EMWSD / Wakis

Electromagnetic and Wake Solver Development

Meeting #19

Elena de la Fuente, Carlo Zannini, Lorenzo Giacomel, Giovanni Iadarola

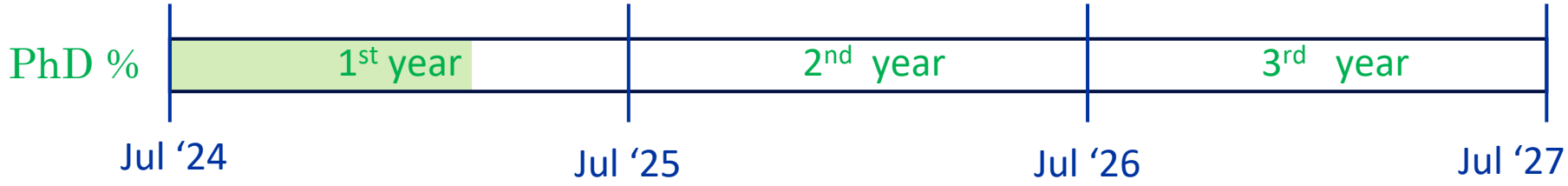
Outline

1. Where are we?
2. Main improvements in the code
3. Beam injection & Absorbing boundary conditions (ABC)
4. 1st Benchmark with CST: PEC pillbox
5. Conclusions & Next steps

Outline

1. Where are we?
2. Main improvements in the code
3. Beam injection & Absorbing boundary conditions (ABC)
4. 1st Benchmark with CST: PEC pillbox
5. Conclusions & Next steps

Wake solver milestones:



meetings #17, #18

- 0. Wake potential and impedance
- 1. FIT Maxwell Equations in 3D
- 2. PEC, Periodic, PMC boundaries
- 3. Embedded Boundaries: geometry import from **.stl** files

In progress

- 4. Beam injection $\mathbf{J_z}$
- 5. Materials: ε, μ, σ
- 6. Open boundaries: PML
- xx. Automatic testing

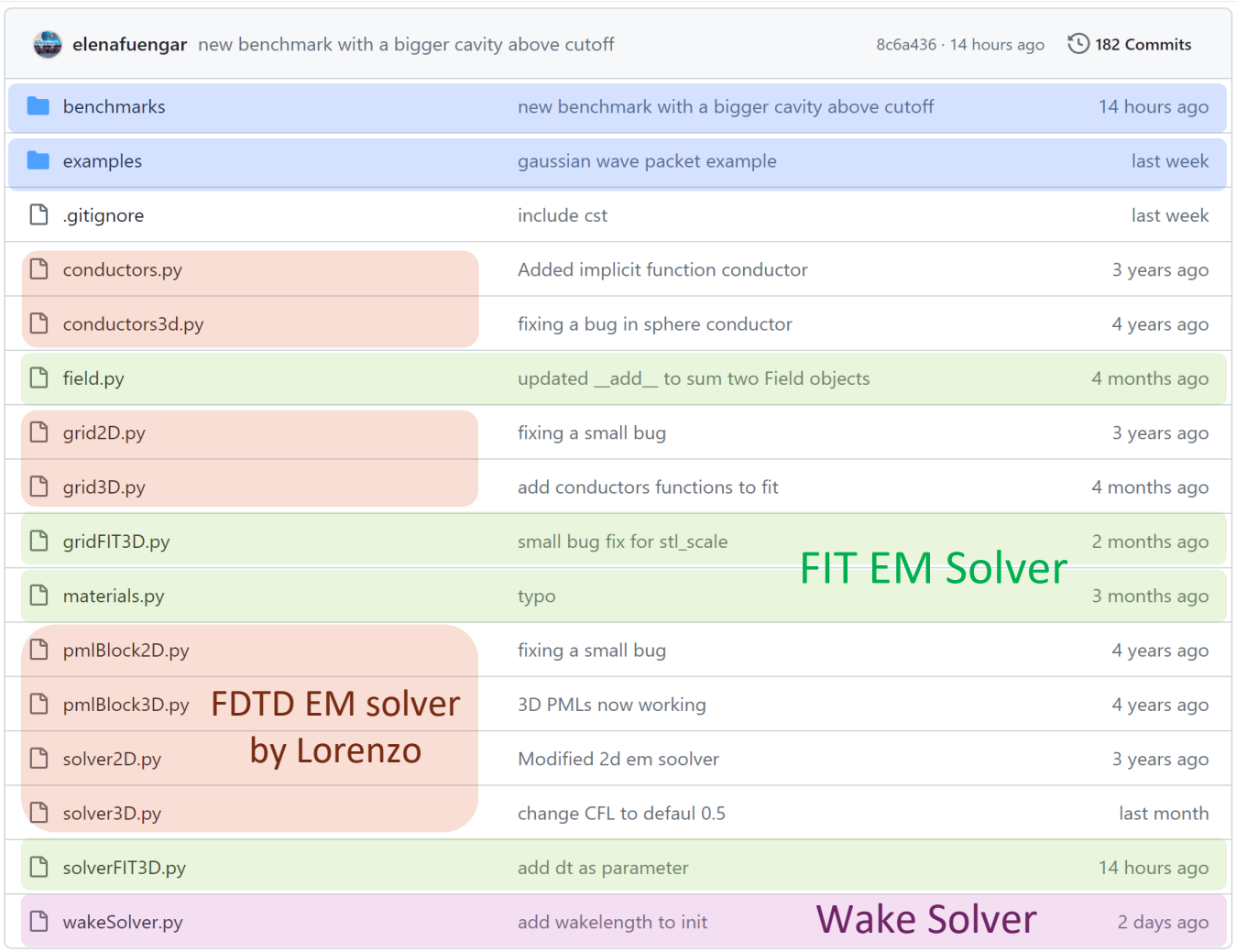
To Do

- 7. Frequency-dependent materials
- 8. …

Outline

1. Where are we?
2. Main improvements in the code
3. Beam injection & Absorbing boundary conditions (ABC)
4. 1st Benchmark with CST: PEC pillbox
5. Conclusions & Next steps

PyFIT GitHub overview



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 code is refactored into class WakeSolver

Should we turn this into a package already?

Merging wakis with PyFIT

Should PyFIT become the wakis package instead?


```
10  class WakeSolver():
11      ''' Class for wake potential and impedance
12      calculation from 3D time domain E fields
13      '''
14
15      def __init__(self, q=1e-9, sigmaz=1e-3, beta=1.0,
16                   xsource=0., ysource=0., xtest=0., ytest=0.,
17                   chargedist=None, ti=None, Ez_file='Ez.h5',
18                   save=True, results_folder='results/',
19                   verbose=0, logfile=True):
20          '''
21          Parameters
22          ----------
23          q : float
24              Beam total charge in [C]
25          sigmaz : float
26              Beam sigma in the longitudinal direction [m]
```

Moved all the relevant functions from wakis to the WakeSolver class

```
class SolverFIT3D:

    def __init__(self, grid, wake=None, cfln=0.5, dt=None,
                 bc_low=['Periodic', 'Periodic', 'Periodic'],
                 bc_high=['Periodic', 'Periodic', 'Periodic'],
                 use_conductors=False, use_stl=False,
                 bg=[1.0, 1.0]):
        '''
        TODO Docstring
        '''
```

WakeSolver instance is passed as a parameter to SolverFIT3D to perform Wakefield simulations

It's kept *optional* since it is not needed to perform just EM time domain simulations

