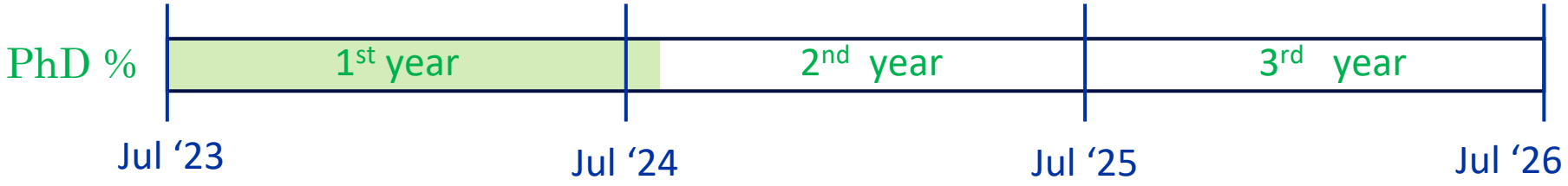
# EMWSD / Wakis

## Electromagnetic and Wake Solver Development

Meeting #20

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

# Wake solver milestones:




meetings #17, #18, #19, CEI

- 0. Wake potential and impedance
- 1. FIT Maxwell Equations in 3D
- 2. PEC, Periodic, PMC boundaries
- 3. Embedded Boundaries: geometry import from **.stl** files
- 4. Beam injection $\mathbf{J_z}$
- 5. Materials: $\varepsilon, \mu, \sigma$

In progress

- x. Documentation
- 6. on GPU & memory opt.
- 7. Low beta & SC
- 8. Open boundaries & PML
- 9. Numerical tests vs analytic

To Do

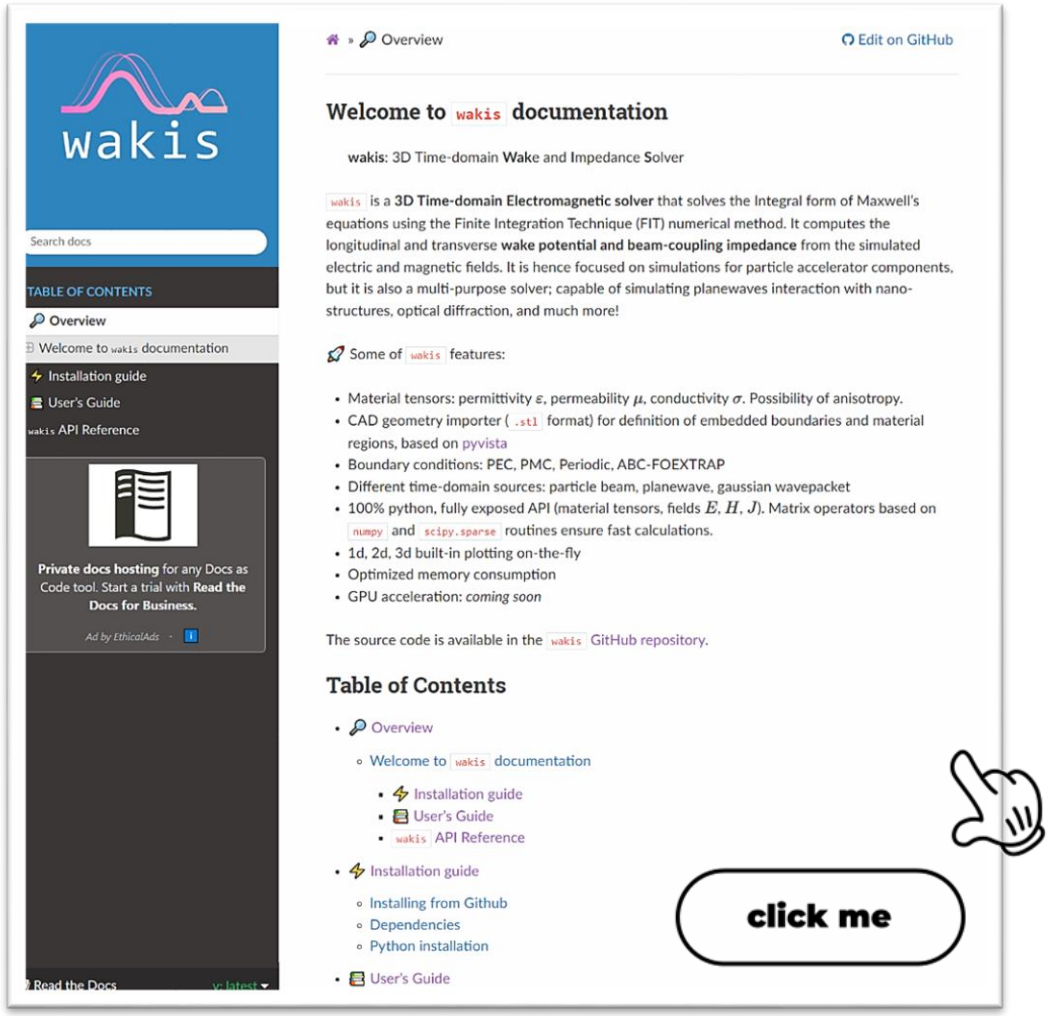
- 10. Leontovich condition for good conductors >100 S/m
- 11. Staircased geometry: PBA/grid refinement
- 12. Moving window solver (?)
- 13. Frequency-dependent materials
- 14. …

# Documentation

So far, positive feedback from first users:

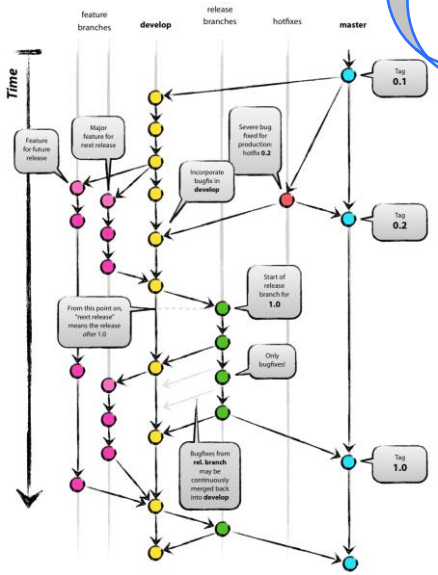
- Elena M.
- Josephine P.
- Sebastien J. (BESY)
- PyVista people via Slack ☺

