
Periodic x-/x+
Periodic y-/y+
PEC z-/z+

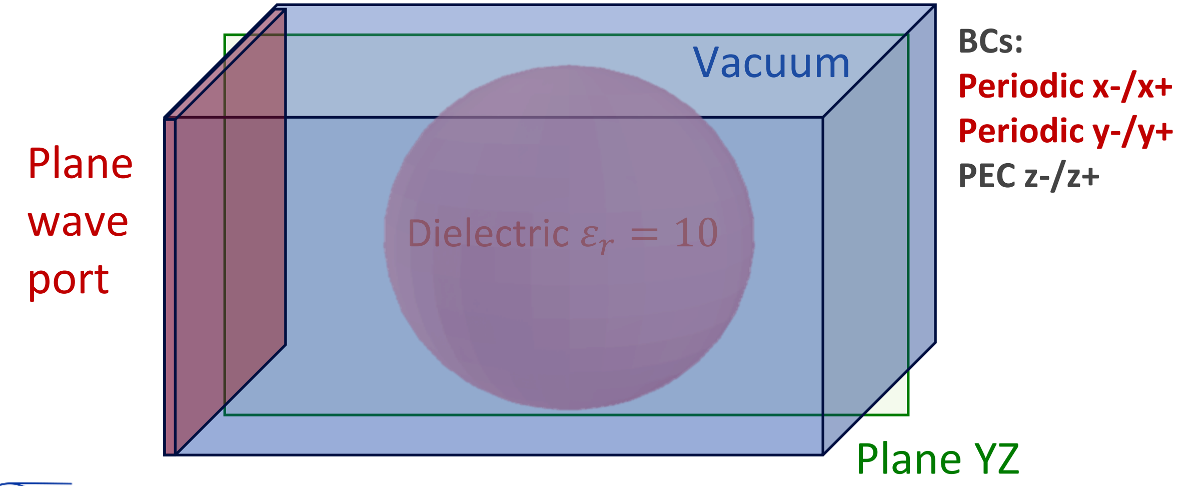
*wakis built-in 2d plotting
`solver.plot2d(**kwargs)`



Example: Planewave interacting with sphere

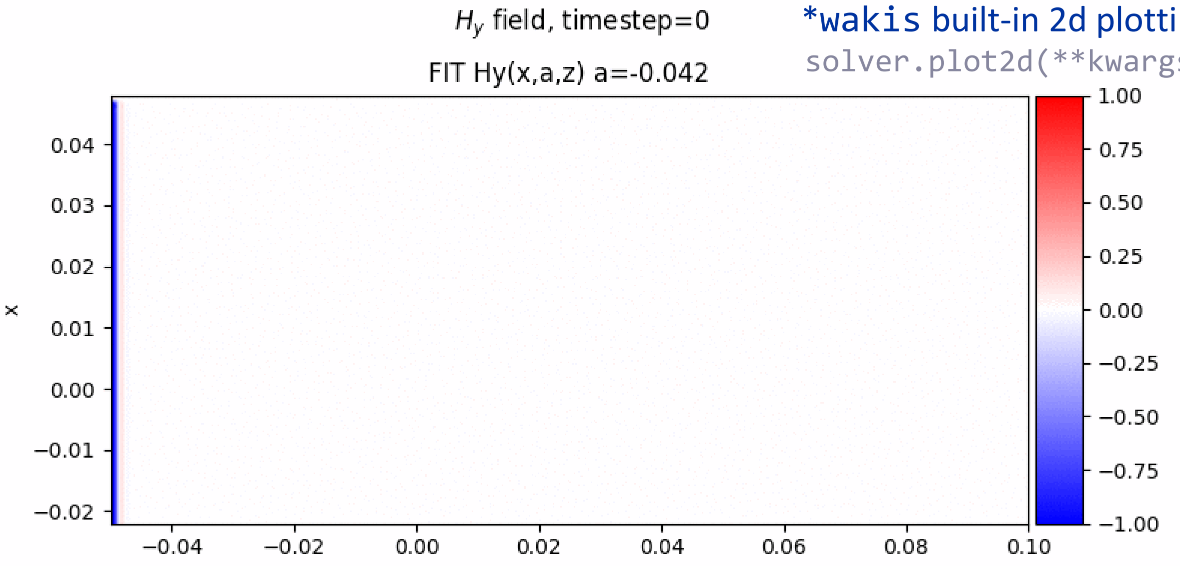


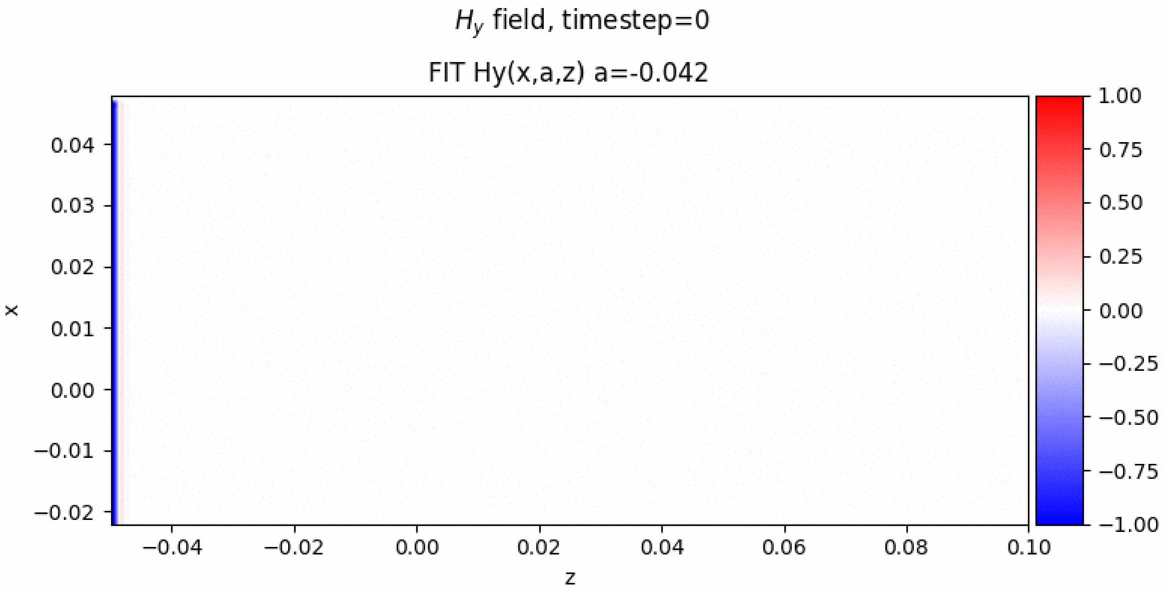
Special thanks to Prof. M. Cotelo (UPM) for the help with the planewave port setup and validation



*wakis built-in 2d plotting

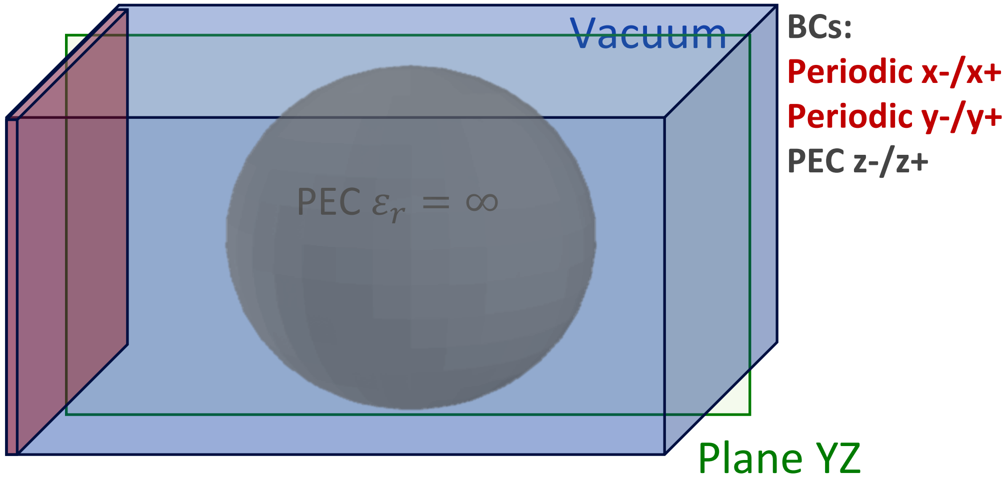
`solver.plot2d(**kwargs)`



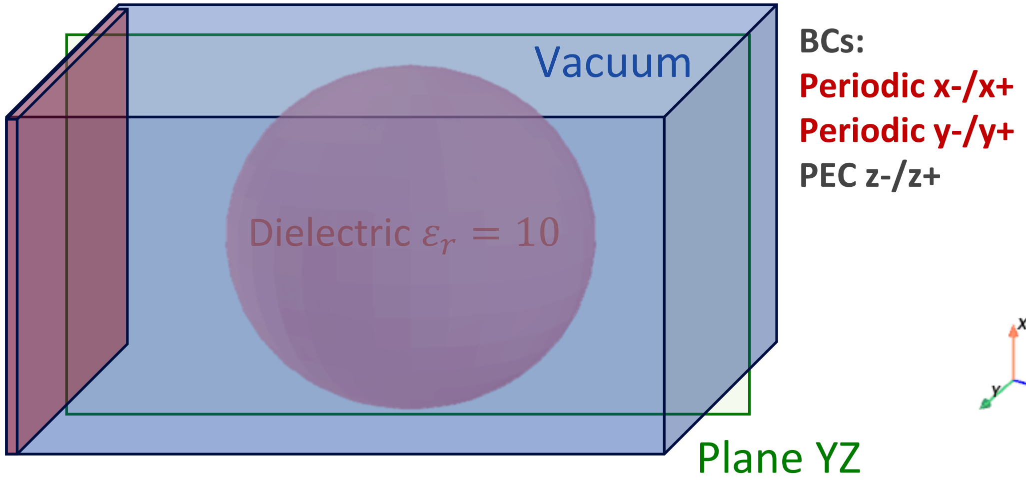


examples/script_planewave_fit.py

Example: Planewave interacting with sphere

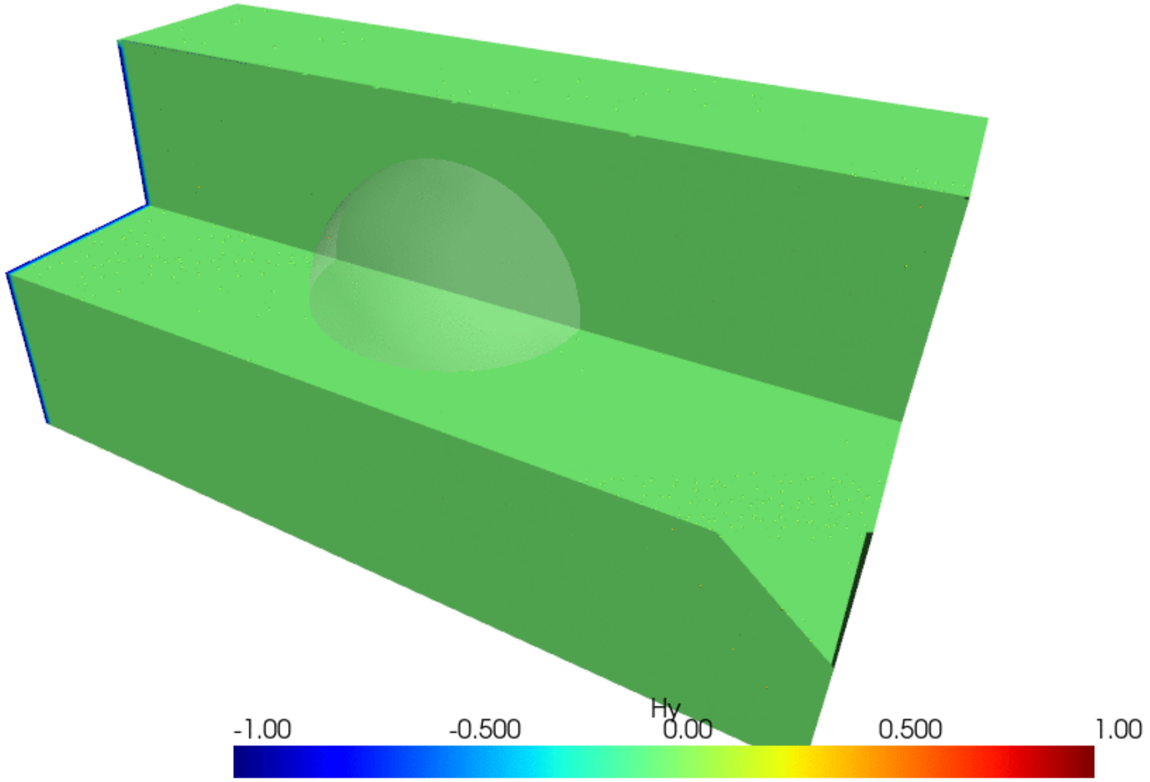


Special thanks to Prof. M. Cotelo (UPM) for the help with the planewave port setup and validation

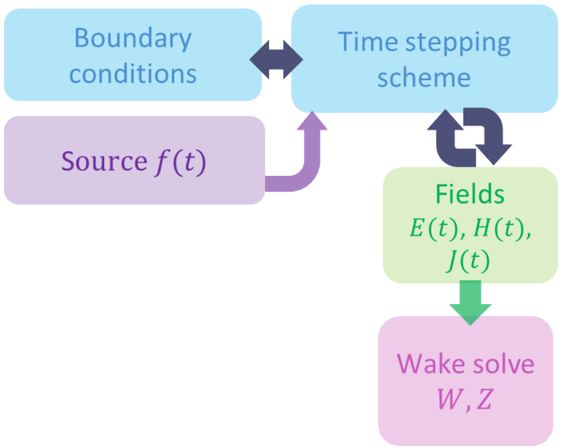


Hy field, timestep=0

Generated with wakis built-in 3d plotting `solver.plot3d(**kwargs)` based on pyVista



Outline



CERN IFN-GV

Beam injection as current J_z

- **Sources ($\boldsymbol{E}, \boldsymbol{H}, \boldsymbol{J}$)** are managed in a dedicated class inside `sources.py`

Each source should have a `source.update(t)` method that will be called every timestep dt in the simulation loop.