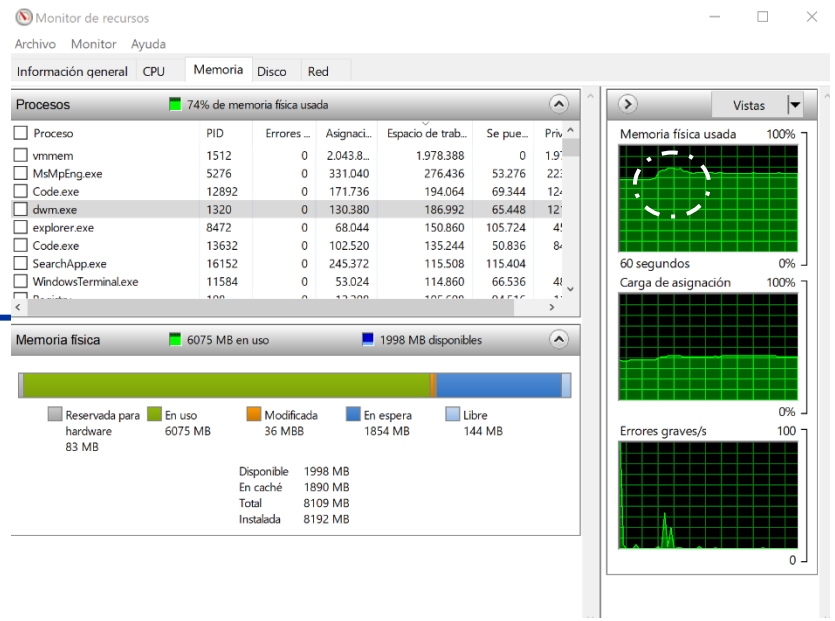
SolverFIT3D development (I): memory optimization

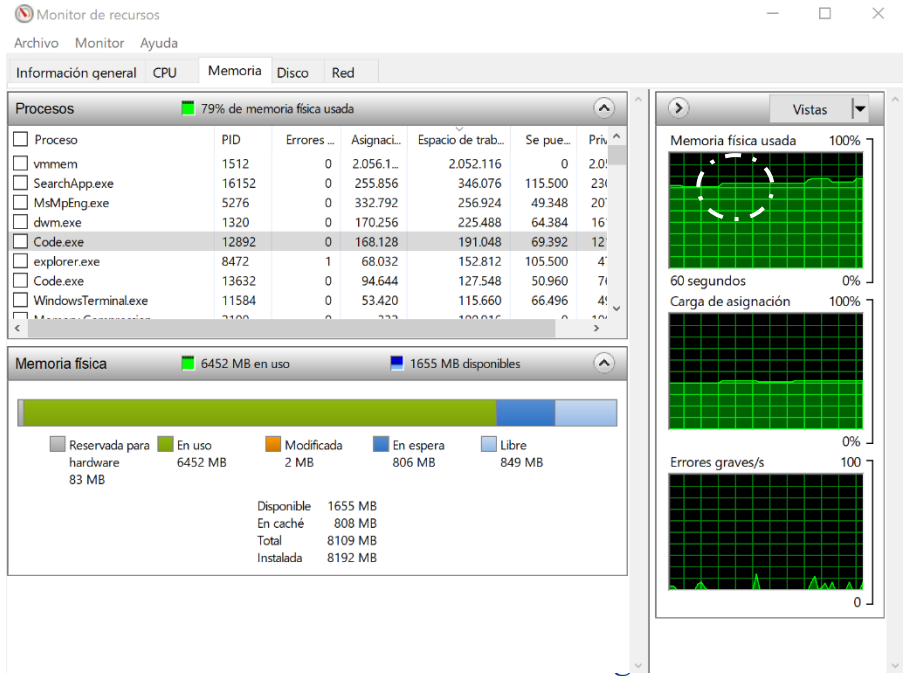
- class SolverFIT3D
  - func __init__
  - func one_step
  - func emsolve
  - func wakesolve
    - func beam
  - func apply_bc_to_C
  - func update_abc
  - func set_ghosts_to_0
  - func apply_conductors
  - func set_field_in_conductors_to_0
  - func apply_stl
  - func attrcleanup
  - func plot3D
  - func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

```
632     def attrcleanup(self):
633
634         # Fields
635         del self.L, self.tL, self.iA, self.itA
636         if hasattr(self, 'BC'):
637             del self.BC
638             del self.Dbc
639
640         # Matrices
641         del self.Px, self.Py, self.Pz
642         del self.Ds, self.iDa, self.tDs, self.itDa
643         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

```
645     def plot3D(self, field='E', component='z', clim=None, hide_solids=None,
646                show_solids=None, add_stl=None, stl_opacity=0.1, stl_colors='white',
647                title=None, cmap='jet', clip_volume=True, clip_normal='-y',
648                clip_box=False, clip_bounds=None, off_screen=False, zoom=0.5,
649                nan_opacity=1.0, n=None):
650         '''
651         Built-in 3D plotting using PyVista
652
653         Parameters:
654         ----------
655         field: str, default 'E'
656             3D field magnitude ('E', 'H', or 'J') to plot
657             To plot a component 'Ex', 'Hy' is also accepted
658         component: str, default 'z'
659             3D field compoonent ('x', 'y', 'z', 'Abs') to plot. It will be overriden
660             if a component is defined in field
```

Using PyVista (vtk based) functions.

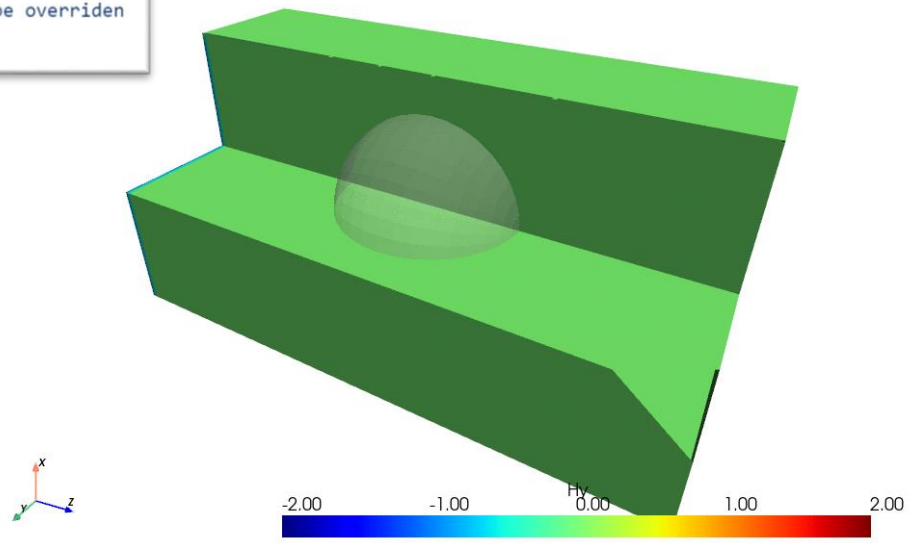
Plots can also be interactive:

- 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)*



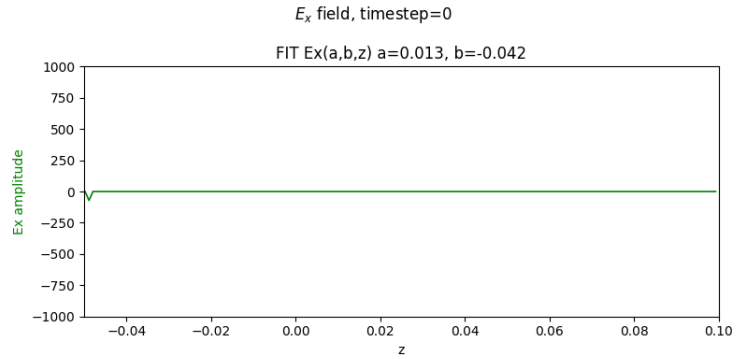
SolverFIT3D development (II): 1D, 2D, 3D plotting

- 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

```
821     def plot2D(self, field='E', component='z', plane='ZY', pos=0.5, norm=None,
822                vmin=None, vmax=None, figsize=[8,4], cmap='jet', patch_alpha=0.1,
823                patch_reverse=False, add_patch=False, title=None, off_screen=False,
824                n=None, interpolation='antialiased'):
825         '''
826         Built-in 2D plotting of a field slice using matplotlib
```