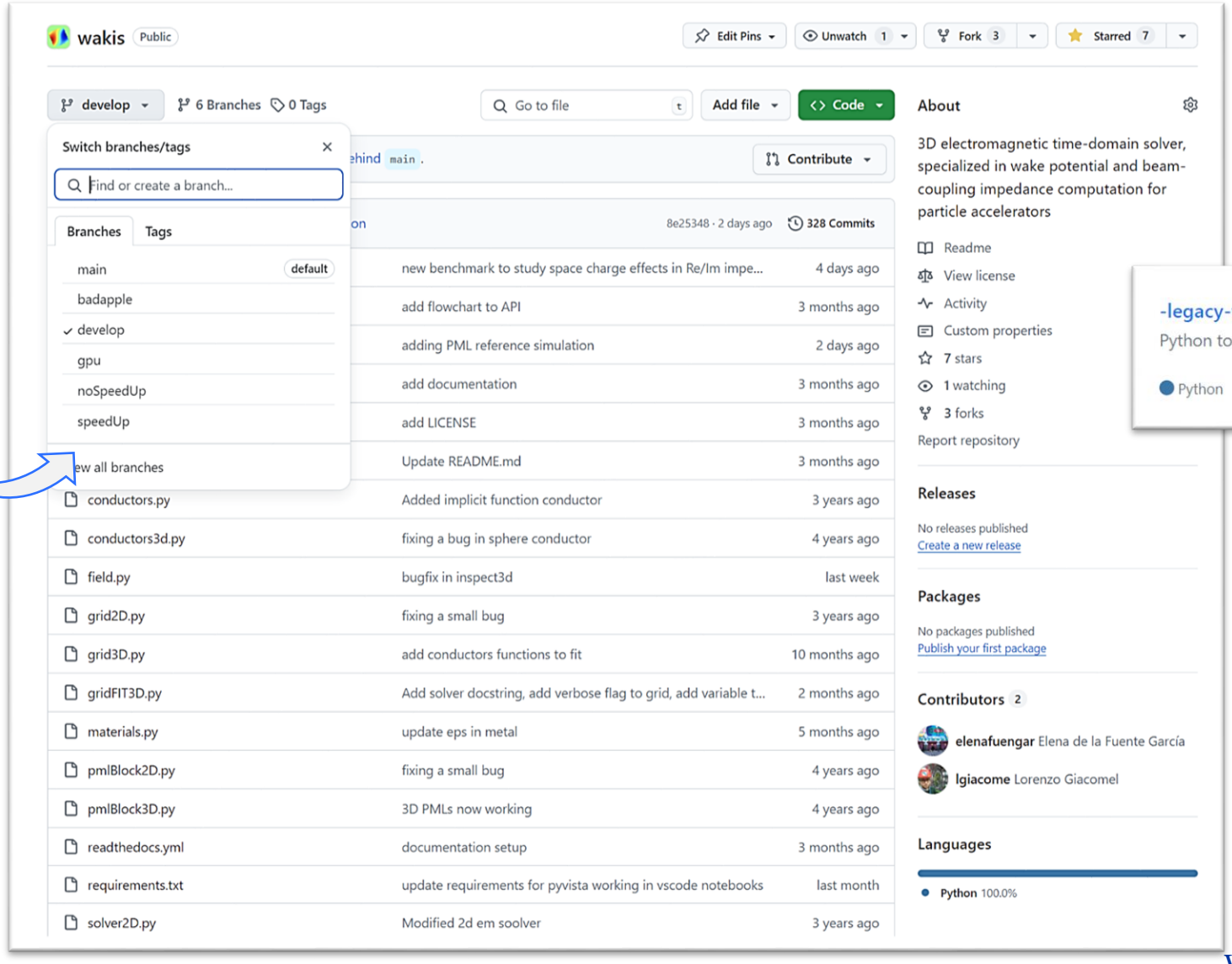
Feedback helped solve installation problems for interactive plots in remote machines. Installation stable for all type of machines in `requirements.txt`



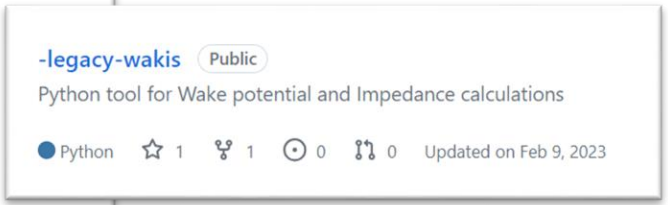
# GitHub strategy

- Stable branch `**main**` for user's to fork
- `**develop**` branch with the latest features
- Individual branches for features
- **cherry-pick** to main for bugfixes




Old `wakis` postprocessor was renamed `–legacy-wakis`



# GPU implementation

**User side:** Enable `use_gpu=True` in Solver instantiation

**Code side:**

wakis / field.py

elenafuengar bugfix in inspect3d

Code Blame 430 lines (348 loc) · 14.9 KB

```python
import numpy as xp

try:
    import cupy as xp_gpu
    imported_cupy = True
except ImportError:
    imported_cupy = False

class Field:
    '''
    Class to switch from 3D to collapsed notation by
    defining the __getitem__ magic method

    linear numbering:
    n = 1 + (i-1) + (j-1)*Nx + (k-1)*Nx*Ny
    len(n) = Nx*Ny*Nz
    '''
    def __init__(self, Nx, Ny, Nz, dtype=float,
                 use_ones=False, use_gpu=False):
```

```python
if self.on_gpu:
    return self.array[key].get()
else:
    return self.array[key]
```

wakis / solverFIT3D.py

elenafuengar small bugfixes in emsolve (close h5) and wakesolve (add_space=0)

Code Blame 1434 lines (1191 loc) · 56.1 KB

```python
from tqdm import tqdm

import numpy as np
import time

from scipy.constants import c as c_light, epsilon_0 as eps_0, mu_0 as mu_0
from scipy.sparse import csc_matrix as sparse_mat
from scipy.sparse import diags, hstack, vstack

from field import Field
from materials import material_lib

try:
    from cupyx.scipy.sparse import csc_matrix as gpu_sparse_mat
    imported_cupyx = True
except ImportError:
    imported_cupyx = False

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, use_gpu=False,
                 bg=[1.0, 1.0], verbose=1):
```

```python
# Move to GPU
if use_gpu:
    if verbose: print('Moving to GPU...')
    if imported_cupyx:
        self.tDsiDmuiDaC = gpu_sparse_mat(self.tDsiDmuiDaC)
        self.itDaiDepsDstC = gpu_sparse_mat(self.itDaiDepsDstC)
        self.iDeps = gpu_sparse_mat(self.iDeps)
        self.Dsigma = gpu_sparse_mat(self.Dsigma)
    else:
        print('*** cupyx could not be imported, please check CUDA installation')

if verbose:  print(f'Total initialization time: {time.time() - t0} s')
```