- For the case of a **particle beam**, the source is a current in J_z, applied at $x_s, y_s \ \forall \ z$ with a gaussian time profile.
  - q is the charge in [C], typically $\approx 10^{-9}$ C
  - σ_z is the beam size [m]
  - β is the ratio of the beam speed v to the speed of light c

$$J_z(x_s, y_s, z) = \frac{q\beta c}{\sqrt{2\pi\sigma_z}} e^{\frac{-(s-s_0)^2}{2\sigma_z^2}}$$

$$s = z - \beta ct \ ; \quad s_0 = z_{min} - \beta ct$$

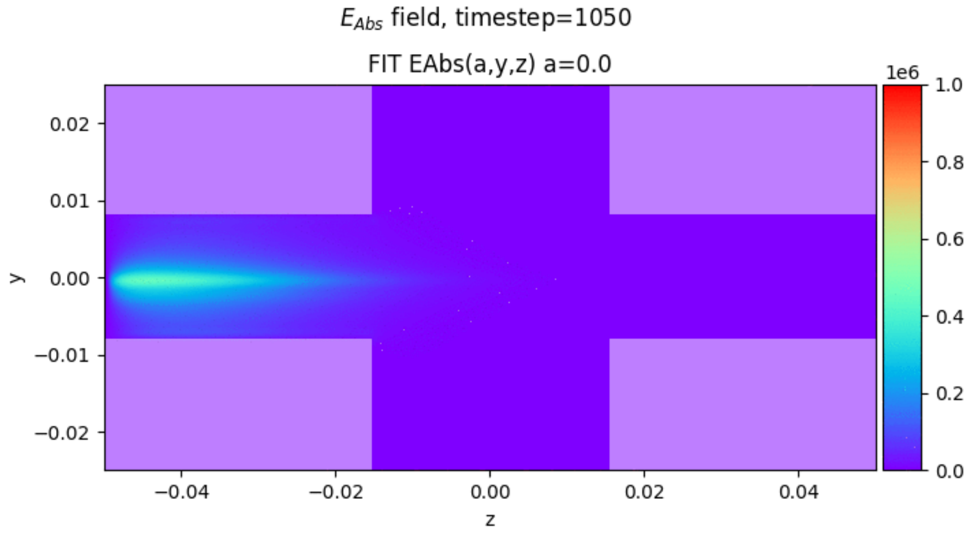
Beam injection as current J_z

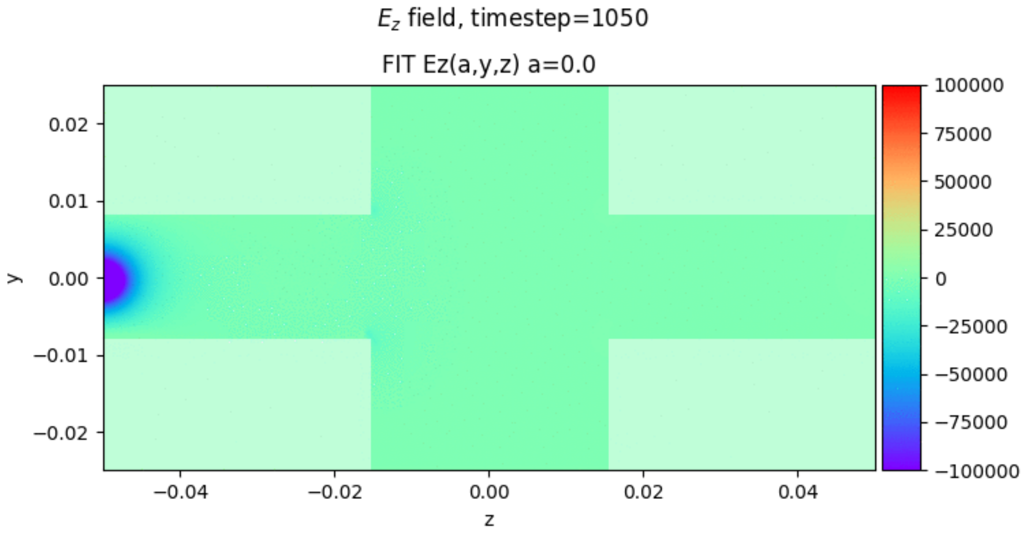
- **Sources ($\boldsymbol{E}, \boldsymbol{H}, \boldsymbol{J}$)** are managed in a dedicated class inside `sources.py`
Each source should have a `source.update(t)` method that will be called every timestep dt in the simulation loop.

- For the case of a **particle beam**, the source is a current in J_z, applied at $x_s, y_s \; \forall \; z$ with a gaussian time profile.
  - q is the charge in [C], typically $\approx 10^{-9}$ C
  - σ_z is the beam size [m]
  - β is the ratio of the beam speed v to the speed of light c

$$J_z(x_s, y_s, z) = \frac{q\beta c}{\sqrt{2\pi\sigma_z}} e^{\frac{-(s-s_0)^2}{2\sigma_z^2}}$$

$$s = z - \beta c t \; ; \quad s_0 = z_{min} - \beta c t$$





Minimizing injection perturbation: ABC

- The **perturbation** appears due to the violation of the continuity law $\tilde{\boldsymbol{S}}\tilde{\boldsymbol{D}}_{\boldsymbol{A}}\left(\frac{\partial \boldsymbol{d}}{\partial t} + \boldsymbol{j}\right) = \boldsymbol{0}$
- We can mitigate it with **boundary conditions**: 1st attempt was the FOEXTRAP* absorbing boundary condition (ABC)

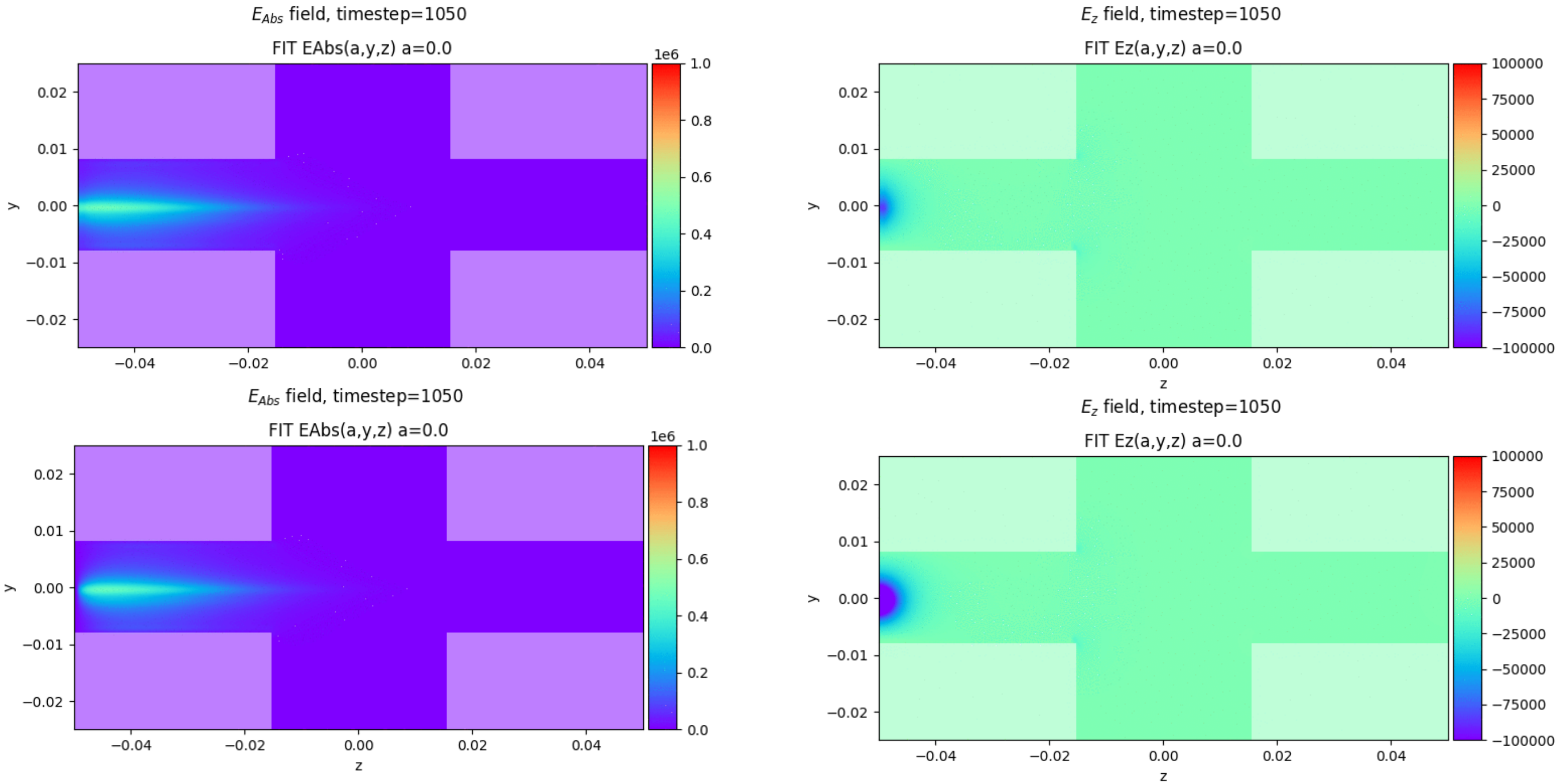
**FOEXTRAP:* First Order Extrapolation (backup slides)

Minimizing injection perturbation: ABC

- The **perturbation** appears due to the violation of the continuity law $\tilde{S}\tilde{D}_A\left(\frac{\partial d}{\partial t} + j\right) = 0$
- We can mitigate it with **boundary conditions**: 1st attempt was the FOEXTRAP* absorbing boundary condition (ABC)

ABC

PEC

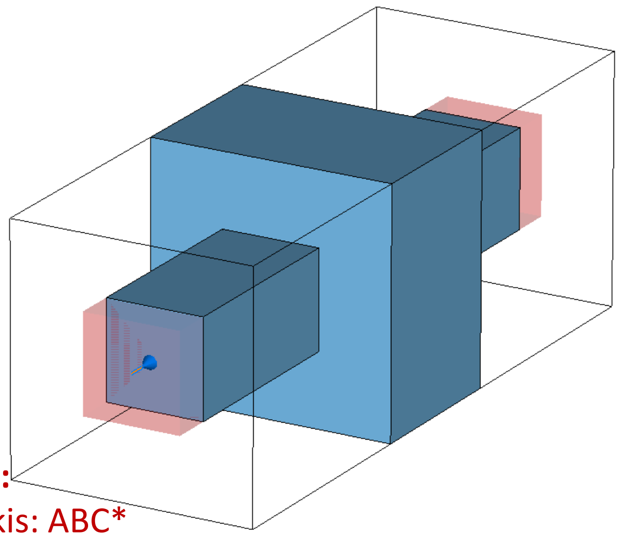


Thanks to Prof. M. Cotelo (UPM) for the ABC idea

**FOEXTRAP:* First Order Extrapolation (backup slides)

Example: PEC cubic cavity

benchmarks/cubcavitymm/



Geometry:
L_{cav} = 30 mm
h_{cav} = 50 mm
w_{cav} = 50 mm
L_{pipe} = 100 mm
h_{pipe} = 15 mm
w_{pipe} = 15 mm

Beam:
σ_z = 18.5 mm (5 GHz)
q = 1e-9 C

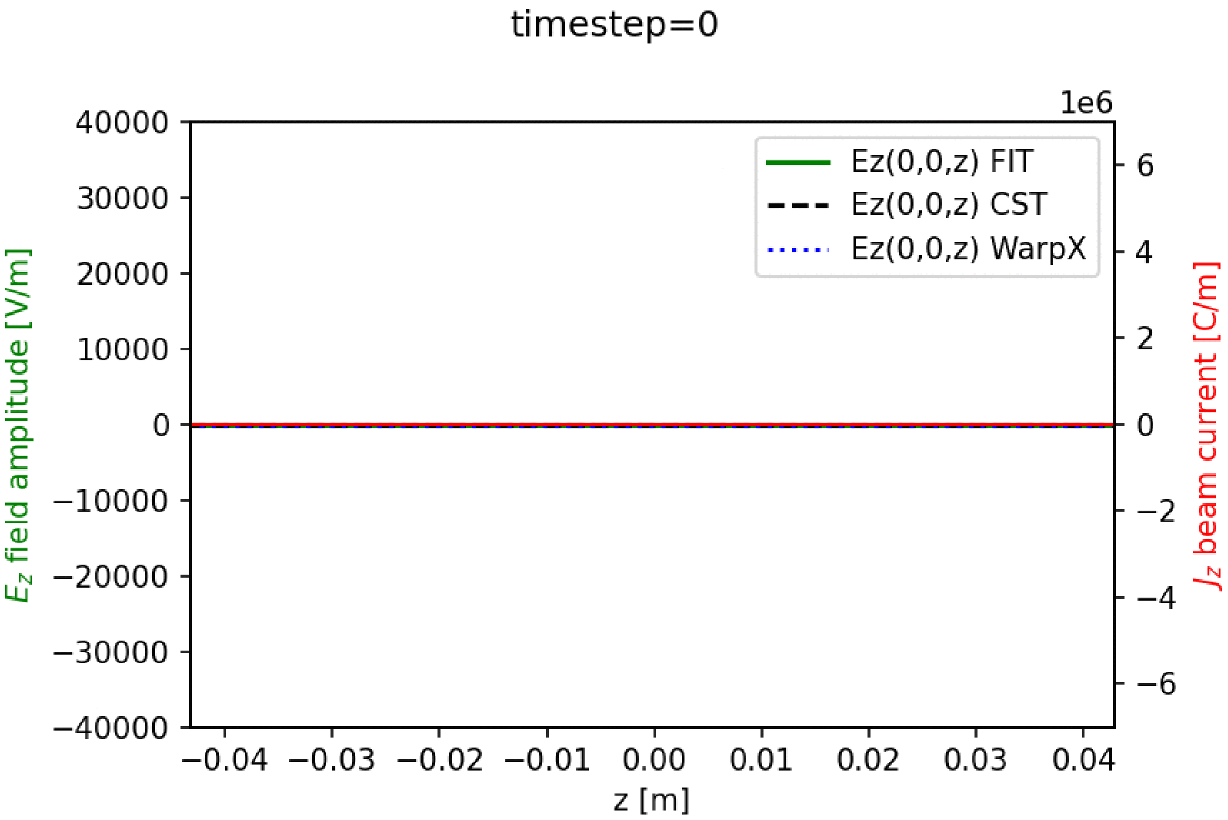
Mesh:
nx, ny, nz = 50, 50, 150
total: 375000 cells