```
993     def plot1D(self, field='E', component='z', line=None, pos=0.5,
994                xscale='linear', yscale='linear', xlim=None, ylim=None,
995                figsize=[8,4], title=None, off_screen=False, n=None, **kwargs):
996         '''
997         Built-in 1D plotting of a field line using matplotlib
```

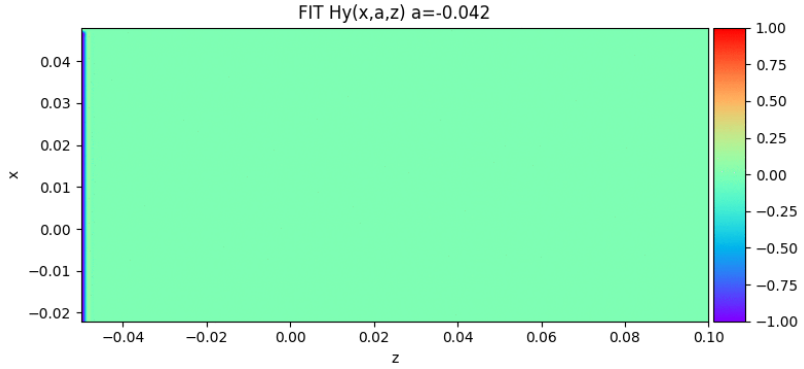
Matplotlib based coutourf (2D) and line plot (1D)



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

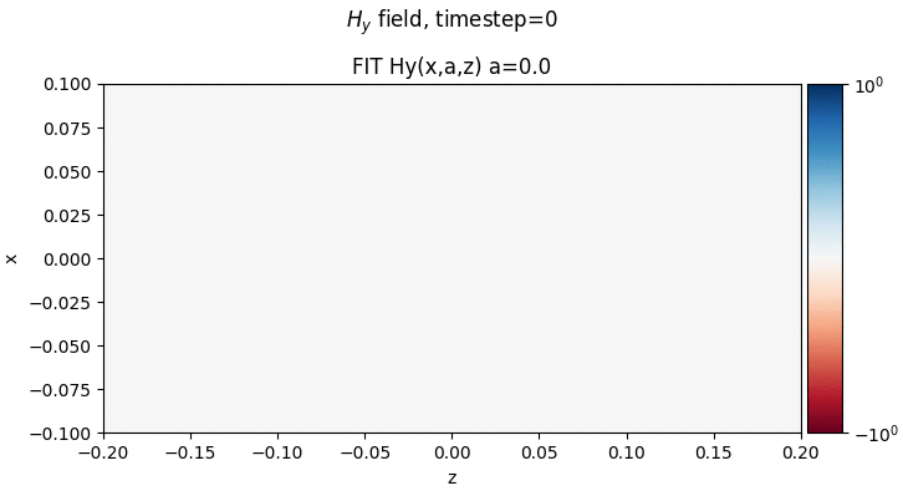
```
141     def emsolve(self, Nt, source=None, save=False, fields=['E'], components=['Abs'],
142                 every=1, subdomain=None, plot=False, plot_every=1, **kwargs):
143             '''
144             Run the simulation and save the selected field components in HDF5 files
145             for every timestep. Each field will be saved in a separate HDF5 file 'Xy.h5'
146             where X is the field and y the component.
147
148             Parameters:
149             ----------
150             Nt: int
151                 Number of timesteps to run
152             source: func
153                 Function defining the time-dependednt source.
154                 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

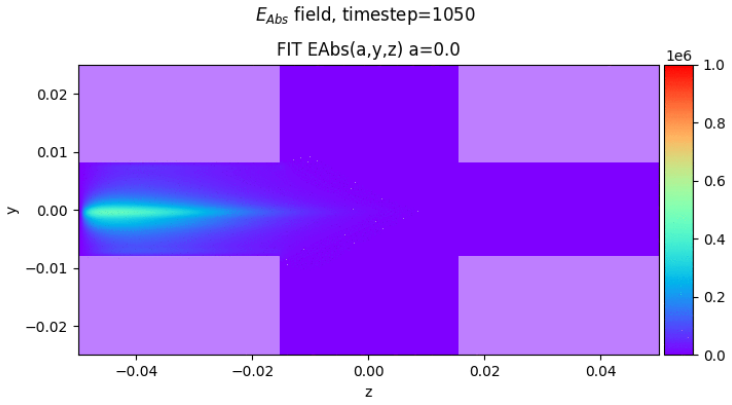
```
247     def wakesolve(self, wakelength, wake=None,
248                   save_J=False, add_space=None,
249                   plot=False, plot_every=1, **kwargs):
250         '''
251         Run the EM simulation and compute the longitudinal (z) and transverse (x,y)
252         wake potential WP(s) and impedance Z(s).
253
254         The `Ez` field is saved every timestep in a subdomain (xtest, ytest, z) around
255         the beam trajectory in HDF5 format file `Ez.h5`.
256
257         The computed results are available as Solver class attributes:
258             - wake potential: WP (longitudinal), WPx, WPy (transverse) [V/pC]
259             - impedance: Z (longitudinal), Zx, Zy (transverse) [Ohm]
260             - beam charge distribution: lambdas (distance) [C/m] lambdaf (spectrum) [C]
261
262         Parameters:
263         ----------
264         wakelength: float
265             Desired length of the wake in [m] to be computed
```

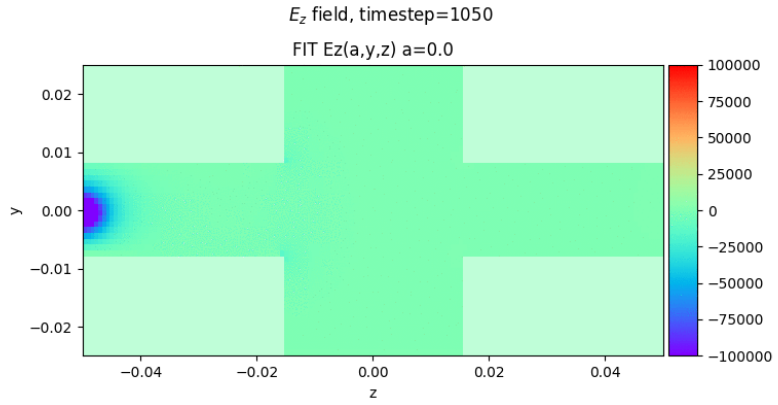
- Runs Wakefield time domain simulation given a wakelength.
- Beam source injected is defined by WakeSolver object.
- Computes wake potential and impedance by saving Ez field every timestep and using the routines in WakeSolver class (needs h5py)

Wake solve example:

examples/script_cubcavity_fit.py

A gaussian beam traversing a PEC cubic pillbox cavity





Outline

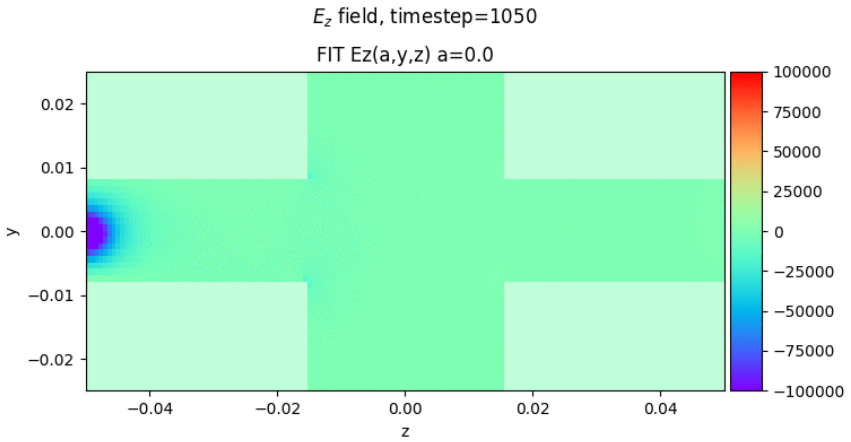
1. Where are we?
2. Main improvements in the code
3. Beam injection & Absorbing boundary conditions (ABC)
4. 1st Benchmark with CST: PEC pillbox
5. Conclusions & Next steps