Check Here

GitHub

# GPU implementation (II)

Tested it in **pcbe-abp-gpu001** (thanks Gianni ☺):

- Pushing the limits, we can go up to 20,000,000 cells, takes 20 minutes
- New bottleneck: **memory!**
- Scales linearly with mesh (e.g., 5M cells: 3,5 kMB → 20M cells: 11.5k MB)

```
In [1]: run condcubcavitymm.py
Starting simulation with N_cells = 20000000
Generating grid...
Importing stl solids...
Assembling operator matrices...
Appliying boundary conditions...
Adding material tensors...
Pre-computing ...
Moving to GPU...
Total initialization time: 73.22341299057007 s
Running electromagnetic time-domain simulation...
  2%|                                                    | 412/24470 [00:19<18:30, 21.66it/s]
```

```
Every 2.0s: ps -axo cpuid,pcpu,pmem,user,pid,tim...  pcbe-abp-gpu001: Fri Aug  9 13:32:43 2024

CPUID %CPU %MEM USER         PID     TIME COMMAND
   38 1469  0.5 afornara  877968 07:18:32 python _new_wake_script.py --yaml_path /afs/cern.ch/
   16 1350  0.5 afornara  877965 06:43:08 python _new_wake_script.py --yaml_path /afs/cern.ch/
   46 1339  0.5 afornara  877962 06:39:43 python _new_wake_script.py --yaml_path /afs/cern.ch/
    8  101  0.7 afornara  867375 04:10:37 /afs/cern.ch/work/a/afornara/public/SPS_studies/from
    0 86.7  1.0 angilard  659638 6-16:21:09 python main_fft_study_hparamsCV.py
   32 83.0  9.1 edelafue  876075 00:52:23 /home/edelafue/miniconda3/bin/python /home/edelafue/
   34 50.3  2.2 elamb     878933 00:02:13 /home/elamb/.vscode-server/cli/servers/Stable-b1c0a1
   34 47.8  1.2 elamb     878987 00:01:05 /home/elamb/.vscode-server/cli/servers/Stable-b1c0a1
   32 13.6  4.0 elamb     871064 00:25:34 /home/elamb/clean/executable/2023_apr/miniconda/bin/
   30  9.6  1.8 afornara  866735 00:27:21 /home/afornara/.vscode-server/cli/servers/Stable-f1e
   11  5.0  0.0 root        3907 1-06:18:42 [nv_queue]
   11  3.6  0.0 root        3056 21:39:16 [nv_queue]
```

```
(base) edelafue@pcbe-abp-gpu001:~$ gpustat --show-user
pcbe-abp-gpu001      Fri Aug  9 12:24:29 2024  470.256.02
[0] NVIDIA TITAN V   | 32'C,   0 % |     9 / 12066 MB | gdm(4M)
[1] NVIDIA TITAN V   | 36'C,  99 % | 11472 / 12066 MB | edelafue(11455M) gdm(4M)
[2] NVIDIA TITAN V   | 32'C,   0 % |   622 / 12066 MB | afornara(613M) gdm(4M)
[3] NVIDIA TITAN V   | 32'C,   0 % |    77 / 12065 MB | gdm(56M) gdm(15M)
(base) edelafue@pcbe-abp-gpu001:~$
```

Comparison with CPU speed in limit case 20M:

```
In [1]: run condcubcavitymm.py
Starting simulation with N_cells = 20000000
Generating grid...
Importing stl solids...
Assembling operator matrices...
Appliying boundary conditions...
Adding material tensors...
Pre-computing ...
Total initialization time: 38.916051149368286 s
Running electromagnetic time-domain simulation...
  6%|                                                    | 1357/24470 [57:11<39:08:15,  6.10s/it]
```

Only 9% of RAM used, but takes 40 h!!!!! To complete

# Memory opt. and speedup

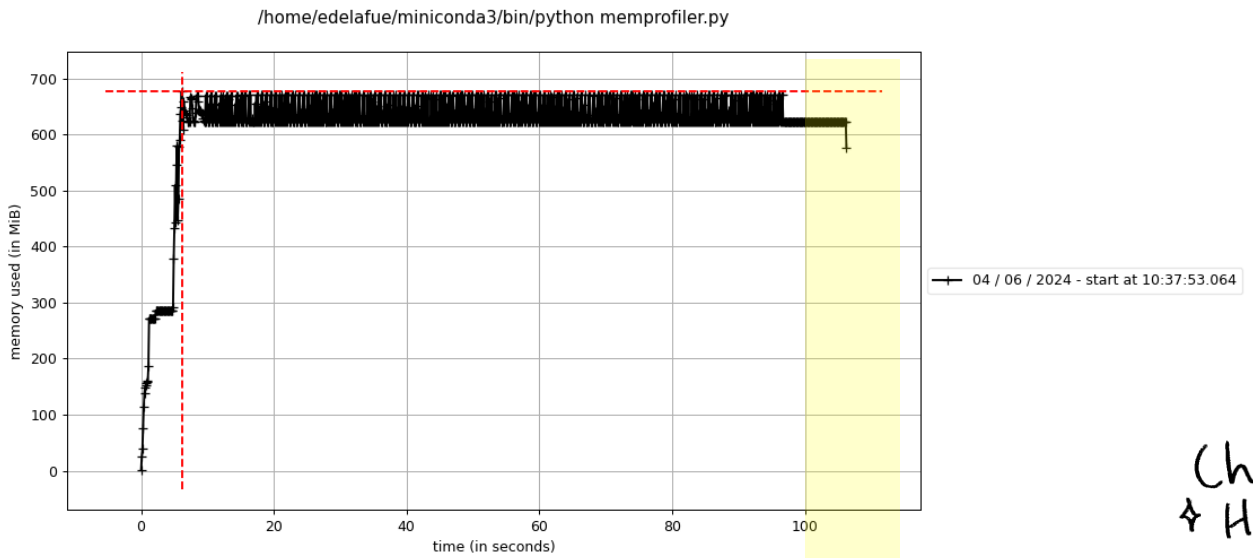
Run through the **memory profiler** for a test simulation of 1M cells (~700 MB).