Cutoff frequency: ~10 GHz

BCs:
wakis: ABC*
WarpX: PML*
CST: CPML

Simulation time: for 1m wakelength (8760 timesteps):

- **CST**: 42s, 16 threads, in \`abpimp60g01\`
- **Wakis**: 3m40s , single core in \`abpimp60g01\`
- **WarpX:** 12m, single core, #MP 1e6, in \`abpimp60g01\`

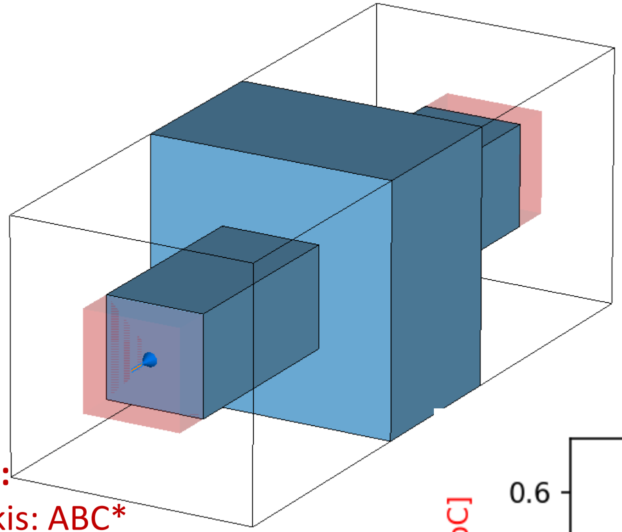


Remarks

- ✓ Very good agreement
- ➢ WarpX shows a non-vanishing injection perturbation while in wakis it disappears
- ➢ Some phase error comes from the CST field monitor extraction (very close to simulation's timestep)

Example: PEC cubic cavity

benchmarks/cubcavitymm/



BCs:
wakis: ABC*
WarpX: PML*
CST: CPML

Geometry:
L_{cav} = 30 mm
h_{cav} = 50 mm
w_{cav} = 50 mm
L_{pipe} = 100 mm
h_{pipe} = 15 mm
w_{pipe} = 15 mm

Beam:
σ_z = 18.5 mm (5 GHz)
q = 1e-9 C

Mesh:
nx, ny, nz = 50, 50, 150
total: 375000 cells

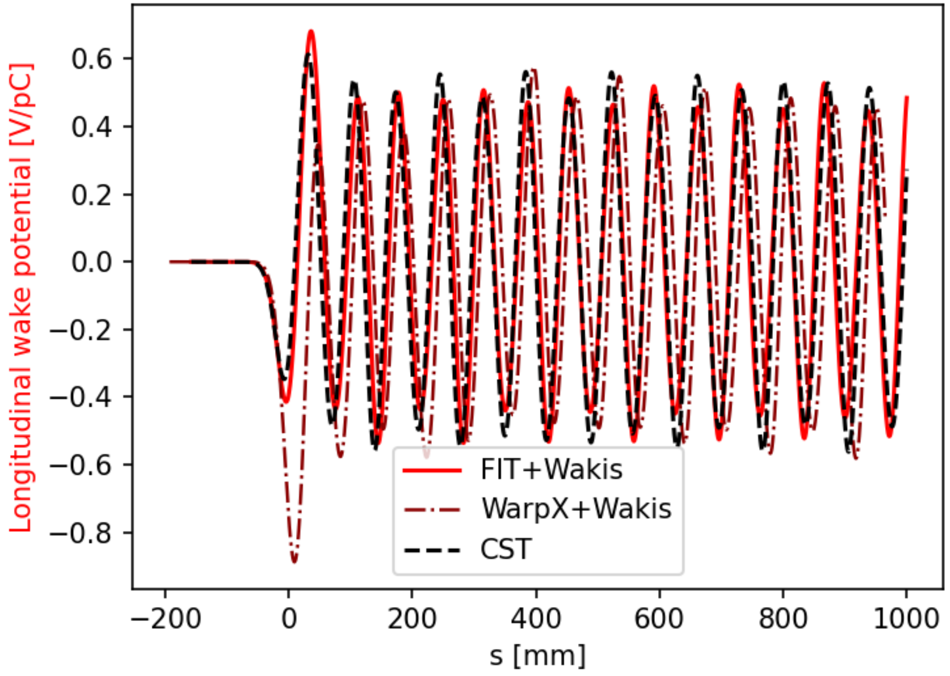
Cutoff frequency: ~10 GHz

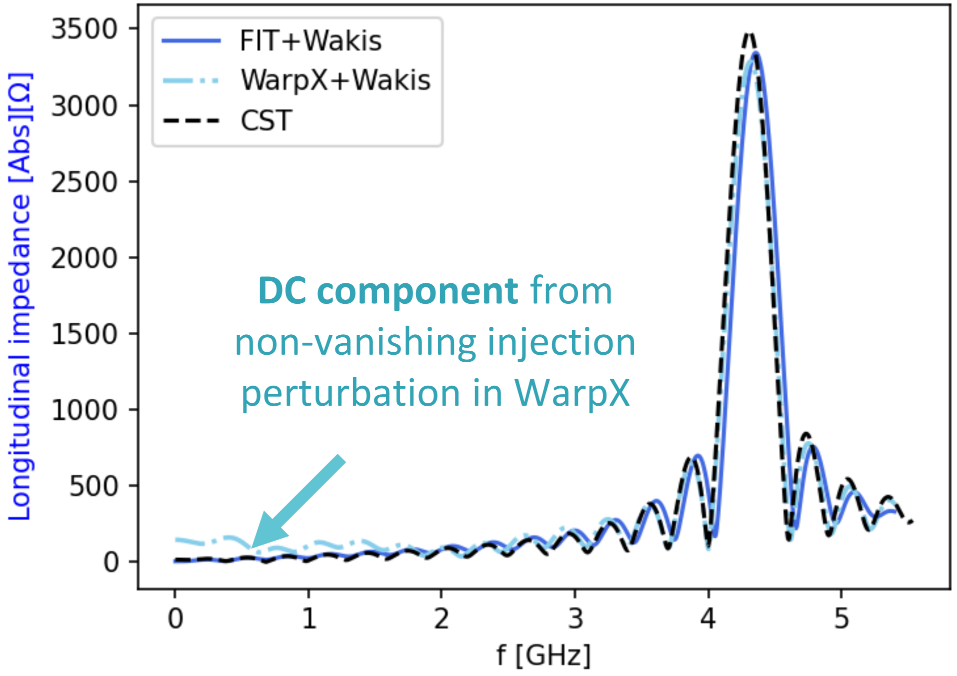
Simulation time: for 1m wakelength (8760 timesteps):

- **CST**: 42s, 16 threads, in \`abpimp60g01\`
- **Wakis**: 3m40s , single core in \`abpimp60g01\`
- **WarpX:** 12m, single core, #MP 1e6, in \`abpimp60g01\`

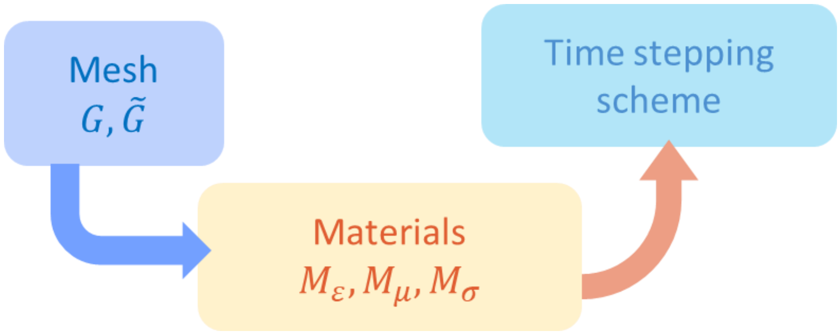
*last 10 cells in z- and z+ were removed for the wake calculation to mitigate injection perturbation

Simulation wakis vs WarpX vs CST: (keeping same mesh)





Outline


Mesh
$G, \tilde{G}$
Materials
$M_\varepsilon, M_\mu, M_\sigma$
Time stepping scheme

Conductivity in time domain

Relative permittivity is often a frequency-dependent, complex quantity:

$$\varepsilon_r(\omega) = \varepsilon'_r(\omega) - j\varepsilon_r''(\omega)$$

The **simple model** for a metal with broadband conductivity σ is:

$$\varepsilon_r(\omega) = \varepsilon_r - j\frac{\sigma}{\omega\varepsilon_0}$$

Time-stepping scheme:

$$\boldsymbol{h}^{n+1} = \boldsymbol{h}^{n} - \Delta t\,\widetilde{\boldsymbol{D}}_s \boldsymbol{D}_\mu^{-1}\boldsymbol{D}_A^{-1}\boldsymbol{C}\boldsymbol{e}^{n+0.5}$$

$$\boldsymbol{e}^{n+1.5} = \boldsymbol{e}^{n+0.5} + \Delta t\boldsymbol{D}_s\widetilde{\boldsymbol{D}}_\varepsilon^{-1}\widetilde{\boldsymbol{D}}_A^{-1}\widetilde{\boldsymbol{C}}\boldsymbol{h}^{n} - \widetilde{\boldsymbol{D}}_\varepsilon^{-1}\boldsymbol{j}_{beam}^{n}$$

? How to implement this in a **time domain (TD)** simulations?

- In TD, fields are purely real $\Re$ → all matrix elements should be real
- Introducing frequency dependence involves a de-convolution… (computationally expensive 💲💲)

Conductivity in time domain

Relative permittivity is often a frequency-dependent, complex quantity:

$$\varepsilon_r(\omega) = \varepsilon'_r(\omega) - j\varepsilon_r''(\omega)$$

The **simple model** for a metal with broadband conductivity σ is:

$$\varepsilon_r(\omega) = \varepsilon_r - j\frac{\sigma}{\omega\varepsilon_0}$$

From the **Frequency domain Maxwell equations**:

Ampere's law: $\nabla \times H = J_{beam} + j\omega D = J_{beam} + j\omega\varepsilon E;$

$$\nabla \times H(\omega) = J_{beam} + j\omega\varepsilon_0\left(\varepsilon_r + \frac{\sigma}{j\omega\varepsilon_0}\right)E(\omega);$$

$$\nabla \times H(\omega) = J_{beam} + j\omega\varepsilon' E(\omega) + \sigma E(\omega)$$

To convert to TD: $j\omega \leftrightarrow {}^{d}/_{dt}, \quad A(\omega) \leftrightarrow A(t)$*

$$\nabla \times H(t) = J_{beam} + \frac{d(\varepsilon' E(t))}{dt} + \sigma E(t)$$

Time-stepping scheme:

$$\boldsymbol{h}^{n+1} = \boldsymbol{h}^{n} - \Delta t\, \widetilde{\boldsymbol{D}}_s \boldsymbol{D}_\mu^{-1} \boldsymbol{D}_A^{-1} \boldsymbol{C} \boldsymbol{e}^{n+0.5}$$

$$\boldsymbol{e}^{n+1.5} = \boldsymbol{e}^{n+0.5} + \Delta t \boldsymbol{D}_s \widetilde{\boldsymbol{D}}_\varepsilon^{-1} \widetilde{\boldsymbol{D}}_A^{-1} \widetilde{\boldsymbol{C}} \boldsymbol{h}^{n} - \widetilde{\boldsymbol{D}}_\varepsilon^{-1} \boldsymbol{j}_{beam}^{n}$$

Time-stepping scheme with σ:

$$\boldsymbol{h}^{n+1} = \boldsymbol{h}^{n} - \Delta t\, \widetilde{\boldsymbol{D}}_s \boldsymbol{D}_\mu^{-1} \boldsymbol{D}_A^{-1} \boldsymbol{C} \boldsymbol{e}^{n+0.5}$$

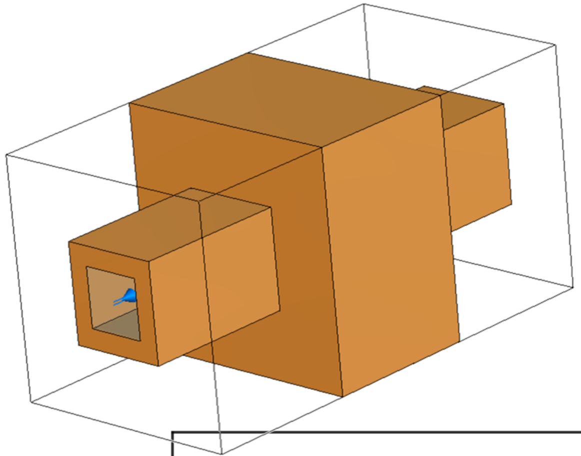
$$\boldsymbol{e}^{n+1.5} = \boldsymbol{e}^{n+0.5} + \Delta t \boldsymbol{D}_s \widetilde{\boldsymbol{D}}_\varepsilon^{-1} \widetilde{\boldsymbol{D}}_A^{-1} \widetilde{\boldsymbol{C}} \boldsymbol{h}^{n} - \widetilde{\boldsymbol{D}}_\varepsilon^{-1} \boldsymbol{j}_{beam}^{n} - \widetilde{\boldsymbol{D}}_\varepsilon^{-1} \widetilde{\boldsymbol{D}}_\sigma \boldsymbol{e}^{n+0.5}$$

+ constitutive relation for $\boldsymbol{J}$

$$J(t) = \sigma E(t) + J_{beam}(t)$$

*Raymond Rumpf, *Electromagnetic and Photonic Simulation for the Beginner*, Artech, 2022.