Beam injection

The particle beam is injected as a linear current with a gaussian profile defined by the beam size σ_z and charge q

- Every timestep, the field J_z is updated at x_{source}, y_{source} for all the cells in z

```
330         def beam(self, t):
331             '''
332             Update the current J every timestep
333             to introduce a gaussian beam
334             moving in +z direction
335             '''
336             s0 = self.z.min() - c_light*self.ti
337             s = self.z - c_light*t
338
339             # gaussian
340             profile = 1/np.sqrt(2*np.pi*self.sigmaz**2)*np.exp(-(s-s0)**2/(2*self.sigmaz**2))
341
342             # update
343             self.J[self.ixs,self.iys,:,'z'] = self.q*c_light*profile/self.dx/self.dy
```



The beam injection produces a big E_z field perturbation at z- and z+, due to violation of the continuity equation (Gauss law)

$$\oiint_{\partial V} \boldsymbol{D} \cdot d\boldsymbol{A} = \iiint_{V} \boldsymbol{\rho}\, d\boldsymbol{V} \xrightarrow{\text{FIT}} \widetilde{\boldsymbol{S}}\widetilde{\boldsymbol{D}}_{\boldsymbol{A}} \left(\frac{\partial \boldsymbol{d}}{\partial t} + \boldsymbol{j} \right) = \boldsymbol{0}$$

Not included in update equations…
Should we correct it / enforce it?

Absorbing boundaries

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

```
120     def one_step(self):
121
122         if self.step_0:
123             self.set_ghosts_to_0()
124             self.step_0 = False
125
126             #if self.use_conductors:
127                 #self.set_field_in_conductors_to_0()
128
129         self.H.fromarray(self.H.toarray() -
130                          self.dt*self.tDsiDmuiDaC*self.E.toarray()
131                          )
132
133         self.E.fromarray(self.E.toarray() +
134                          self.dt*(self.itDaiDepsDstC * self.H.toarray() - self.iDeps*self.J.to
135                          )
136
137         #update ABC
138         if self.activate_abc:
139             self.update_abc()
```

It has to be updated every timestep

```
401     def apply_bc_to_C(self):
505
506         # Absorbing boundary conditions ABC
507         if any(True for x in self.bc_low if x.lower() == 'abc'):
508             self.activate_abc = True
509
510     def update_abc(self):
511         '''
512         Apply ABC algo to the selected BC,
513         to be applied after each timestep
514         '''
515
516         if self.bc_low[0].lower() == 'abc':
517             for d in ['x', 'y', 'z']:
518                 self.E[0, :, :, d] = self.E[1, :, :, d]
519                 self.H[0, :, :, d] = self.H[1, :, :, d]
```

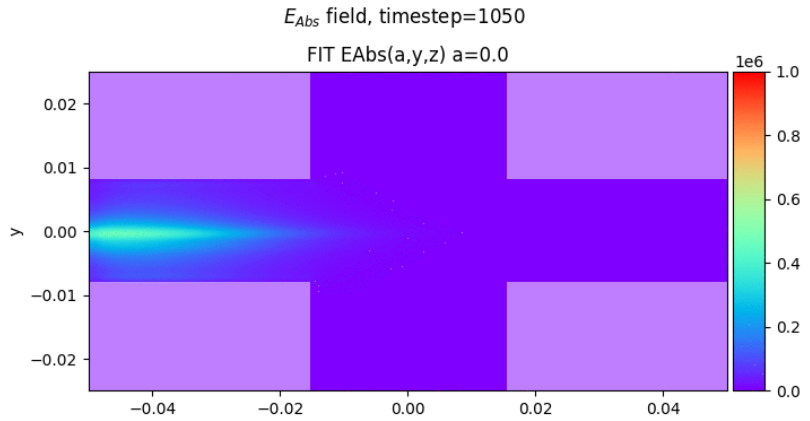
Same for all 6 boundaries (low and high, x, y, z)

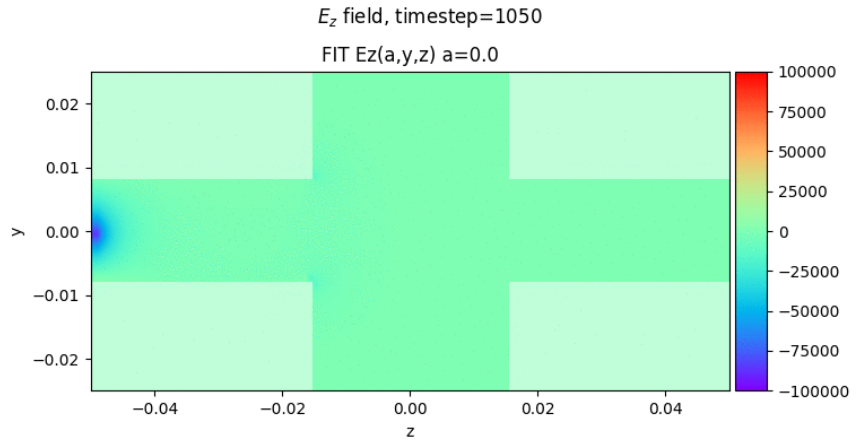
Absorbing boundaries (II)

The effect of this simple ABC is clearly visible at the edges of the domain:

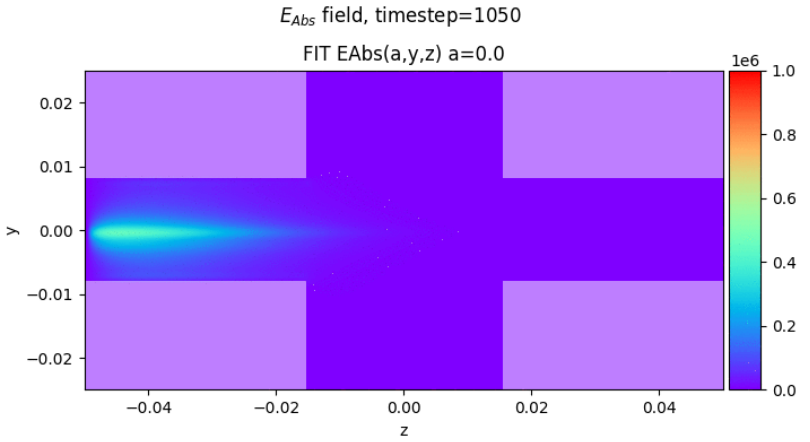
- ABC gives a smaller perturbation amplitude but it oscillates from negative to positive