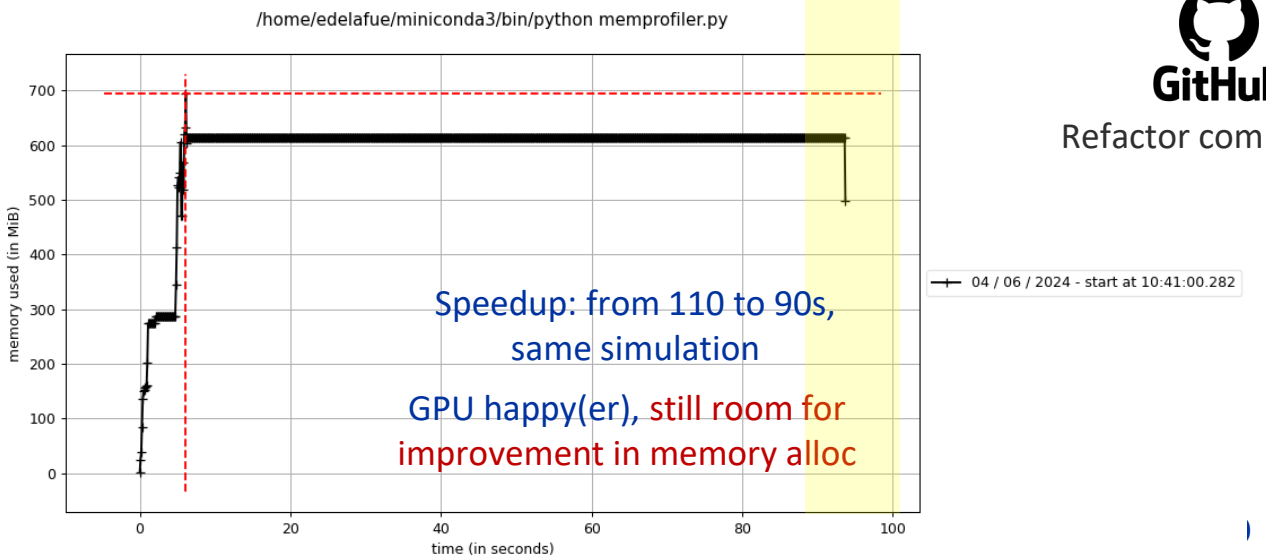
Storing fields E, H, J and tensors $\sigma$, $\varepsilon$, $\mu$ in numpy (x,y,z)x3d matrices inside `Field class` caused spikes in memory every timestep…

GPU slowed down or crashing.

Complete refactor of the `Field class` to store fields and tensors in 1x1d array of length 3*Nx*Ny*Nz.

Access to data and matrix form thorugh magic methods (setter, getter, __setitem__, __getitem__, __add__, __mul__, __div__,…), completely transparent to the solver and the rest of the code ☺.




Speedup: from 110 to 90s, same simulation

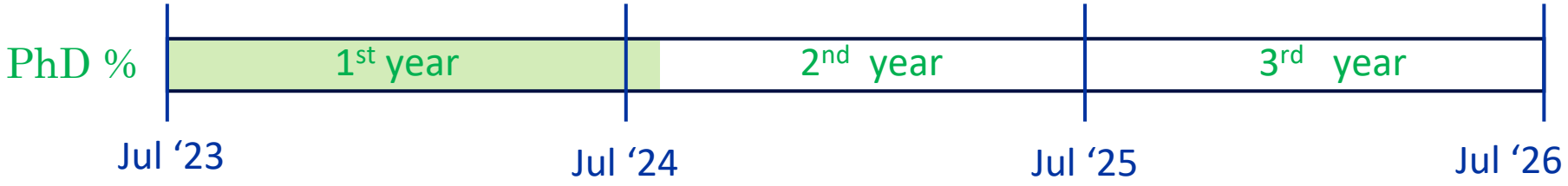
GPU happy(er), still room for improvement in memory alloc

Check Here

GitHub

Refactor comparison

# Wake solver milestones:




meetings #17, #18, #19, CEI

- 0. Wake potential and impedance
- 1. FIT Maxwell Equations in 3D
- 2. PEC, Periodic, PMC boundaries
- 3. Embedded Boundaries: geometry import from **.stl** files
- 4. Beam injection $\mathbf{J_z}$
- 5. Materials: $\varepsilon, \mu, \sigma$

In progress

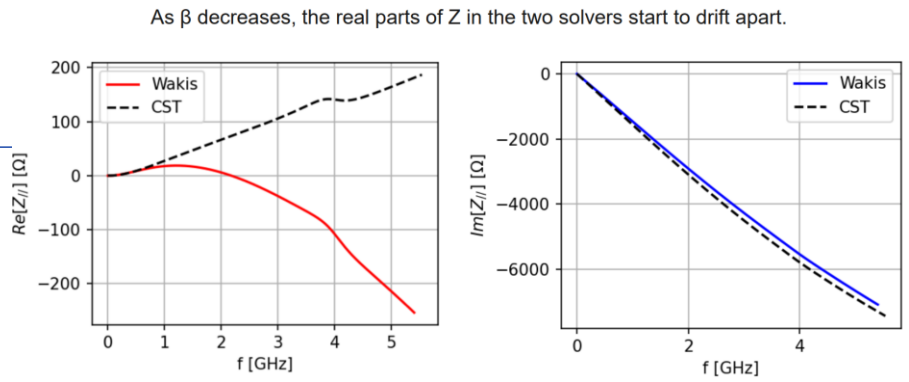
- x. Documentation
- 6. on GPU & memory opt.
- 7. Low beta & SC
- 8. Open boundaries & PML
- 9. Numerical tests vs analytic

To Do

- 10. Leontovich condition for good conductors >100 S/m
- 11. Staircased geometry: PBA/grid refinement
- 12. Moving window solver (?)
- 13. Frequency-dependent materials
- 14. …

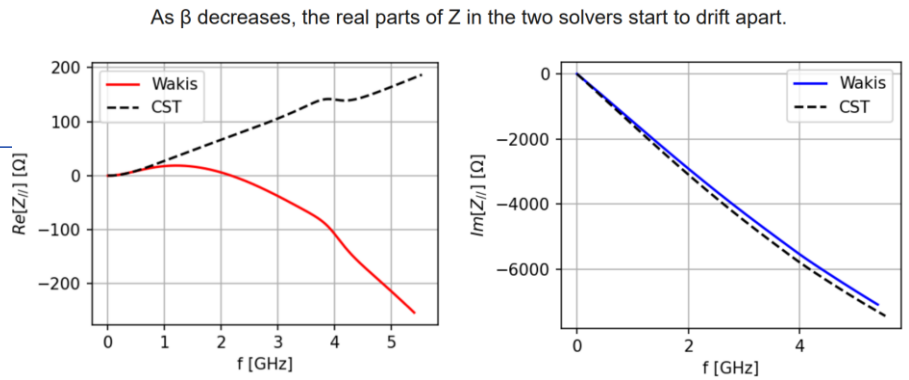
# Low-beta simulations

Thanks to Elena Macchia's work with `wakis`, we noticed some disagreement in the Real part of the impedance that worsen with lower β:



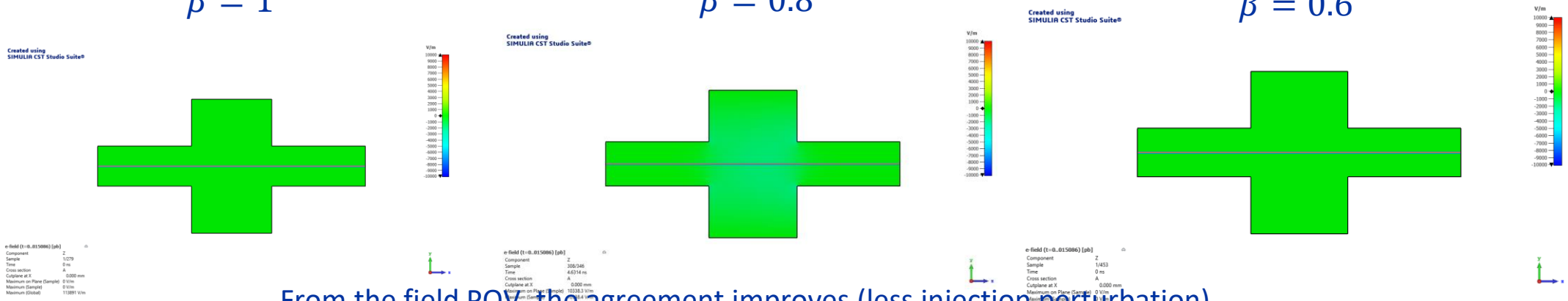
# Low-beta simulations

Thanks to Elena Macchia's work with `wakis`, we noticed some disagreement in the Real part of the impedance that worsen with lower β:



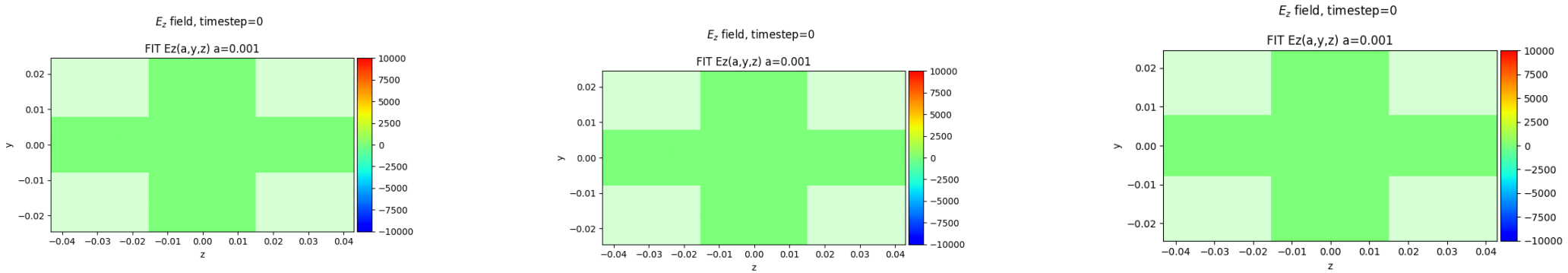

$\beta = 1$ $\beta = 0.8$ $\beta = 0.6$

CST



From the field POV, the agreement improves (less injection perturbation)

wakis

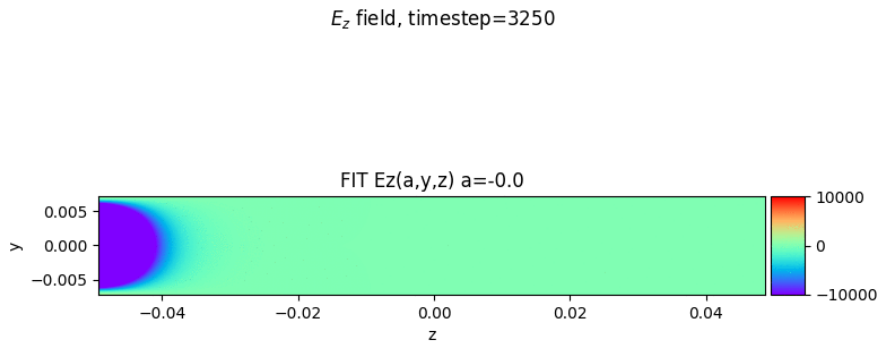



Found that injection time in CST follows the formula: $t_{inj} = \frac{8.54\sigma_z}{c\beta^{3/2}}$

# Low-beta simulations (II)

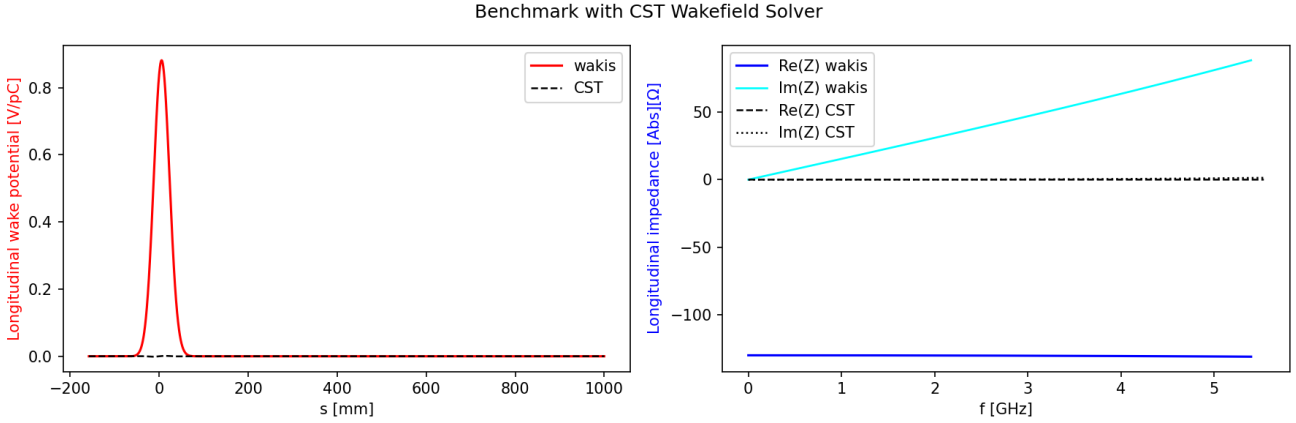
Testing with a simple squared pipe in CST: need Re = 0 and only reactive (Im) impedance, increasing with $\beta$. Quite difficult to converge even in CST (need 80 cells per wavelength)