Example: lossy cubic cavity



Conductivity: 10 S/m

Skin-Depth: $\delta = \sqrt{\frac{1}{\pi f \mu \sigma}} \approx 2$ mm

N_{cells}: 542754

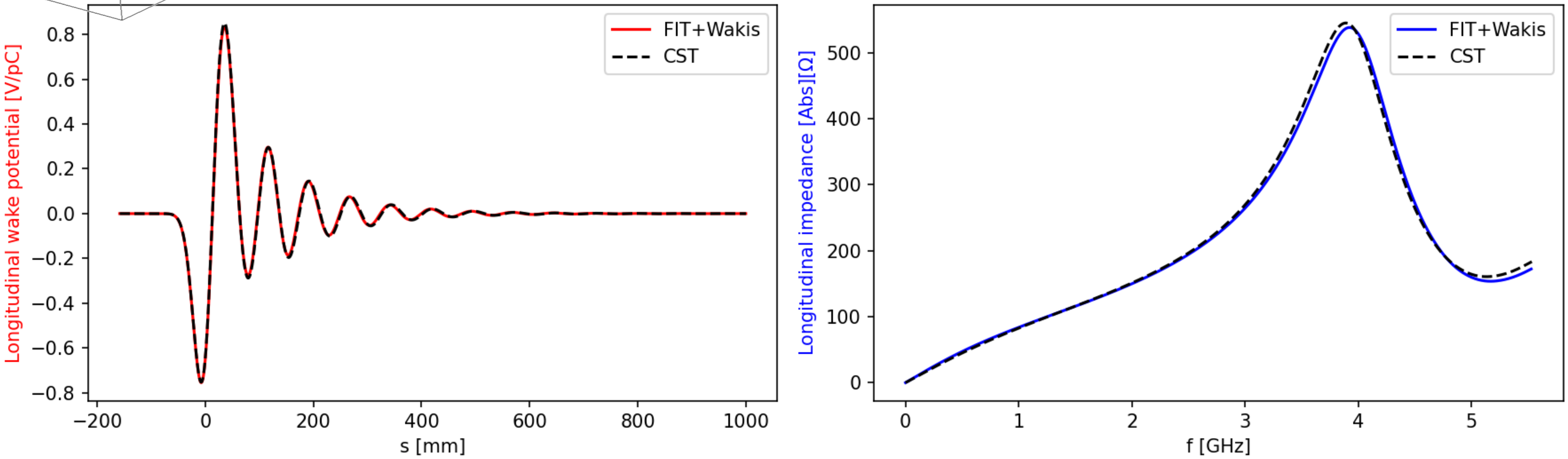
Runtime: 4', 30'' , single core in `abpimp60g01`

Δ_{cell}: 0.5 mm > 3δ

Remarks:

- To match CST results, we had to remove 10 cells z-/z+ and increase the mesh by 20%
- Low conductivity was chosen to have a fast decaying wake

Benchmark with CST Wakefield Solver

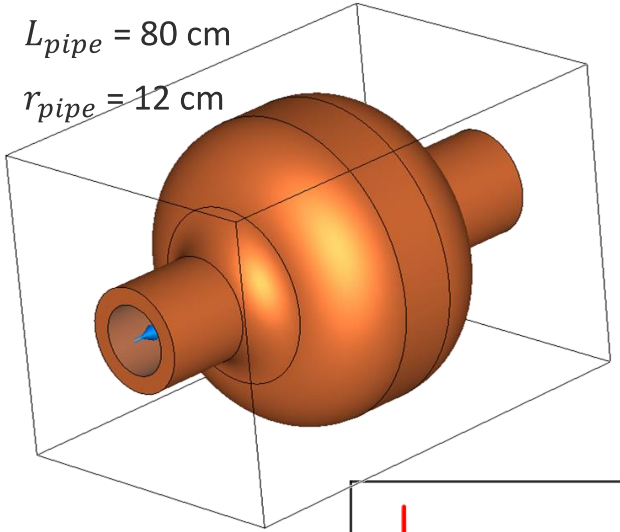


*CST's Default Acc. Cavity model

Example: lossy fancy-shaped* cavity

L_{pipe} = 80 cm

r_{pipe} = 12 cm



Conductivity: 1000 S/m

Skin-Depth: $\delta = \sqrt{\frac{1}{\pi f \mu \sigma}} \approx \mathbf{0.5}\text{mm}$

N_{cells}: 271600,

Wakelength: 100 m

Δ_{cell}: 1.5 mm < 3δ

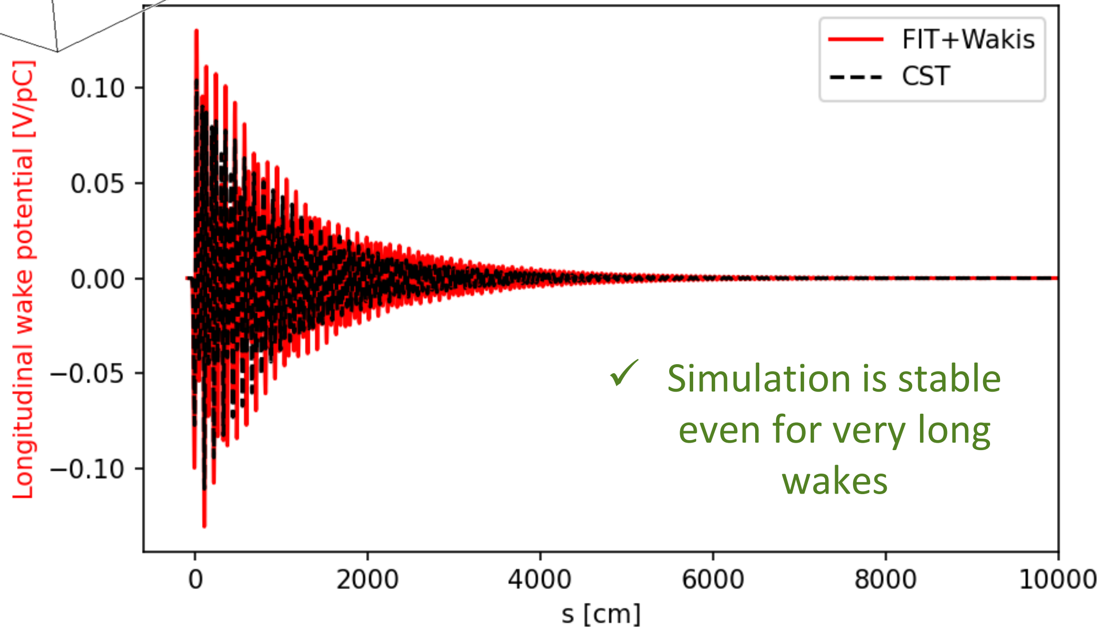
Runtime: 20' 40'' single core in `abpimp60g01`

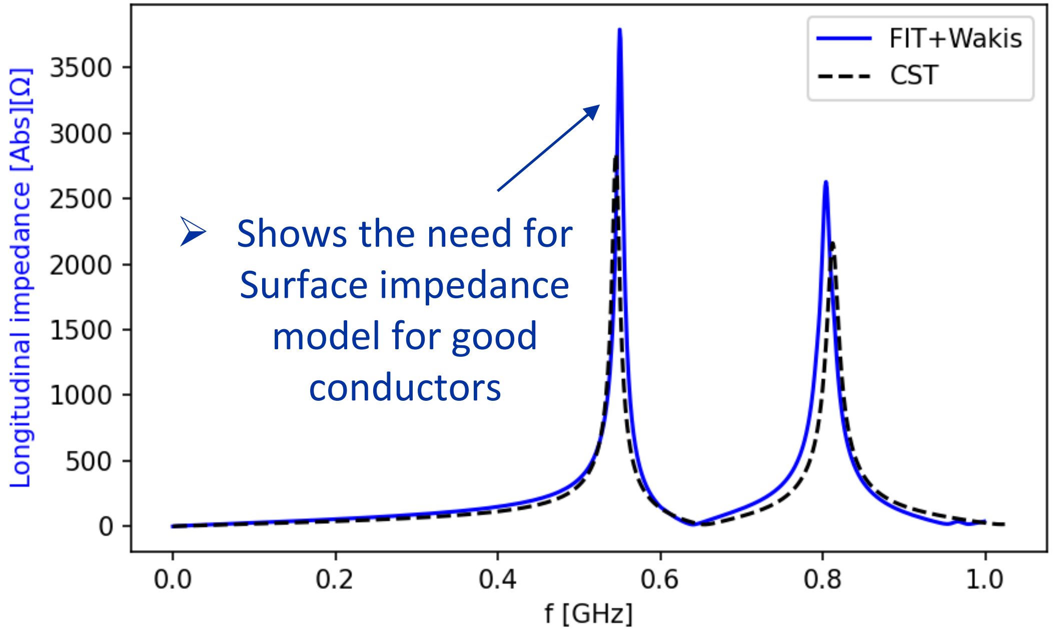
Remarks:

- Staircased grid can induce the frequency shift at 800 MHz
- Skin Depth** is not well modelled → difference in shunt impedance

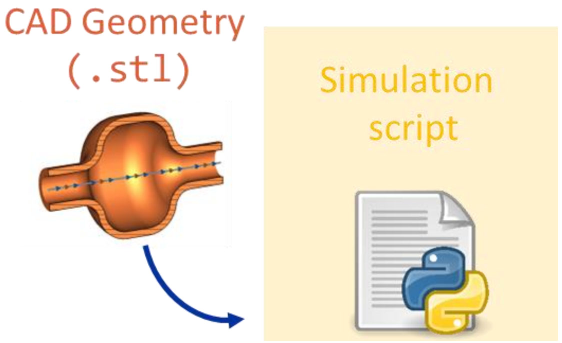
for $\sigma > 100\ S/m$, CST uses **surface impedance model (Leontovich)

Benchmark with CST Wakefield Solver



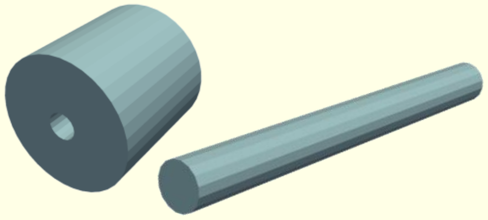


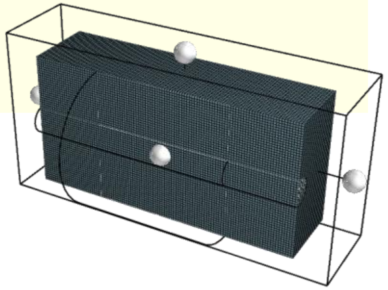
Outline



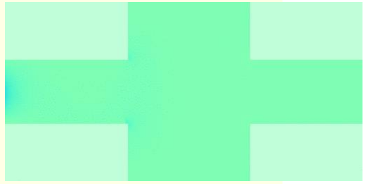
Simulation script example

```python
from wakis import GridFIT3D, SolverFIT3D, WakeSolver
import pyvista as pv

# ---------- Domain and Grid setup ---------
# Number of mesh cells
Nx = 57
Ny = 57
Nz = 109
#dt = 5.707829241e-12

# Geometry Import
stl_cavity = 'cavity.stl'
stl_pipe = 'beampipe.stl'
stl_solids = {'cavity': stl_cavity, 'pipe': stl_pipe}

# Materials
stl_materials = {'cavity': 'vacuum', 'pipe':  'vacuum'}
background = [1.0, 1.0, 100] # lossy metal [ε_r, μ_r, σ]

# Domain bounds (from stl)
surf = pv.read(stl_cavity) + pv.read(stl_pipe)
xmin, xmax, ymin, ymax, zmin, zmax = surf.bounds

# Set grid and geometry
grid = GridFIT3D(xmin, xmax, ymin, ymax, zmin, zmax, Nx, Ny, Nz,
                stl_solids=stl_solids,
                stl_materials=stl_materials)
#grid.inspect()
```





```python
# ------------ Beam source ----------------
# Beam parameters and wake obj.
beta = 0.8          # beam relativistic beta
sigmaz = beta*6e-2  # [m] -> multiplied by beta to have f_max cte
q = 1e-9            # [C]
xs = 0.             # x source position [m]
ys = 0.             # y source position [m]
xt = 0.             # x test position [m]
yt = 0.             # y test position [m]
# tinj = 8.53*sigmaz/(beta*c)  # injection time offset [s]

wake = WakeSolver(q=q, sigmaz=sigmaz, beta=beta,
            xsource=xs, ysource=ys, xtest=xt, ytest=yt,
            save=True, logfile=True)

# ----------- Solver & Simulation ----------
# boundary conditions and solver obj.
bc_low=['pec', 'pec', 'pec']
bc_high=['pec', 'pec', 'pec']
solver = SolverFIT3D(grid, wake,
                     bc_low=bc_low, bc_high=bc_high,
                     use_stl=True, bg=background)

# Run wakefield time-domain simulation
wakelength = 5. #[m]
add_space = 10  # no. cells to remove for the wake calculation

solver.wakesolve(wakelength=wakelength, add_space=add_space,
                 plot=True, plot_every=30, save_J=True,
                 **plotkw)
```

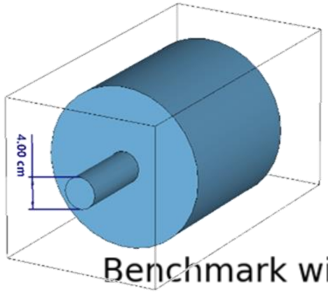


Example: results for different β

benchmarks/betacavity/

Simulation of a cylindrical pillbox* below cut-off for different relativistic β values