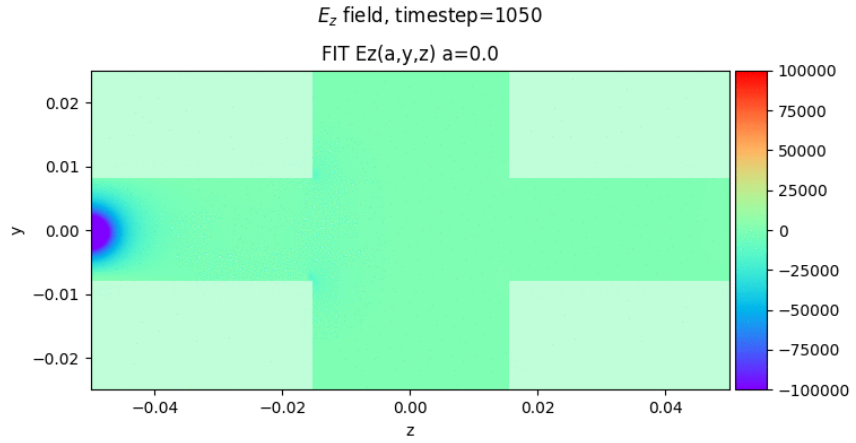
ABC





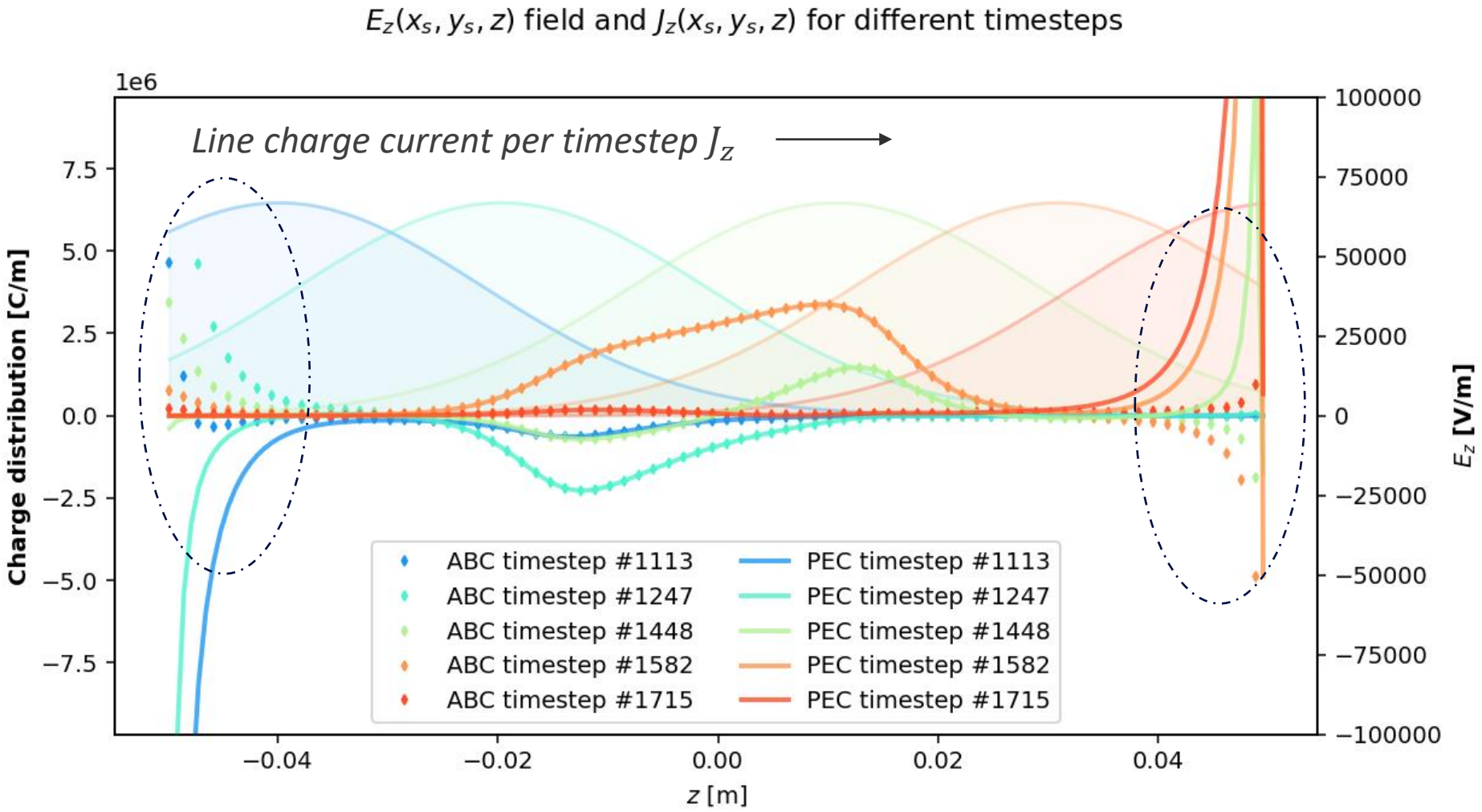
PEC





Absorbing boundaries (III)

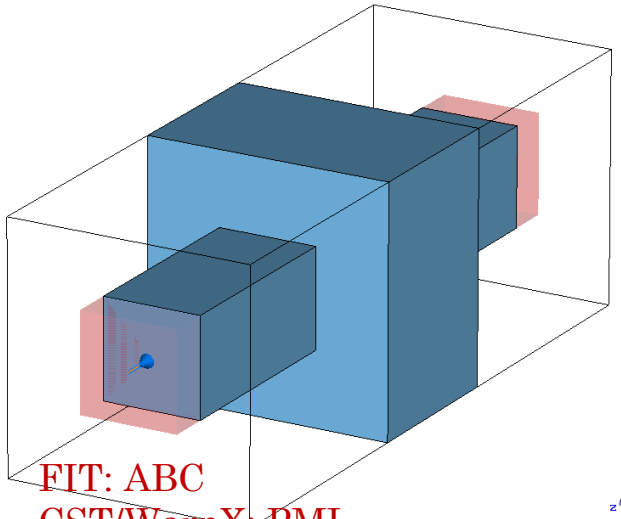
Order of magnitude reduction of the amplitude of the perturbation, specially at z+.



Outline

1. Where are we?
2. Main improvements in the code
3. Beam injection & Absorbing boundary conditions (ABC)
4. 1st Benchmark with CST: PEC pillbox
5. Conclusions & Next steps

Pillbox Cavity (bellow cutoff): FIT vs WarpX vs CST



FIT: ABC
CST/WarpX: PML

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

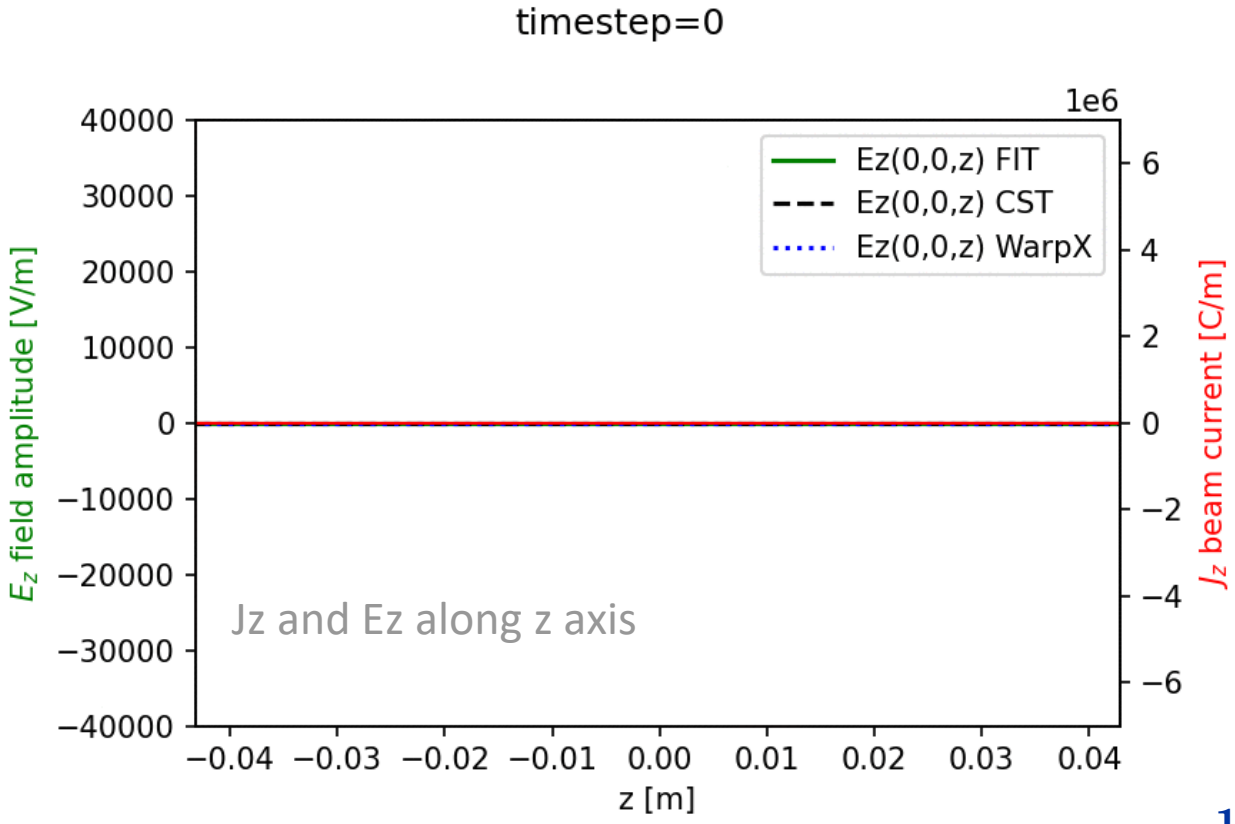
Cutoff frequency: ~10 GHz

Simulation time: for 1m Wakelength (8760 timesteps)