For $\beta = 1$




WP and impedance coming from injection perturbation

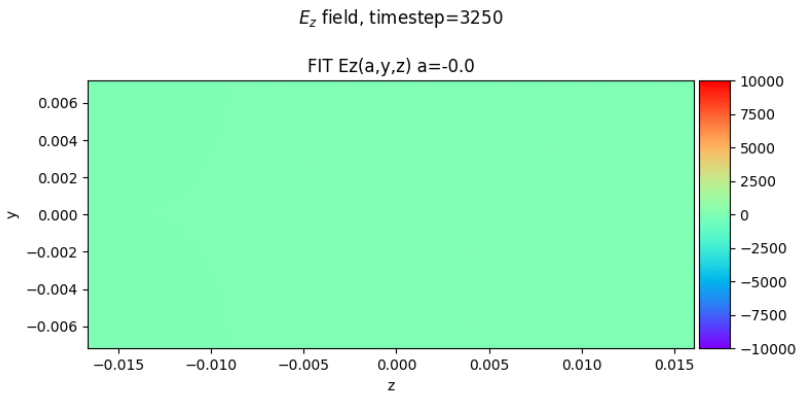
add_space = 0



# Low-beta simulations (II)

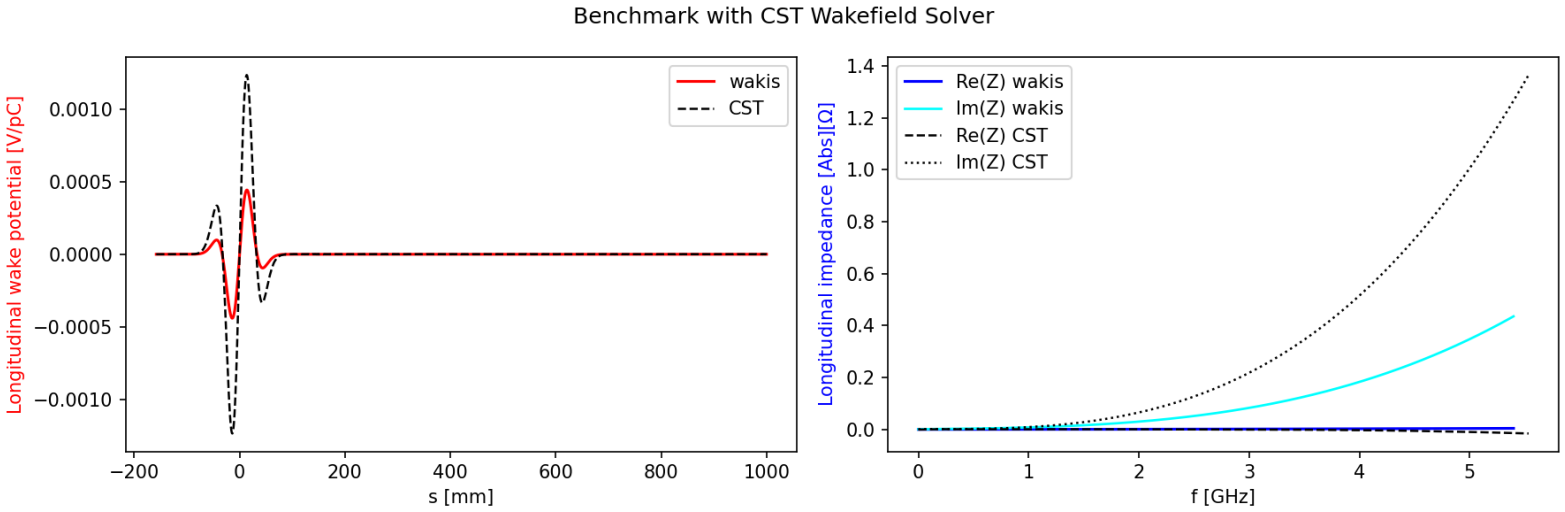
Testing with a simple squared pipe in CST: need Re = 0 and only reactive (Im) impedance, increasing with $\beta$. Quite difficult to converge even in CST (need 80 cells per wavelength)

For $\beta = 1$




Removing the first and last 50(!!) cells of the domain, the perturbation vanishes, finding expected WP and Z

add_space = 50




Is this what CST does behind the scene??

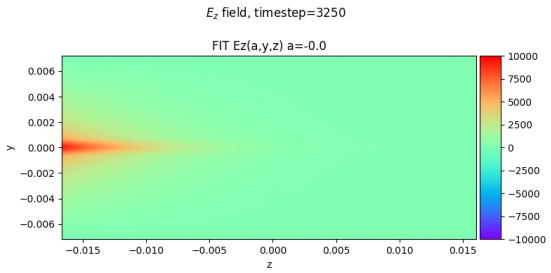
# Low-beta simulations (III)