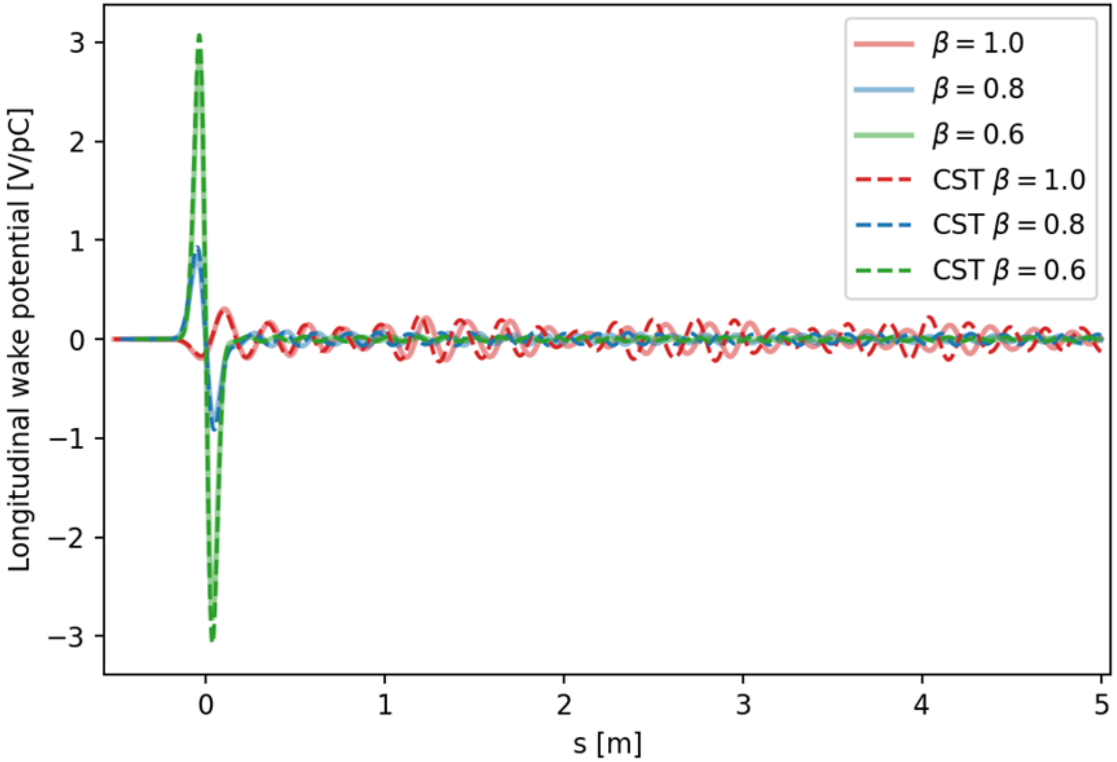
*Model by Elena Macchia

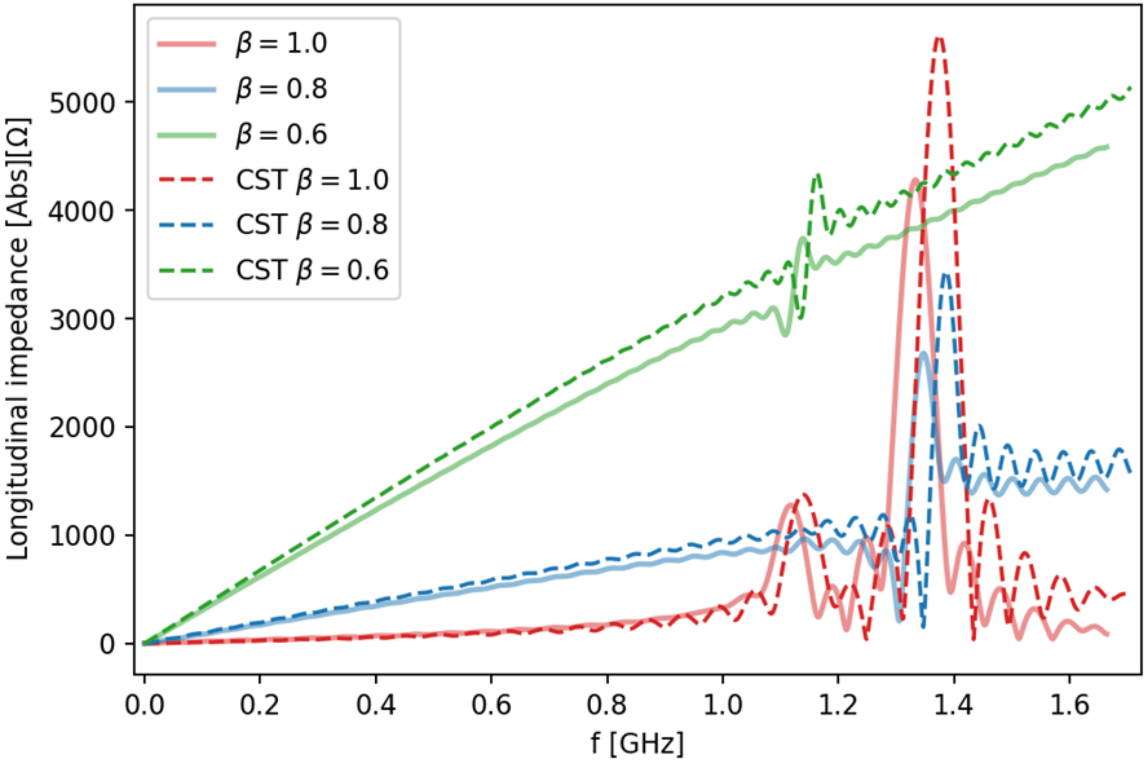


Benchmark with CST Wakefield Solver

Work in progress!

- ✓ 1st attempt shows some agreement, but mesh needs to be optimized
- ✓ The behaviour with β is consistent





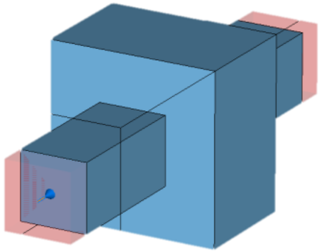
Outline

CERN IFN-GV

Challenges: Simulations above cut-off f

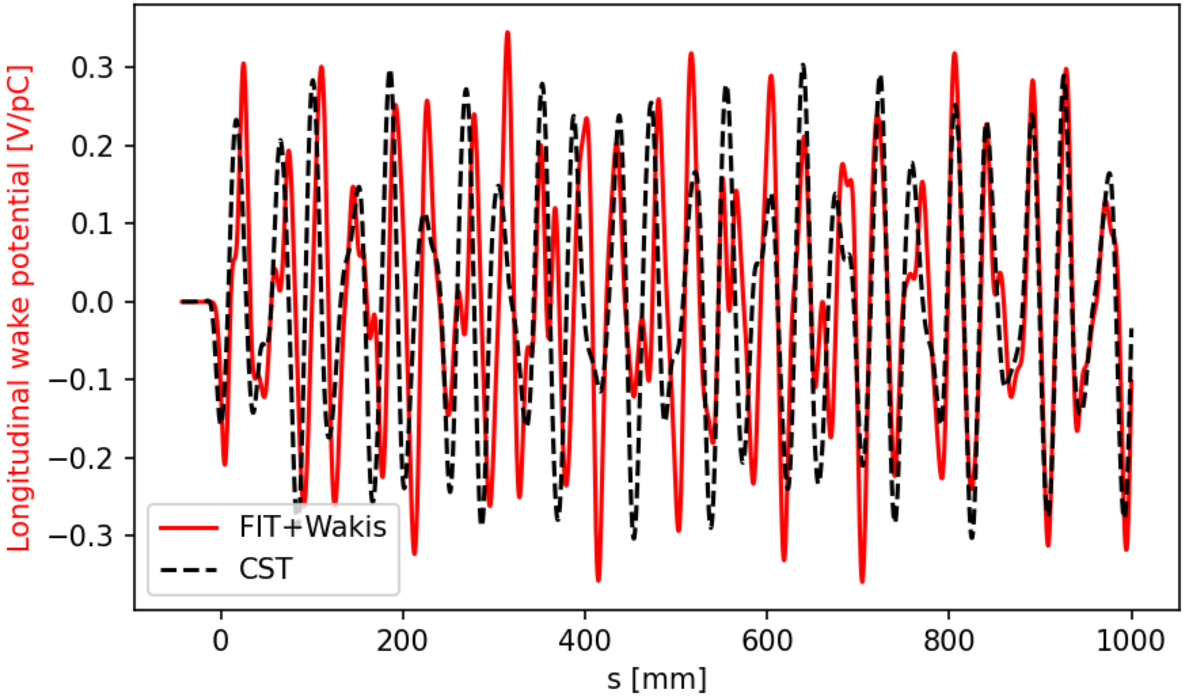
When the bunches are short, they excite to higher frequencies: $f_{max} = \frac{\beta c}{3\sigma_z}$

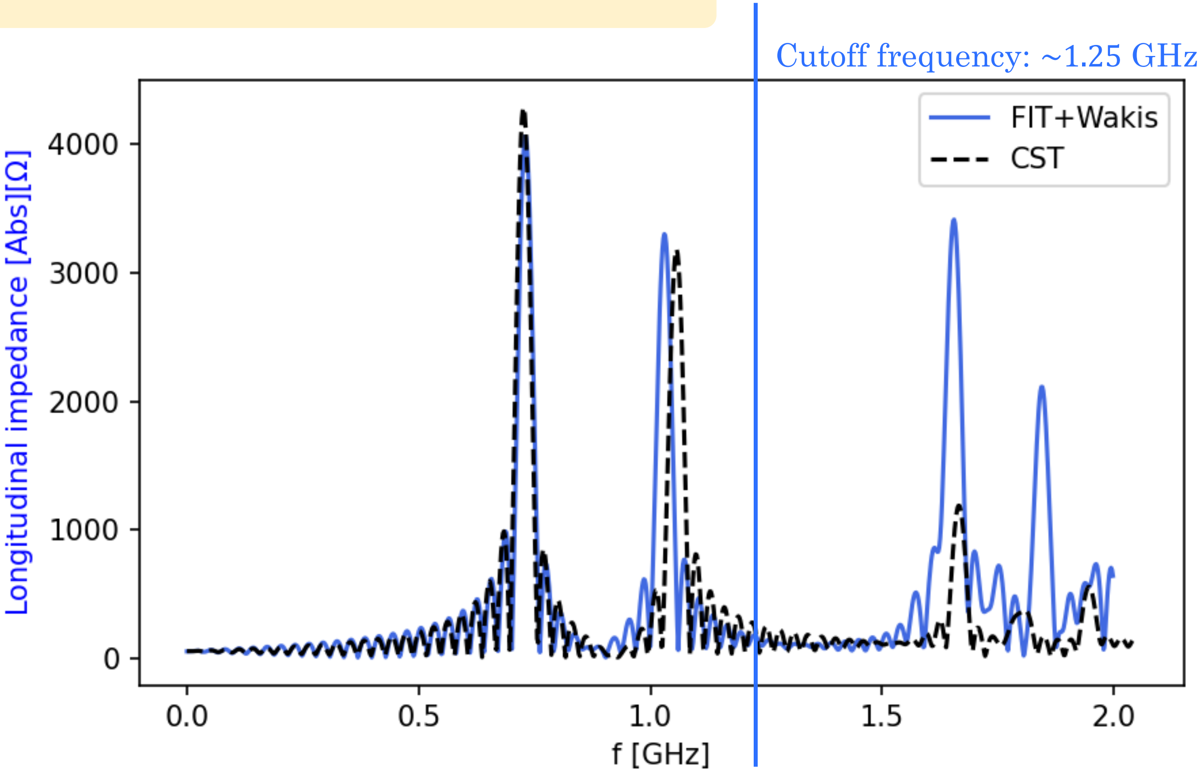
When $f_{max} > f_{cutoff}$ of the pipe, some modes are in propagation and reach the boundaries



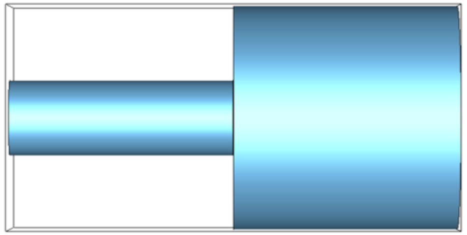
`wakis` does not show good agreement above cut-off (yet)

Perfect matching layers (PML) BCs are needed to simulate propagating modes

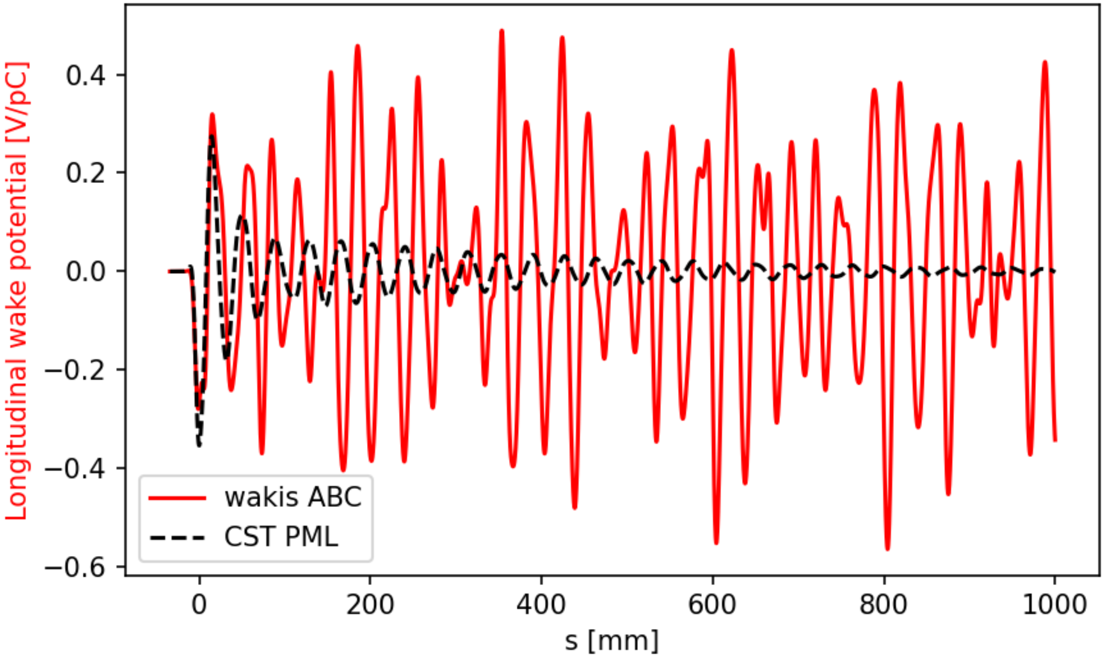


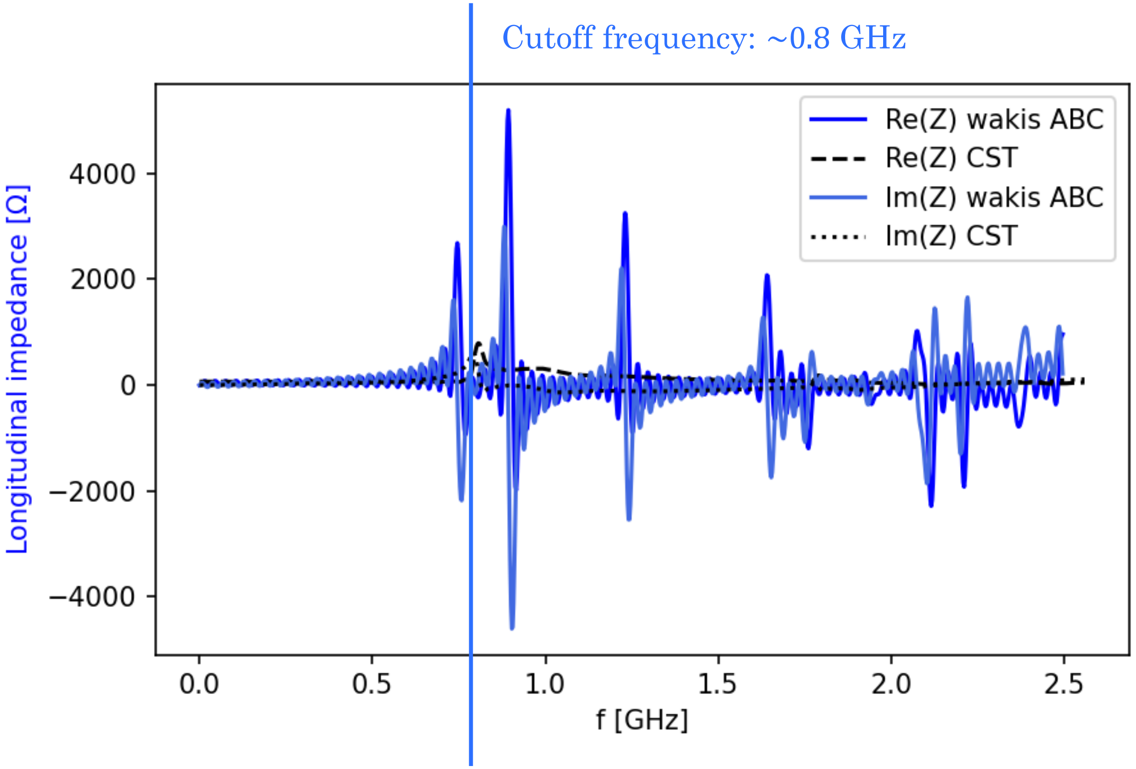


Challenges: Simulations above cut-off *f*



This becomes more evident in a non-resonant structure, like a transition (i.e., Step-out)
All modes should propagate, but in `wakis` they are reflected back. To simulate propagating modes, we need the **PML** boundary conditions