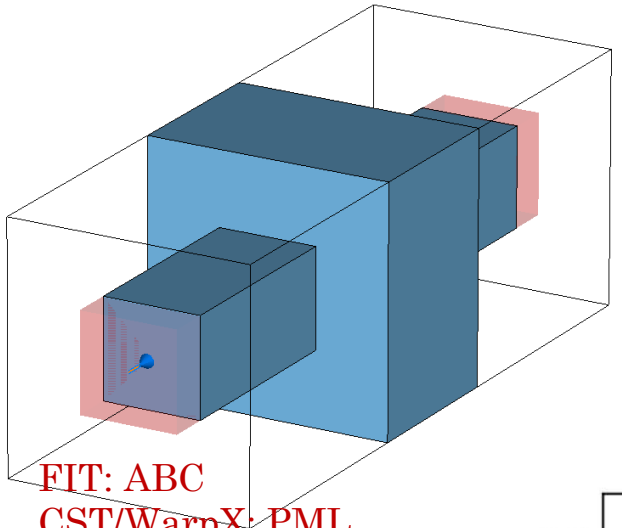
- CST: 42s, 16 threads, in `abpimp60g01`
- FIT: 3m40s , single core in `abpimp60g01`, 5m20s in lxplus
- WarpX: 42m to 1h 23m, single core in lxplus
  (high variability due to load of lxplus node…)

Remarks:

- Some error comes from the CST field export, since it does not respect the sampling when it's too close to the simulation timestep.
- **WarpX** reflections are higher on z-, lower on z+, but are present longer in the domain -> FIT performs slightly better.



Pillbox Cavity (bellow cutoff): FIT vs WarpX vs CST



FIT: ABC
CST/WarpX: PML

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

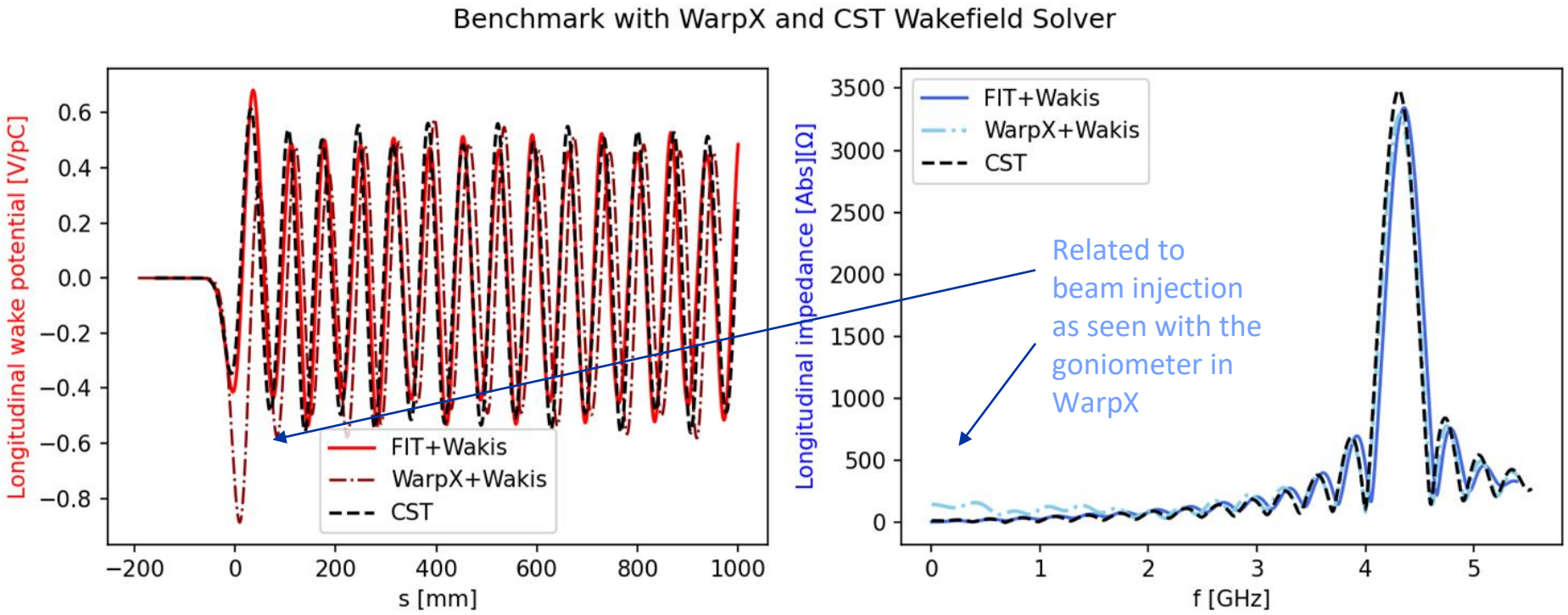
Beam:
σ_z = 18.5 mm (5 GHz)
q = 1e-9 C

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

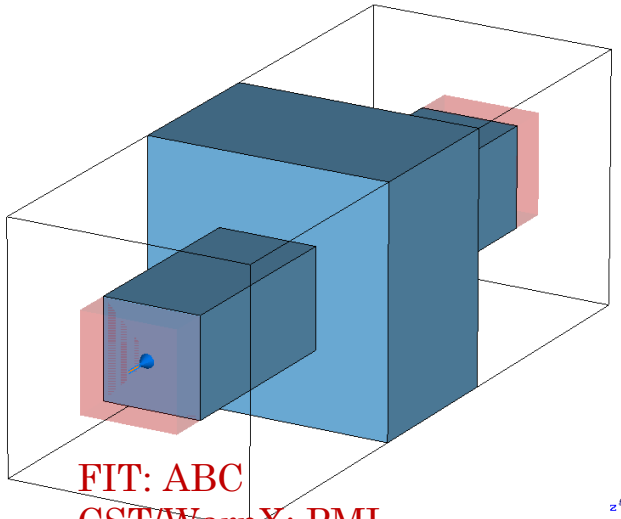
Cutoff frequency: ~10 GHz

*Calculation performed removing 10 cells at z+ and z- of the FIT domain to reduce the beam injection perturbation

Remarks: WarpX agreement worsens when not using the extended pipe model (pipe+50mm) see CEI-030823



Pillbox Cavity (above cutoff): FIT vs CST



FIT: ABC
CST/WarpX: PML

Geometry:
L_{cav} = 20 cm
h_{cav} = 30 cm
w_{cav} = 30 cm
L_{pipe} = 70 cm
h_{pipe} = 12 cm
w_{pipe} = 12 cm