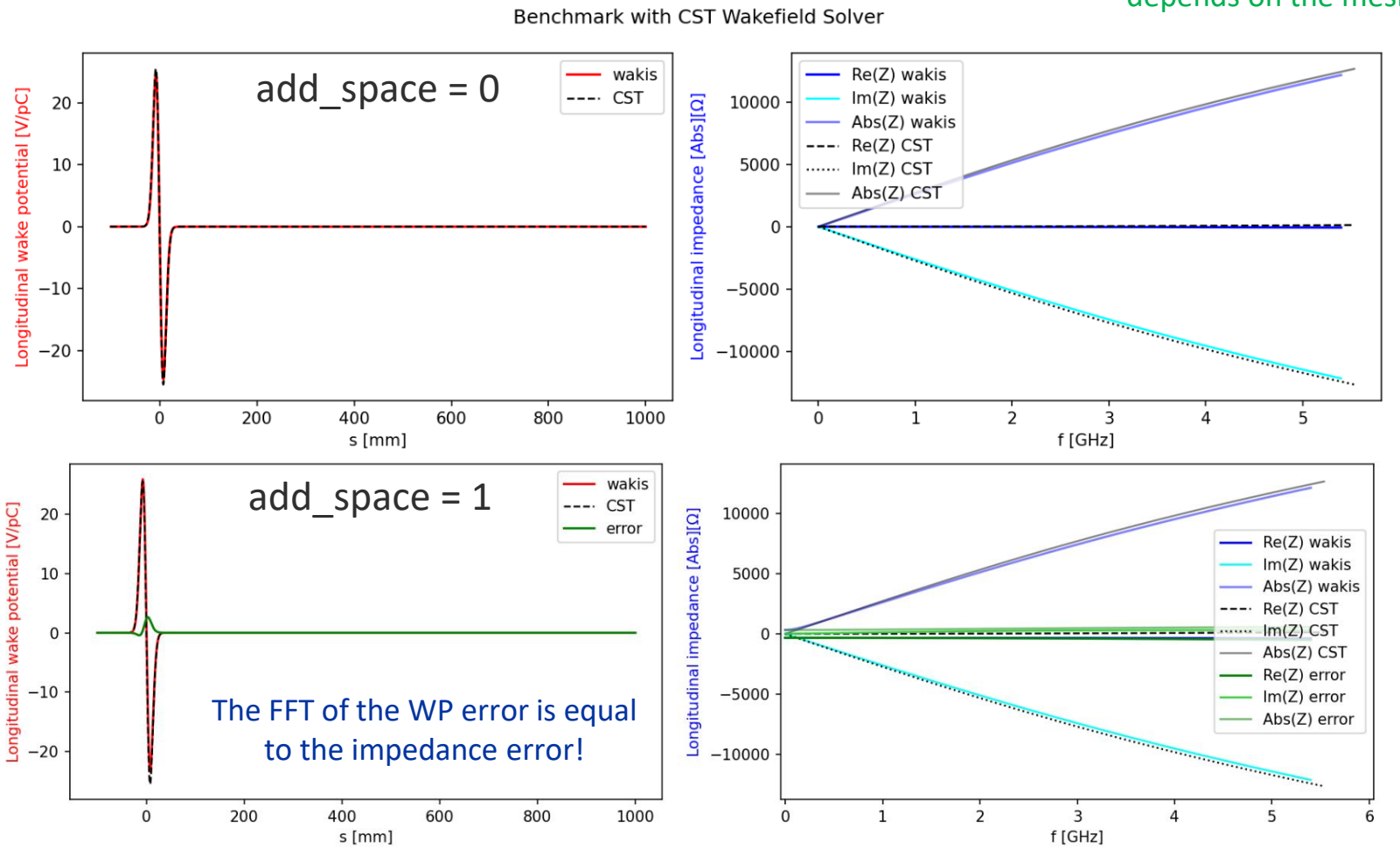
Testing with a simple squared pipe in CST: need Re = 0 and only reactive (Im) impedance, increasing with $\beta$. Quite difficult to converge even in CST (need 80 cells per wavelength)

We could make a model to remove this SC from the final WP, it only depends on the mesh

For $\beta < 1$

We find the opposite behavior… the more cells we skip, the worse agreement



# Wake solver milestones:

PhD %

| 1st year | 2nd year | 3rd year |
|---|---|---|

Jul '23 — Jul '24 — Jul '25 — Jul '26

meetings #17, #18, #19, CEI

- 0. Wake potential and impedance
- 1. FIT Maxwell Equations in 3D
- 2. PEC, Periodic, PMC boundaries
- 3. Embedded Boundaries: geometry import from **.stl** files
- 4. Beam injection $\mathbf{J_z}$
- 5. Materials: $\varepsilon, \mu, \sigma$

In progress

- x. Documentation
- 6. on GPU & memory opt.
- 7. Low beta & SC
- 8. Open boundaries & PML
- 9. Numerical tests vs analytic

To Do

- 10. Leontovich condition for good conductors >100 S/m
- 11. Staircased geometry: PBA/grid refinement
- 12. Moving window solver (?)
- 13. Frequency-dependent materials
- 14. …

# PML implementation

- Derivation from Snell and Fresnel laws: Codimd (needs a dedicated presentation…)
- Summary:
  - Many different types, some affect the curl operator (convolutional) some have multiple poles, etc…
  - For now, the simplest PML consist of adding an anisotropic conductivity that grows exponentially inside the PML

$$\sigma_x = \frac{\varepsilon_0}{2\Delta t}\left(\frac{x_{N_{pml}} - x_{0_{pml}}}{N_{pml}\Delta x}\right)^n \text{ for E}$$

$$\sigma_x^* = \mu_0 \sigma_x \text{ for H (optional)?}$$

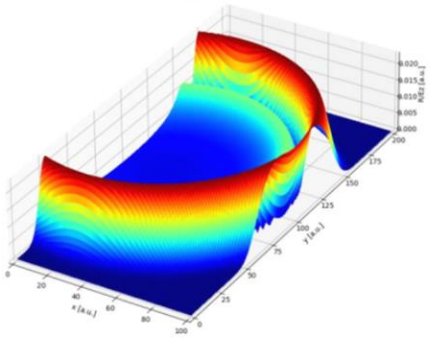
We add it to the conductivity tensor in the PML region before the time-stepping
(all pre-computed)

```python
def fill_pml_sigmas(self):
    '''
    Routine to calculate pml sigmas and apply them
    to the conductivity tensor sigma
    '''

    # Initialize
    sx, sy, sz = np.zeros(self.Nx), np.zeros(self.Ny), np.zeros(self.Nz)
    pml_exp = 2

    # Fill
    if self.bc_low[0].lower() == 'pml':
        sx[0:self.npml] = eps_0/(2*self.dt)*((self.x[self.npml] - self.x[:self.npml])/(self.npml*self.dx))**pml_exp
        for d in ['x', 'y', 'z']:
            for i in range(self.npml):
                self.sigma[i, :, :, d] = sx[i]
```