More challenges for the near future

I. PML & Simulations above cutoff

II. Staircased grid

III. Numerical dispersion

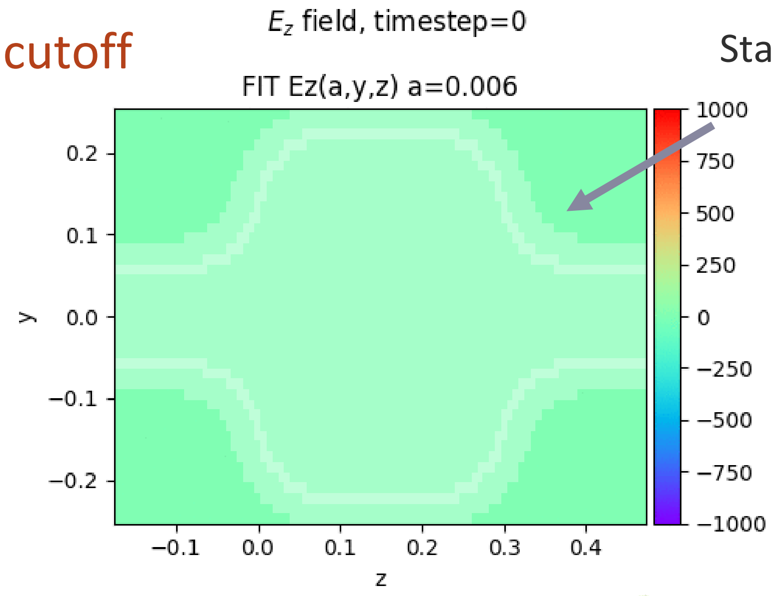
IV. GPU acceleration

V. Surface impedance

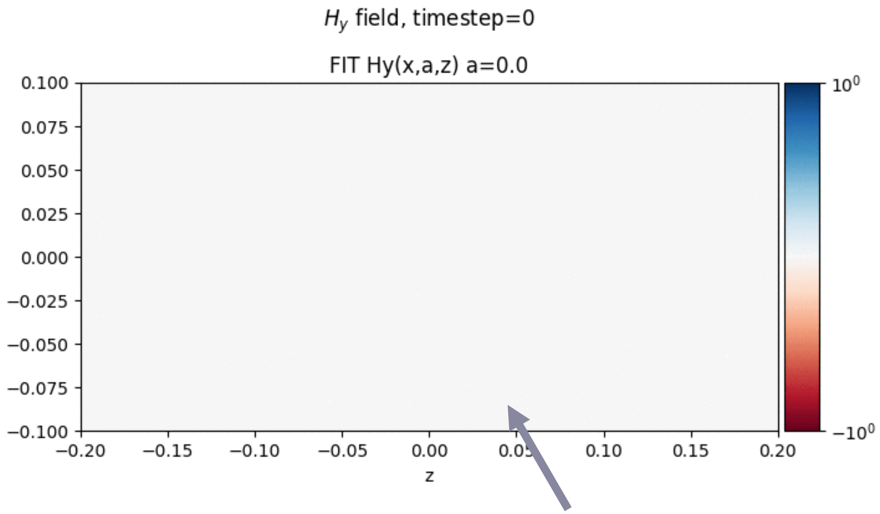
VI. Wake extrapolation (genetic algorithm)

VII. Frequency dependent (dispersive) materials

VIII. Frequency domain monitors



Staircased grid with curved geometry generates artifacts and difficulties convergence



For some optic applications, the code shows numerical dispersión

examples/script_wavepacket_fit.py

CuPy

Acceleration should be possible with cupy since it supports `scipy.sparse`

Conclusions

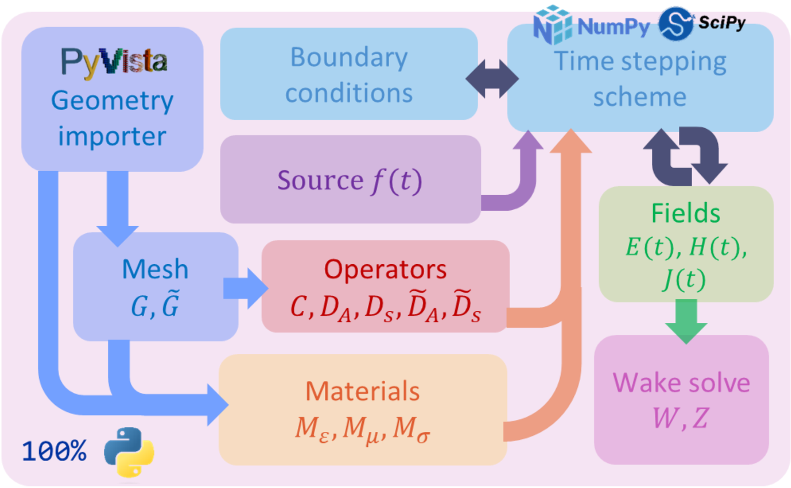
- Developed a **3D Electromagnetic and Wake Solver in time domain**, 100% in python: `wakis`
  1. Uses the **Finite Integration Technique** and `scipy.sparse` matrices, allowing for fast computations, extendable to GPU
  2. Different **boundary conditions**: PEC, PMC, Periodic, ABC
  3. **Importing CAD geometry** with `pyvista`
  4. **Simulate materials with ε, μ, σ** with diagonal tensors. Possibility of anisotropic properties.
  5. Simulate several sources: Planewave, gaussian wave packet, and the **particle beam for different β**
  6. Several **on the fly built-in plotting** capabilities in 1d, 2d, 3d
  7. **Interactive visualization** methods to inspect the grid and the geometry
  8. +10 examples and benchmark simulations (⌛~2' to 20') already available on GitHub.

- `wakis` solver has been **benchmarked with CST Wakefield Solver®** with pillbox cavities of different materials and geometries, below cut-off.

- It is built also as a **general-purpose 3D TD EM code**, capable of simulating optical lenses, diffraction gratings and much more.

➢ However, **many challenges need to be investigated**: simulations above cutoff with PML boundaries, advanced mesh generation, GPU acceleration, numerical dispersion correction, surface impedance condition… That will be addressed during the PhD!

`wakis (v0.3)`



GitHub

https://github.com/ImpedanCEI/FITwakis

Thank you ☺ !!!

wakis:

3D Electromagnetic Time- Domain

Wake and Impedance Solver

Elena de la Fuente García (BE–ABP–CEI)

Bibliography used

1990 Joint US-CERN Accelerator Course
Hilton Head, So. Carolina

Wake Fields and Impedances

T. Weiland, R. Wanzenberg

Contents

1

i. Wakefields and Impedances [T. Weiland, 1991]

ii. TE/TM FIT for accelerators [I. Zarg, 2005]

iii. Open boundaries for FIT [MC. Balk, 2005]

iv. 2D expansion [I. Zarg, 2015]

v. 2D freq. domain [R. SChumann, 2000]

vi. Frequency domain FIT and FEM Uwe Niedermayer PhD thesis [2015]

vii. Eigenmode + MPI + Gyrotropic materials Klaus Klopfer PhD thesis [2014]

viii. EMcLAW: An unsplit Godunov method for Maxwell 's Equations. (UPM) Moreno, José A. PhD Thesis [2020]

Absorbing boundary condition (ABC)

A first attempt to reduce the perturbation of the E_z field when the beam current enters/exits it to use absorbing boundary conditions (ABC)

- Since PML formulation is complex, the simplest ABC, the FOEXTRAP, was tested first. This is a first order extrapolation that mimics a continuous field at the boundary cells

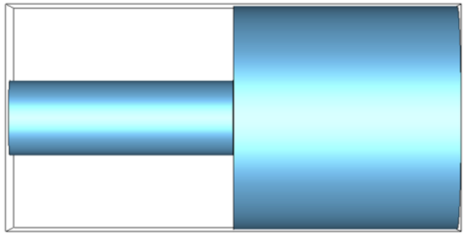
```
    def one_step(self):

        if self.step_0:
            self.set_ghosts_to_0()
            self.step_0 = False

            #if self.use_conductors:
                #self.set_field_in_conductors_to_0()

        self.H.fromarray(self.H.toarray() -
                         self.dt*self.tDsiDmuiDaC*self.E.toarray()
                         )

        self.E.fromarray(self.E.toarray() +
                         self.dt*(self.itDaiDepsDstC * self.H.toarray() - self.iDeps*self.J.to
                         )

        #update ABC
        if self.activate_abc:
            self.update_abc()
```

It has to be updated every timestep

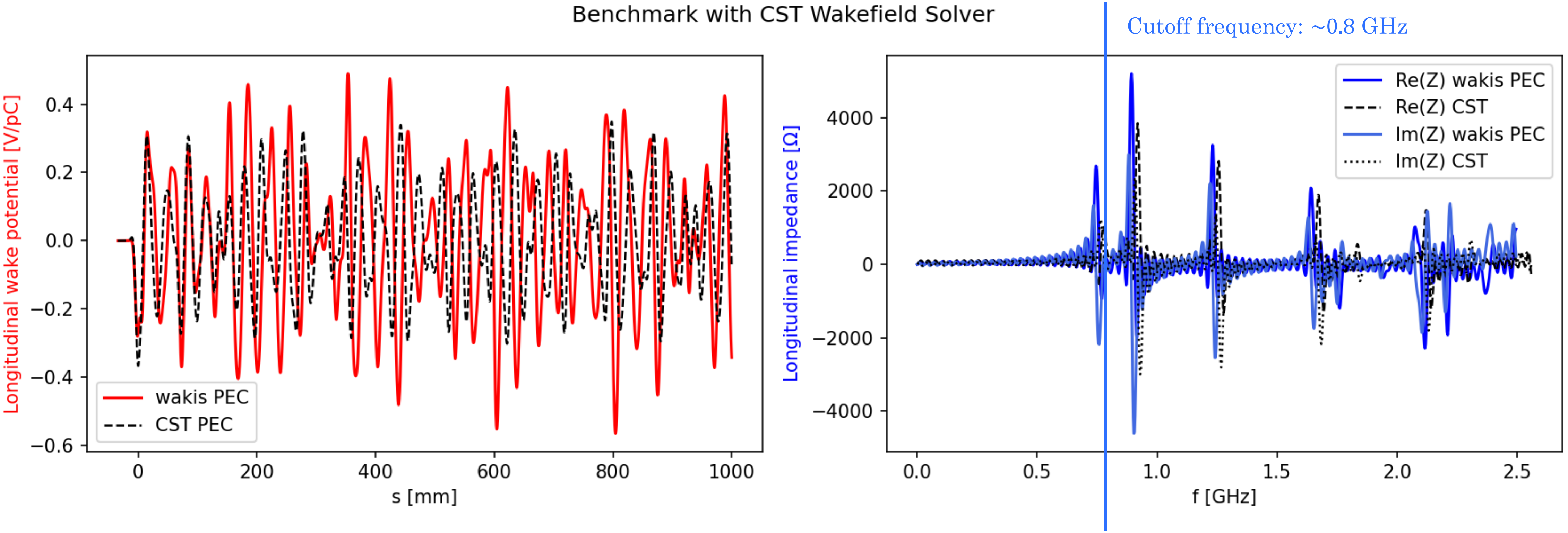
```
    def apply_bc_to_C(self):

        # Absorbing boundary conditions ABC
        if any(True for x in self.bc_low if x.lower() == 'abc'):
            self.activate_abc = True

    def update_abc(self):
        '''
        Apply ABC algo to the selected BC,
        to be applied after each timestep
        '''

        if self.bc_low[0].lower() == 'abc':
            for d in ['x', 'y', 'z']:
                self.E[0, :, :, d] = self.E[1, :, :, d]
                self.H[0, :, :, d] = self.H[1, :, :, d]
```

Same for all 6 boundaries (low and high, x, y, z)