Beam:
σ_z = 5 cm (2 GHz)
q = 1e-9 C

Mesh:
nx, ny, nz = 51, 51, 104
total: 270,504 cells
20 cells/wavelength

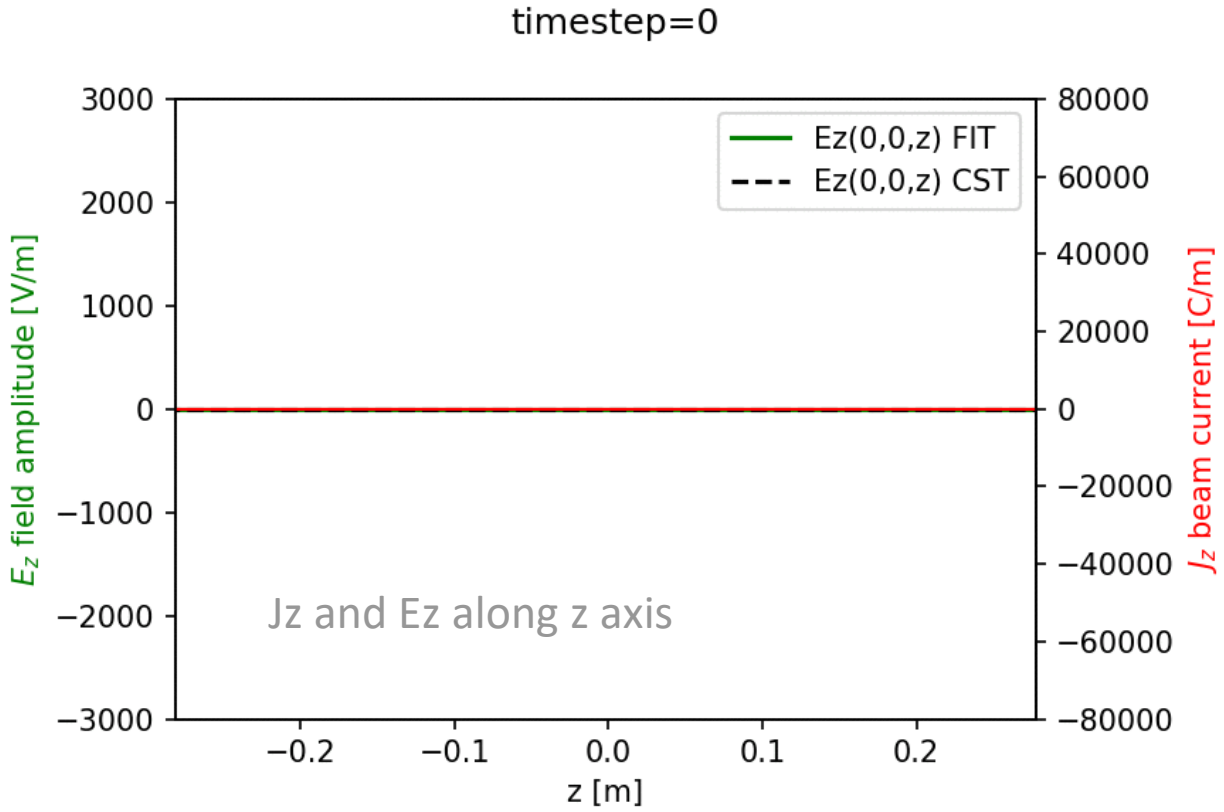
Cutoff frequency: ~1.25 GHz

Simulation time: for 10m Wakelength (10340 timesteps)

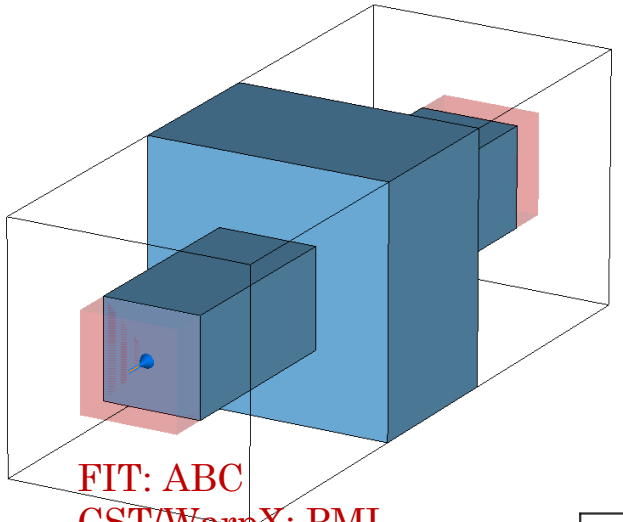
- CST: 52s, 16 threads, in `abpimp60g01`
- FIT: 4m50s , single core in `abpimp60g01`

Remarks:

- Main pattern is there, contaminated by the high relfections from the boundaries for the modes above cutoff



Pillbox Cavity (above cutoff): FIT vs CST



FIT: ABC
CST/WarpX: PML

Geometry:
L_{cav} = 20 cm
h_{cav} = 30 cm
w_{cav} = 30 cm
L_{pipe} = 70 cm
h_{pipe} = 12 cm
w_{pipe} = 12 cm

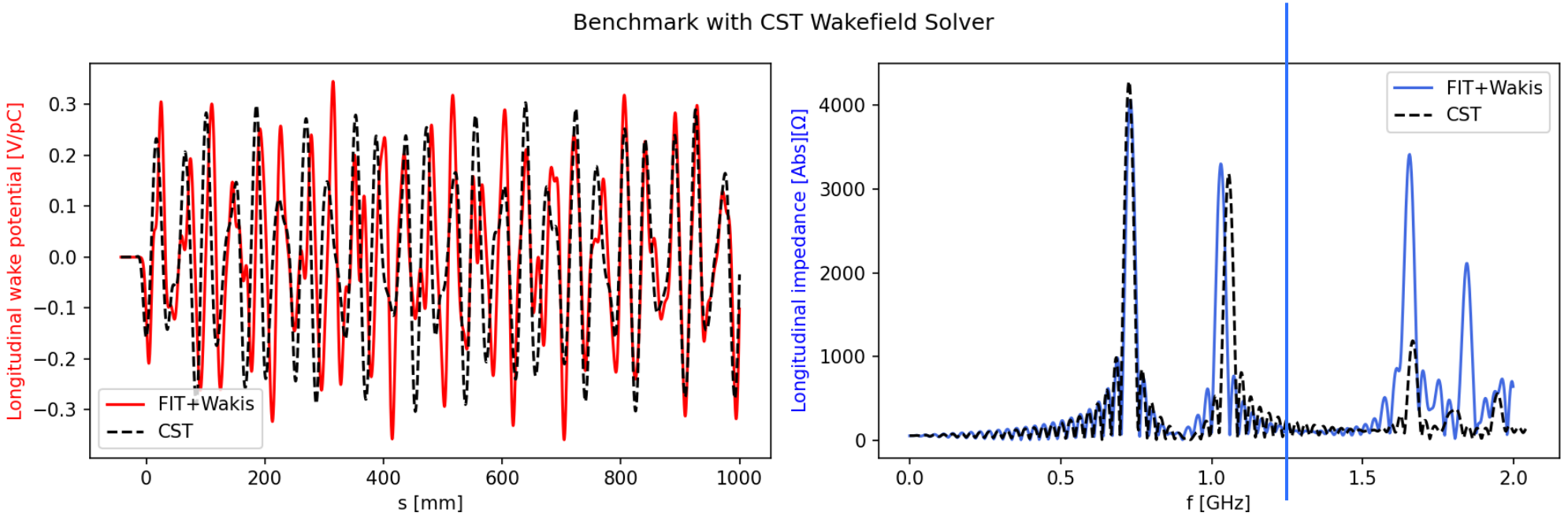
Beam:
σ_z = 5 cm (2 GHz)
q = 1e-9 C

Mesh:
nx, ny, nz = 51, 51, 104
total: 270,504 cells
20 cells/wavelength

*Calculation performed removing 10 cells at z+ and z- of the FIT domain to reduce the beam injection perturbation

Remarks: the agreement worsens the closer to cutoff frequency. The modes above cuttof present an artificial amplitude (not propagating) –> Need for PML

Cutoff frequency: ~1.25 GHz



Outline

1. Where are we?
2. Main improvements in the code
3. Beam injection & Absorbing boundary conditions (ABC)
4. 1st Benchmark with CST: PEC pillbox
5. Conclusions & Next steps

Conclusions

✓ Progress on the code:

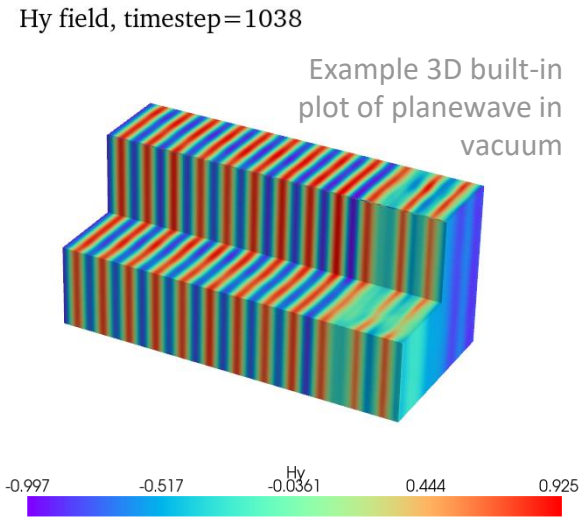
- Built-in plotting 1D, 2D, (matlab based) and 3D (pyvista/vtk based): Fast, flexible (**kwargs), proven not memory consuming, possibility of offscreen plotting to create animations.
- Attribute cleanup: reduces memory consumption 60% and slightly improves performance
- Solving routines:
  - emsolve() for pure **Electromagnetic time domain**, source can be any user function: func(solver, t).
  - wakesolve() for **Wakefield time domain**, source is a particle beam, computes wake potential and impedance W, Z.

✓ Beam injection and Absorbing boundaries:

- Beam injection as in CST using line current: barely affects performance >0.1%. Produces perturbation at the boundaries due to breaking continuity equation + reflections
- Absorbing boundaries ABC FOEXTRAP implemented. Small impact on computation time <1%. Helps reducing perturbations, specially at boundary z+. Not comparable to PML.

✓ 1st Benchmark vs CST and WarpX:

- Simulated 2 cubic pillbox cavities: below and above cutoff.
  - Agreement satisfactory below cutoff, while above cutoff the need of PML becomes relevant to get the right amplitude and frequency of the modes.
- Performance vs WarpX: FIT is 88% faster*, and ABC gives smaller reflections** than WarpX's PML.



→ Safe to say we are in a better position with FITWakis Feb 2024 compared to July 2023 WarpX+Wakis ? ☺

*comparison on lxplus (same node). Workload variability may impact WarpX's time
**probably due to beam injection with 10^6 macroparticles perturbations not being absorbed by PML

Next steps / Questions

i. GitHub strategy: moving to a package? what to do with wakis? (I would like to keep the name ☺)

ii. Beam injection perturbation correction: enforce continuity equation?

iii. Working on implementing conductivity M_σ:
   - It can be fairly easy: Just add update equation for the current J -> $\boldsymbol{j^{n+1} = M_\sigma e^{n+0.5}}$
   - Or it can be quite convoluted: Weiland formulation & Berenguer formulation (backup)
   - To be benchmarked with a lossy pillbox ?

iv. Completing tests for university: probably trip around end of April
   - Using the planewave and gaussian wave packet tests, we can perform tests in symmetry of the solver, dispersion, speed of light conservation, refraction angle when interacting with dielectric, energy conservation.. Etc. The idea is to use pytest

v. Try to make the code faster with cython?

vi. CERN School of Computing?

Thank you ☺ !!!

Electromagnetic and Wake Solver Development
meeting #19

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

Berenguer PML vs Weiland update equations

$$E_y^{n+1}(i, j+1/2) = e^{-\sigma_x(i)\,\Delta t/\varepsilon_0} E_y^n(i, j+1/2) - \frac{(1 - e^{-\sigma_x(i)\,\Delta t/\varepsilon_0})}{\sigma_x(i)\,\Delta x} \times [H_{zx}^{n+1/2}(i+1/2, j+1/2) + H_{zy}^{n+1/2}(i+1/2, j+1/2) - H_{zx}^{n+1/2}(i-1/2, j+1/2) - H_{zy}^{n+1/2}(i-1/2, j+1/2)] \quad (37)$$

$$H_{zx}^{n+1/2}(i+1/2, j+1/2) = e^{-\sigma_x^*(i+1/2)\,\Delta t/\mu_0} H_{zx}^{n-1/2}(i+1/2, j+1/2) - \frac{(1 - e^{-\sigma_x^*(i+1/2)\,\Delta t/\mu_0})}{\sigma_x^*(i+1/2)\,\Delta x} \times [E_y^n(i+1, j+1/2) - E_y^n(i, j+1/2)], \quad (38)$$

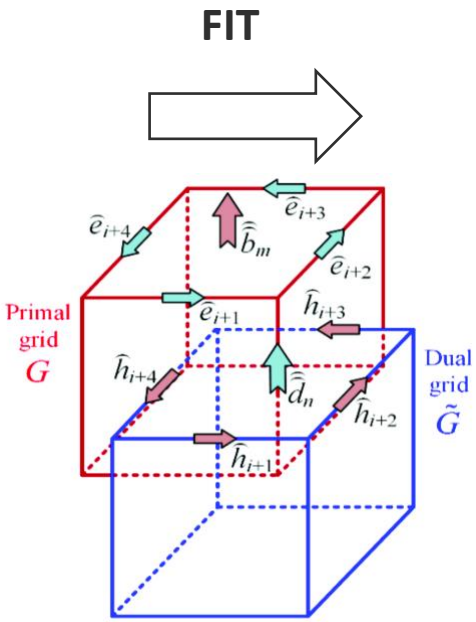
$$e^{n+1.5} = exp\left(-\widetilde{D}_{\varepsilon}^{-1}\widetilde{D}_{\kappa}\Delta t\right)e^{n+0.5} + (1 - exp(-\widetilde{D}_{\varepsilon}^{-1}\widetilde{D}_{\kappa}\Delta t)\,\widetilde{D}_{\kappa}^{-1}\,D_A^{-1}CD_sD_{\mu}^{-1}b^{n+1} - (1 - exp(-\widetilde{D}_{\varepsilon}^{-1}\widetilde{D}_{\kappa}\Delta t)\,\widetilde{D}_{\kappa}^{-1}j^{n+1}$$

$$h^{n+1} = h^n - \Delta t\,\widetilde{D}_sD_{\mu}^{-1}D_A^{-1}Ce^{n+0.5}$$

FIT theory: Grid Maxwell Equations

$$
\begin{cases}
\oint_{\partial A} \boldsymbol{E} \cdot d\boldsymbol{s} = -\iint_{A} \dfrac{\partial \boldsymbol{B}}{\partial t} \cdot d\boldsymbol{A} \\
\oint_{\partial A} \boldsymbol{H} \cdot d\boldsymbol{s} = -\iint_{A} \left(\dfrac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J} \right) \cdot d\boldsymbol{A} \\
\oiint_{\partial V} \boldsymbol{B} \cdot d\boldsymbol{A} = 0 \\
\oiint_{\partial V} \boldsymbol{D} \cdot d\boldsymbol{A} = \iiint_{V} \boldsymbol{\rho}\, dV \\
\boldsymbol{D} = \underline{\underline{\varepsilon}} \boldsymbol{E}, \quad \boldsymbol{B} = \underline{\underline{\mu}} \boldsymbol{H}, \quad \mathbf{J} = \underline{\underline{\sigma}} \boldsymbol{E} + \boldsymbol{\rho v}
\end{cases}
$$

FIT



Grid Maxwell Equations

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

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

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

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

- Operators
- Spatial matrixes
- Material properties

$$\boldsymbol{d} = \boldsymbol{\tilde{D}_\varepsilon e}, \quad \mathbf{b} = \boldsymbol{D_\mu h}, \quad \mathbf{j} = \boldsymbol{\tilde{D}_\sigma e} + \boldsymbol{D_\rho v}$$

$$\boldsymbol{\varepsilon} = (\boldsymbol{\varepsilon_r} + \frac{\boldsymbol{\sigma}}{\boldsymbol{j\omega}})\boldsymbol{\varepsilon_0}$$

What we care about:
Update equations

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

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

We need to build all these matrices and then apply these equations every timestep !