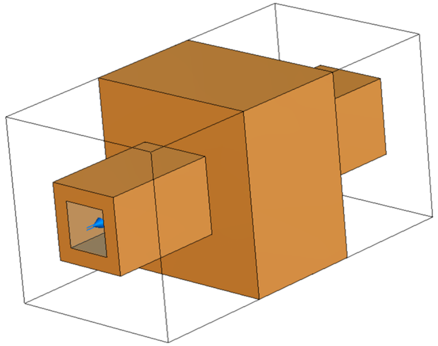
Challenges: Simulations above cut-off f

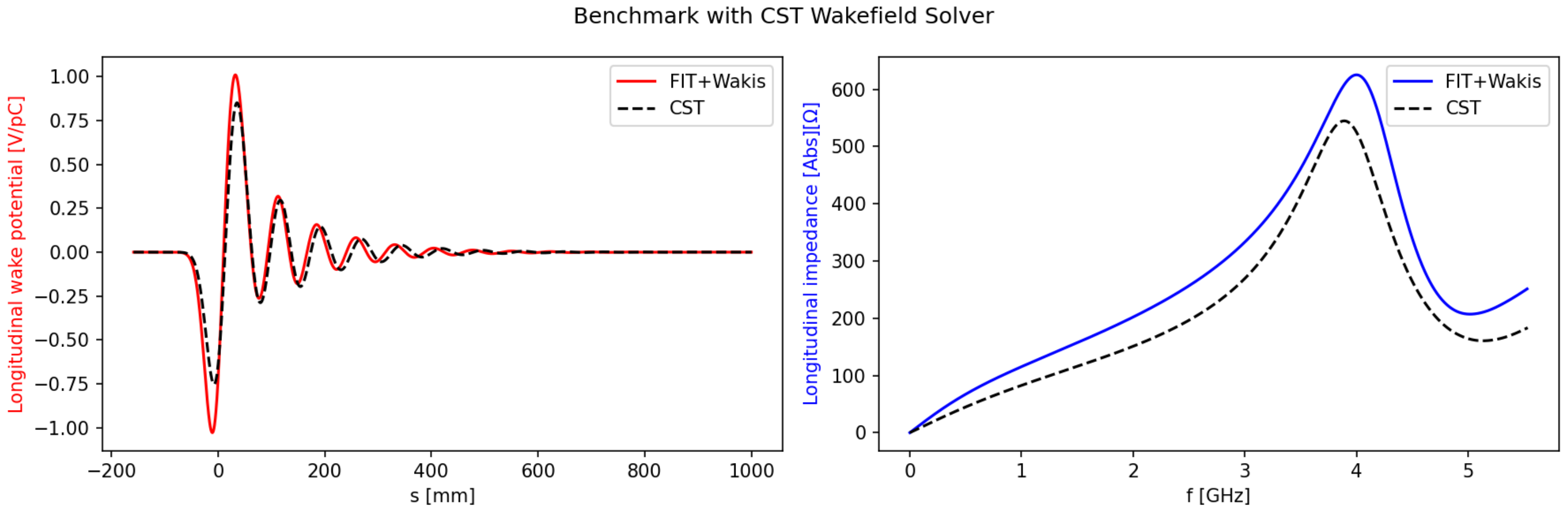


The agreement is okay when we use PEC boundaries In both wakis and CST®

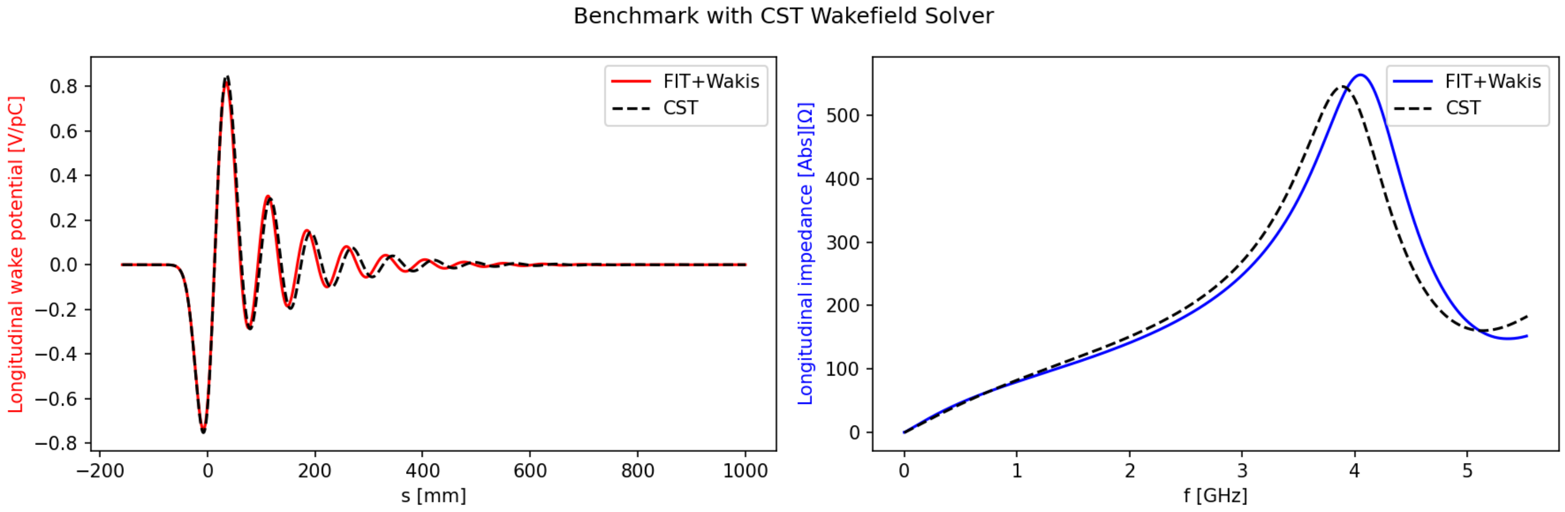


Optimizing results with wakis





If we remove the last 8 cells in z-/z+ with the add_space parameter



Conductivity: 10 S/m

Skin-Depth: $\delta = \sqrt{\frac{1}{\pi f \mu \sigma}} \approx 2$ mm

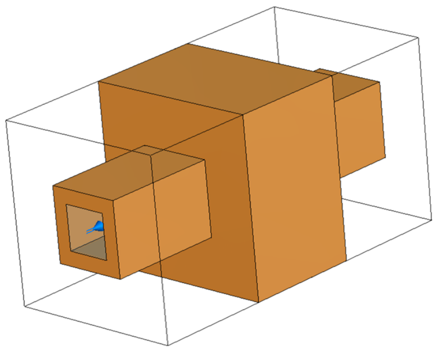
N_{cells}: 542754

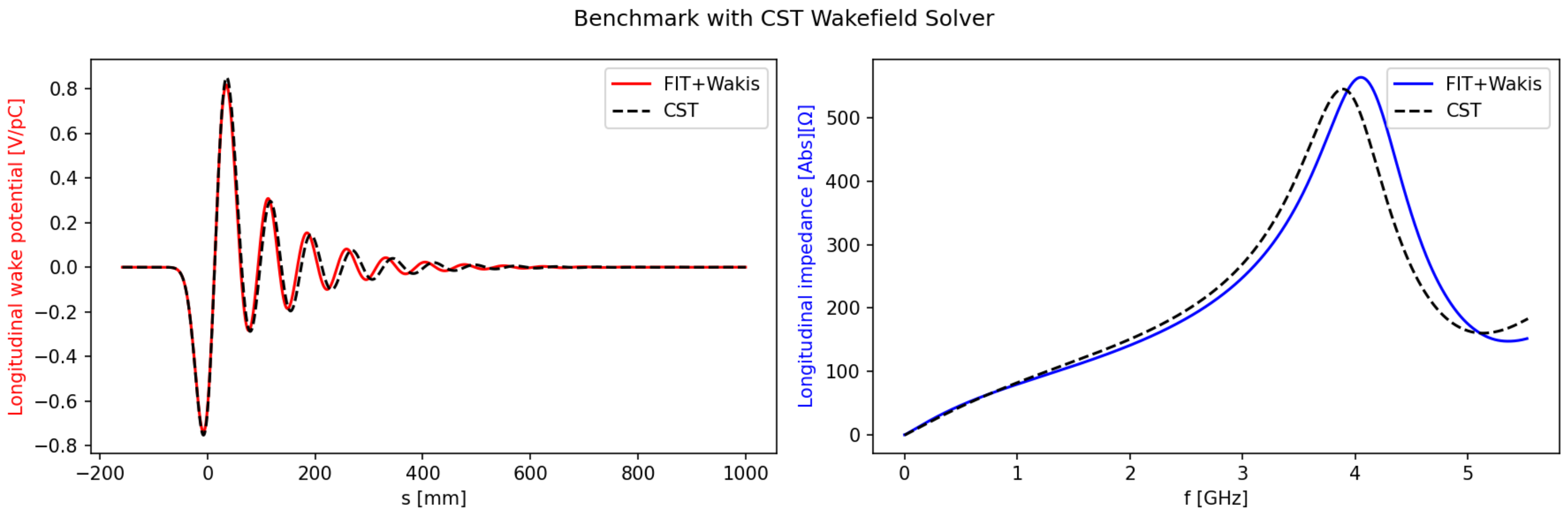
Runtime: 4’, 30’’ ,

single core in `abpimp60g01`

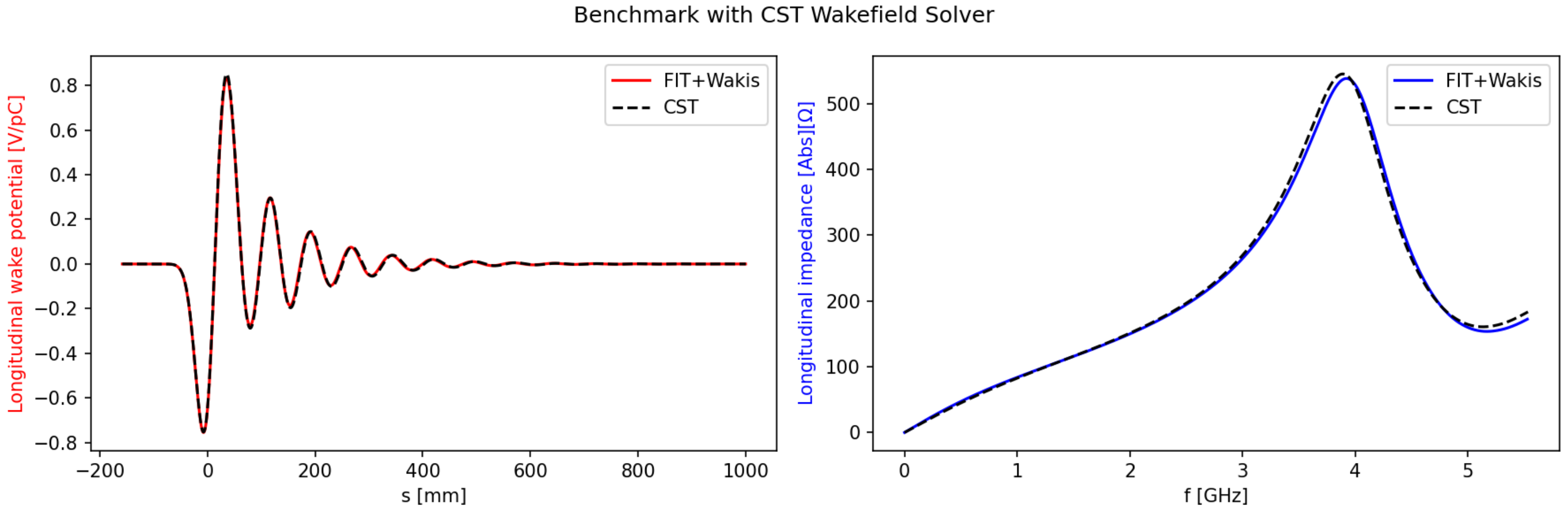
Δ_{cell}: 0.5 mm > 3δ

Optimizing results with wakis





Then, if we increase the mesh resolution by 15%



Conductivity: 10 S/m

Skin-Depth: $\delta = \sqrt{\frac{1}{\pi f \mu \sigma}} \approx 2$ mm

N_{cells}: 542754

Runtime: 4’, 30’’ ,

single core in `abpimp60g01`

Δ_{cell}: 0.5 mm $> 3\delta$

FITwakis GitHub overview

GitHub

elenafuengar new benchmark with a bigger cavity above cutoff		8c6a436 · 14 hours ago · 182 Commits
benchmarks	new benchmark with a bigger cavity above cutoff	14 hours ago
examples	gaussian wave packet example	last week
.gitignore	include cst	last week
conductors.py	Added implicit function conductor	3 years ago
conductors3d.py	fixing a bug in sphere conductor	4 years ago
field.py	updated __add__ to sum two Field objects	4 months ago
grid2D.py	fixing a small bug	3 years ago
grid3D.py	add conductors functions to fit	4 months ago
gridFIT3D.py	small bug fix for stl_scale	2 months ago
materials.py	typo	3 months ago
pmlBlock2D.py	fixing a small bug	4 years ago
pmlBlock3D.py	3D PMLs now working	4 years ago
solver2D.py	Modified 2d em soolver	3 years ago
solver3D.py	change CFL to defaul 0.5	last month
solverFIT3D.py	add dt as parameter	14 hours ago
wakeSolver.py	add wakelength to init	2 days ago

FDTD EM solver by Lorenzo

FIT EM Solver

Wake Solver

Benchmarks vs CST, WarpX …

Examples & university tests:

- Plane wave propagation
- Gaussian wavepacket propagation
- Cubic Resonator

Field class to manage matrix formulation
$E, H, J, \varepsilon, \mu$ are instances of this class

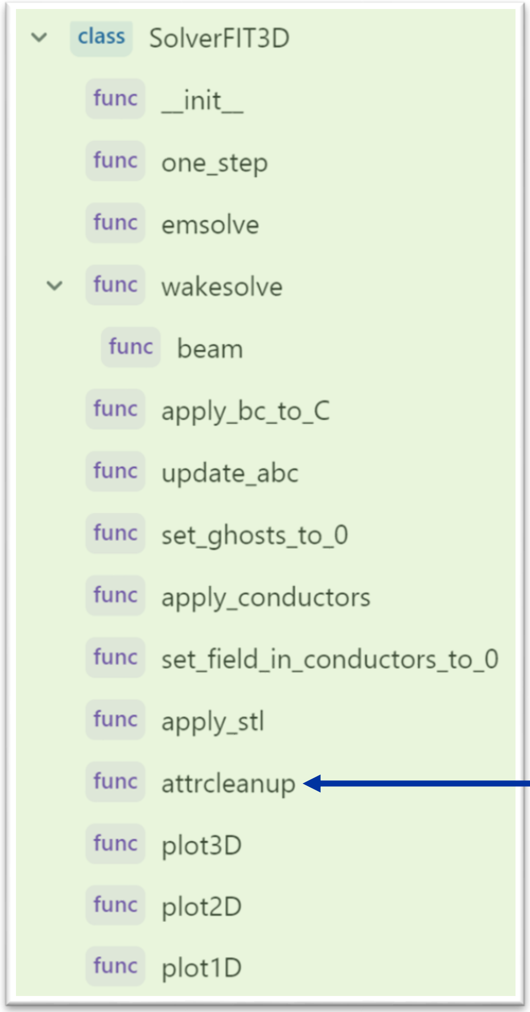
GridFIT3D class in charge of STL importer and grid definition

Pre-defined materials library (vacuum, dielectric, PEC)

SolverFIT3D class that solves Maxwell equations

Wakis(v0.2) code is refactored into class WakeSolver

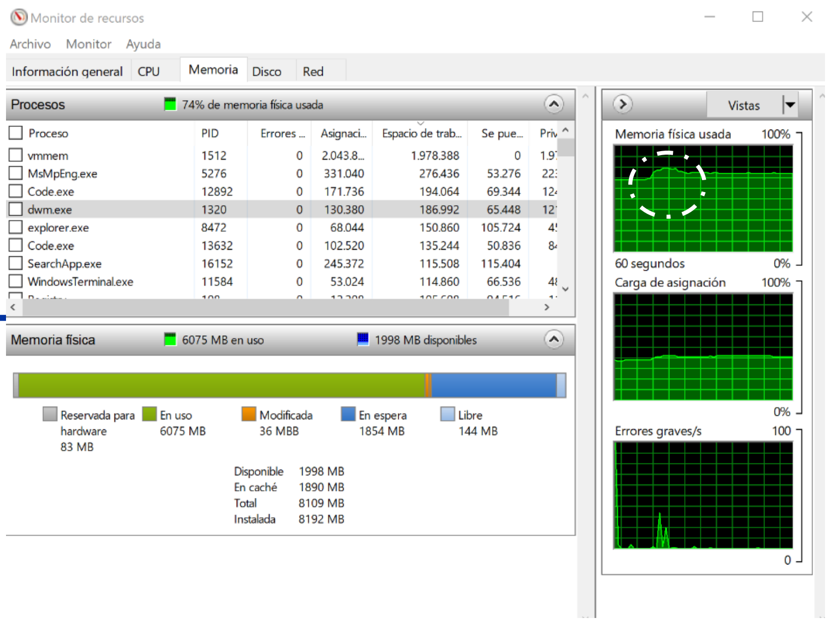
SolverFIT3D development (I): memory optimization

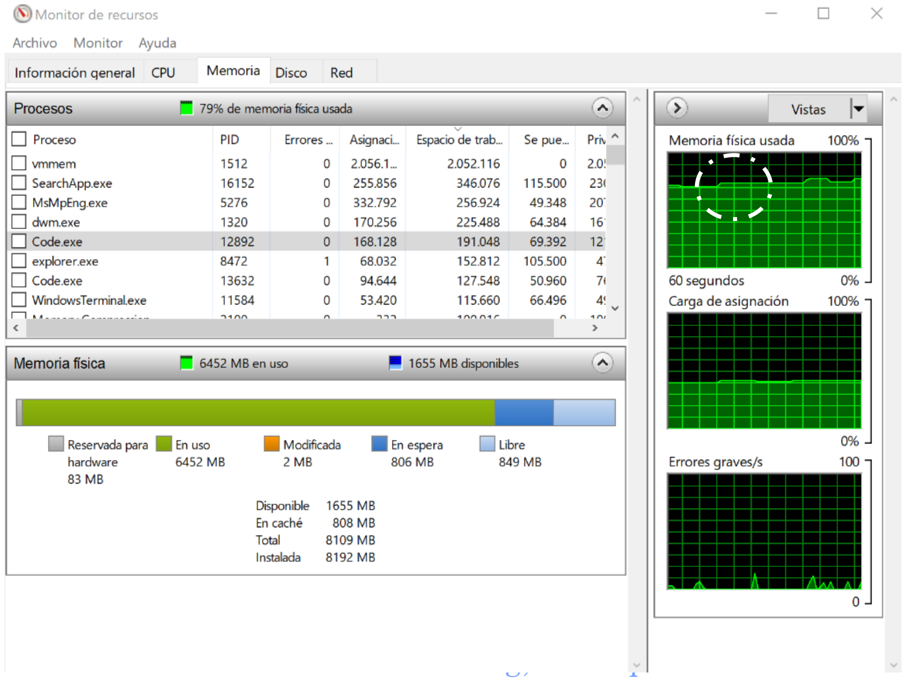



```python
def attrcleanup(self):

    # Fields
    del self.L, self.tL, self.iA, self.itA
    if hasattr(self, 'BC'):
        del self.BC
        del self.Dbc

    # Matrices
    del self.Px, self.Py, self.Pz
    del self.Ds, self.iDa, self.tDs, self.itDa
    del self.C
```

Deletes from memory the matrices that will not be used for the timestepping routine:

- Improves memory allocation by 60%
- Increases speed performance by 5%





SolverFIT3D development (II): 1D, 2D, 3D plotting

- class SolverFIT3D
  - func __init__
  - func one_step
  - func emsolve
  - func wakesolve
    - func beam
  - func apply_bc_to_C
  - func update_abc
  - func set_ghosts_to_0
  - func apply_conductors
  - func set_field_in_conductors_to_0
  - func apply_stl
  - func attrcleanup
  - func plot3D ←
  - func plot2D
  - func plot1D

```python
    def plot3D(self, field='E', component='z', clim=None, hide_solids=None,
               show_solids=None, add_stl=None, stl_opacity=0.1, stl_colors='white',
               title=None, cmap='jet', clip_volume=True, clip_normal='-y',
               clip_box=False, clip_bounds=None, off_screen=False, zoom=0.5,
               nan_opacity=1.0, n=None):
        '''
        Built-in 3D plotting using PyVista

        Parameters:
        ----------
        field: str, default 'E'
            3D field magnitude ('E', 'H', or 'J') to plot
            To plot a component 'Ex', 'Hy' is also accepted
        component: str, default 'z'
            3D field compoonent ('x', 'y', 'z', 'Abs') to plot. It will be overriden
            if a component is defined in field
```

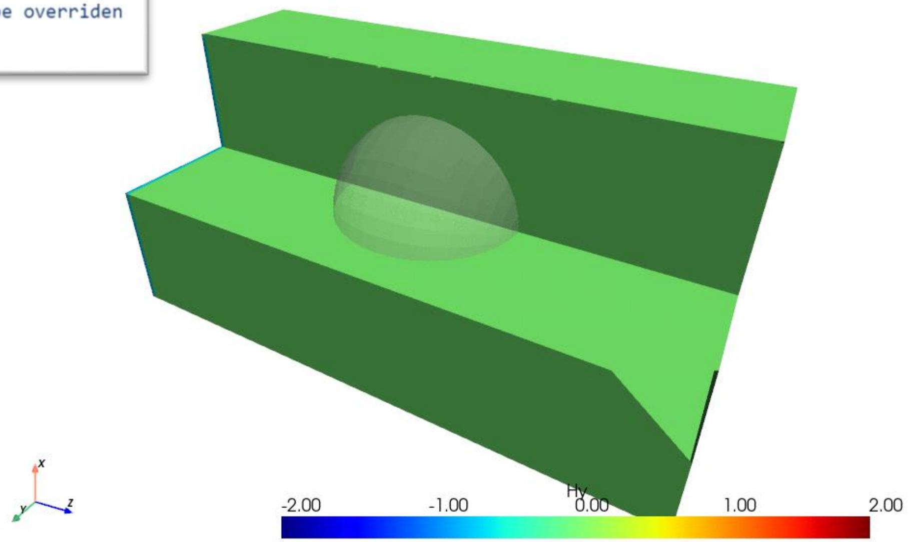
Using PyVista (vtk based) functions.

Plots can also be interactive:
- clip_volume or clip_normal flags when off_screen =True

Plot3D example:

examples/script_planewave_fit.py

A planewave interacting with a dielectric sphere *(University test)*



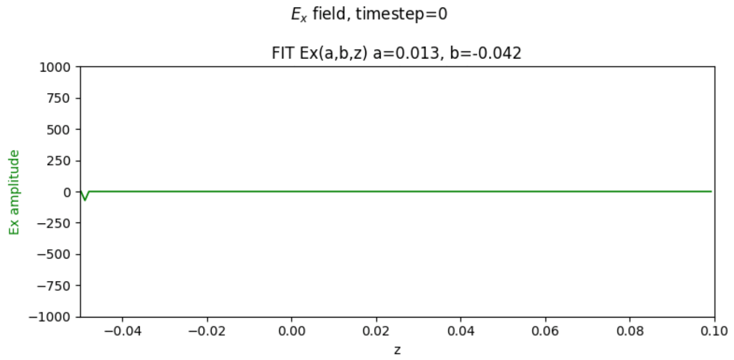
SolverFIT3D development (II): 1D, 2D, 3D plotting

- class SolverFIT3D
  - func __init__
  - func one_step
  - func emsolve
  - func wakesolve
    - func beam
  - func apply_bc_to_C
  - func update_abc
  - func set_ghosts_to_0
  - func apply_conductors
  - func set_field_in_conductors_to_0
  - func apply_stl
  - func attrcleanup
  - func plot3D
  - func plot2D
  - func plot1D

```python
    def plot2D(self, field='E', component='z', plane='ZY', pos=0.5, norm=None,
               vmin=None, vmax=None, figsize=[8,4], cmap='jet', patch_alpha=0.1,
               patch_reverse=False, add_patch=False, title=None, off_screen=False,
               n=None, interpolation='antialiased'):
        '''
        Built-in 2D plotting of a field slice using matplotlib
```

```python
    def plot1D(self, field='E', component='z', line=None, pos=0.5,
               xscale='linear', yscale='linear', xlim=None, ylim=None,
               figsize=[8,4], title=None, off_screen=False, n=None, **kwargs):
        '''
        Built-in 1D plotting of a field line using matplotlib
```

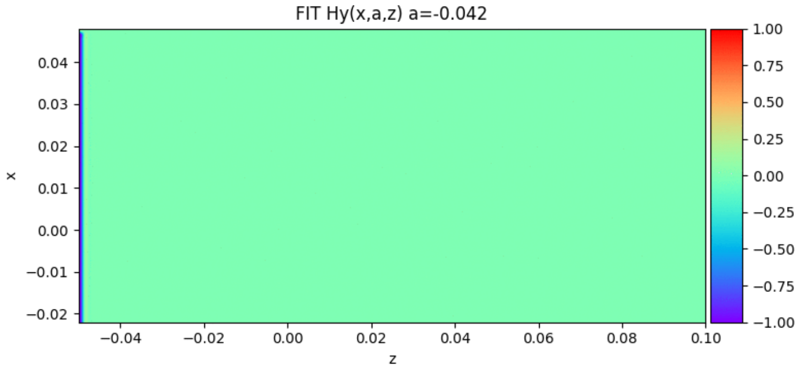
Matplotlib based coutourf (2D) and line plot (1D)



Plot2D and 1D example:

examples/script_planewave_fit.py

A planewave propagating through vacuum being repflected at the PEC boundary



SolverFIT3D development (IV): EM solve

- class SolverFIT3D
  - func __init__
  - func one_step
  - func emsolve
  - func wakesolve
    - func beam
  - func apply_bc_to_C
  - func update_abc
  - func set_ghosts_to_0
  - func apply_conductors
  - func set_field_in_conductors_to_0
  - func apply_stl
  - func attrcleanup
  - func plot3D
  - func plot2D
  - func plot1D

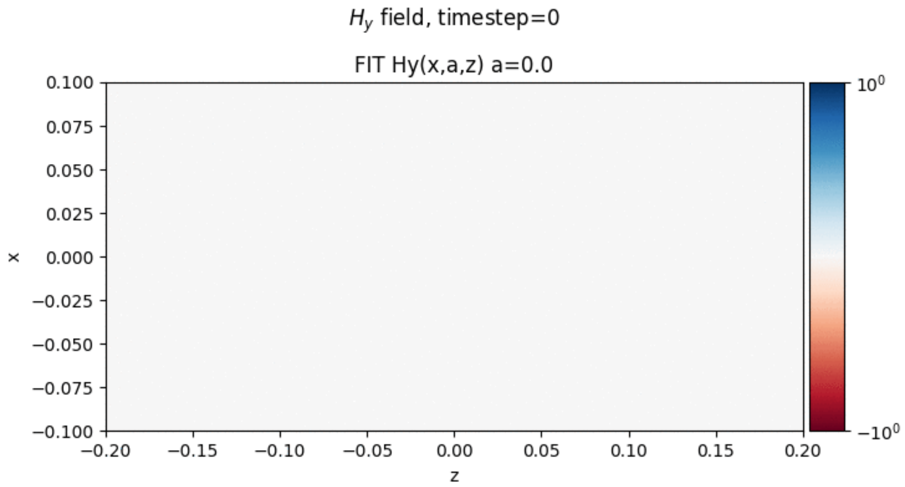
```python
def emsolve(self, Nt, source=None, save=False, fields=['E'], components=['Abs'],
            every=1, subdomain=None, plot=False, plot_every=1, **kwargs):
        '''
        Run the simulation and save the selected field components in HDF5 files
        for every timestep. Each field will be saved in a separate HDF5 file 'Xy.h5'
        where X is the field and y the component.

        Parameters:
        ----------
        Nt: int
            Number of timesteps to run
        source: func
            Function defining the time-dependednt source.
            It should be in the form `func(solver, t)`
```

Runs Electromagnetic time domain simulation given an initial condition or source

EM solve example:

examples/script_wavepacket_fit.py

A gaussian wavepacket propagating through vacuum domain *(University test)*



SolverFIT3D development (V): Wake solve

- class SolverFIT3D
  - func __init__
  - func one_step
  - func emsolve
  - func wakesolve
    - func beam
  - func apply_bc_to_C
  - func update_abc
  - func set_ghosts_to_0
  - func apply_conductors
  - func set_field_in_conductors_to_0
  - func apply_stl
  - func attrcleanup
  - func plot3D
  - func plot2D
  - func plot1D

```
    def wakesolve(self, wakelength, wake=None,
                  save_J=False, add_space=None,
                  plot=False, plot_every=1, **kwargs):
        '''
        Run the EM simulation and compute the longitudinal (z) and transverse (x,y)
        wake potential WP(s) and impedance Z(s).

        The `Ez` field is saved every timestep in a subdomain (xtest, ytest, z) around
        the beam trajectory in HDF5 format file `Ez.h5`.

        The computed results are available as Solver class attributes:
            - wake potential: WP (longitudinal), WPx, WPy (transverse) [V/pC]
            - impedance: Z (longitudinal), Zx, Zy (transverse) [Ohm]
            - beam charge distribution: lambdas (distance) [C/m] lambdaf (spectrum) [C]

        Parameters:
        ----------
        wakelength: float
            Desired length of the wake in [m] to be computed
```

- Runs Wakefield time domain simulation given a wakelength.
- Beam source injected is defined by WakeSolver object.
- Computes wake potential and impedance by saving Ez field every timestep and using the routines in WakeSolver class (needs h5py)

Wake solve example:

examples/script_cubcavity_fit.py

A gaussian beam traversing a PEC cubic pillbox cavity