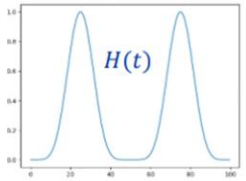
# Test PML reflection, sounds familiar…

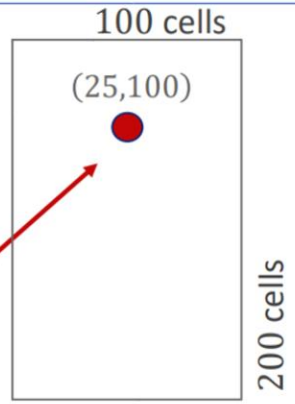
## PML study in Warp

**Test domain** $t = 200\delta t$

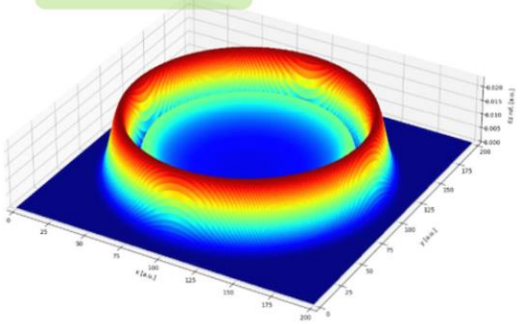


$100\delta x \times 200\delta y$
Simulate 200 t steps
$E_z(25,100) = H(t)$

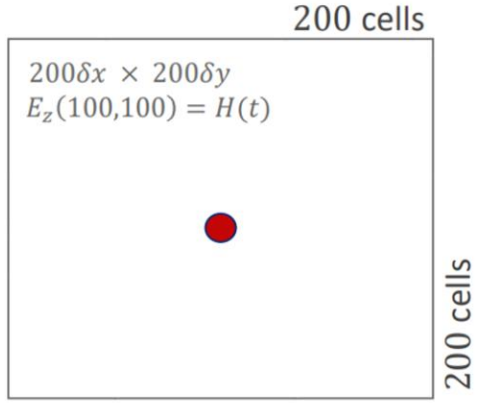



**Ref domain** $t = 200\delta t$



$200\delta x \times 200\delta y$
$E_z(100,100) = H(t)$




$$\boldsymbol{R}(0{:}75, 0{:}200) = \boldsymbol{E}_{z,\,test}(0{:}75, 0{:}200) - \boldsymbol{E}_{z,\,ref}(100{:}175, 0{:}200)$$

EMWSD / Wakis meeting #15 8

EMWSD #15

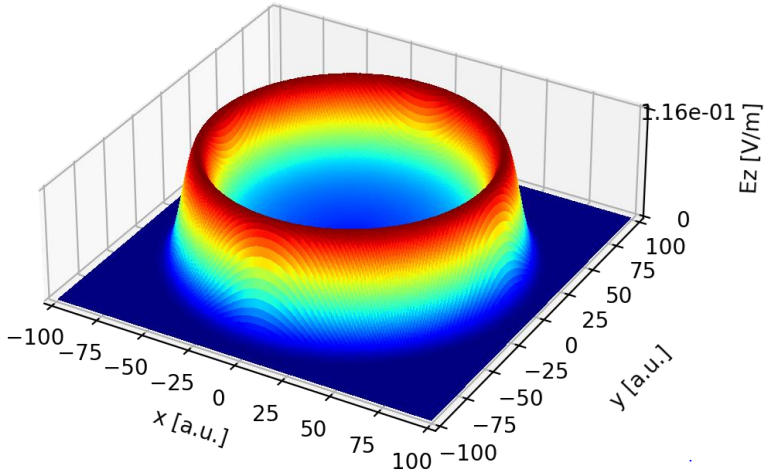
2D simulation in Warp
(never replicated in WarpX)

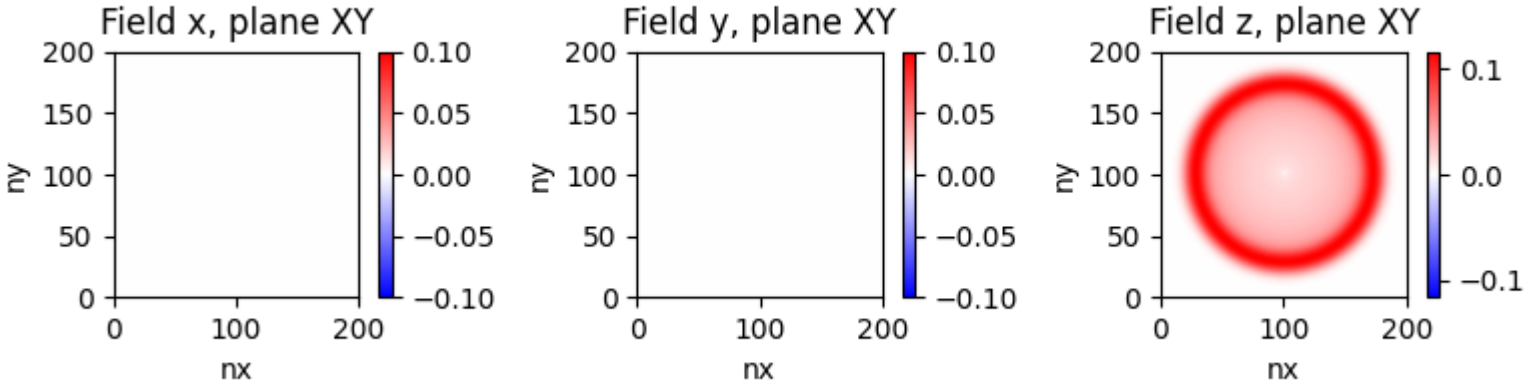
*Original test from Berenger paper presenting the PML formulation

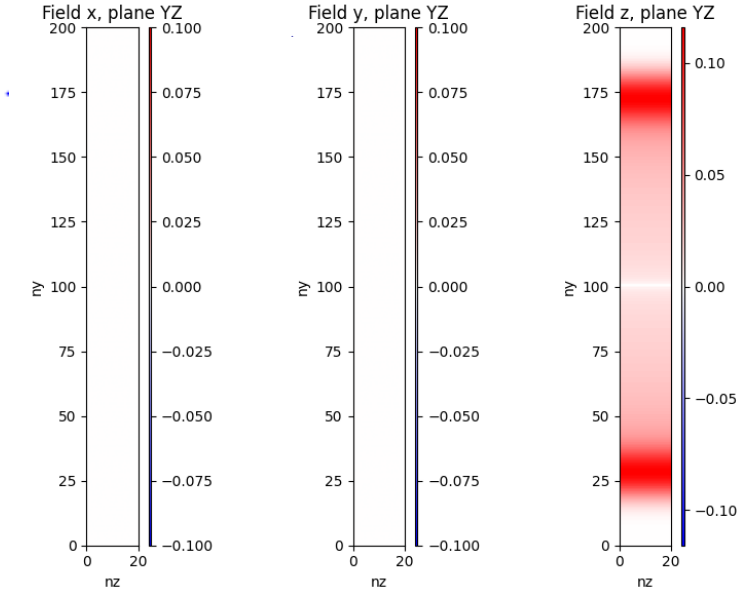
# Test PML in wakis

Replicating test and reference 2D simulations in wakis: **successful**

- Created new source class `Pulse` that injects at a given xs, ys, zs
- To have a 2D pulse, zs=slice(0, Nz) with PEC BCs
- This Harris pulse gives a very clean positive field pulse in Ez.
- All the other field components are 0



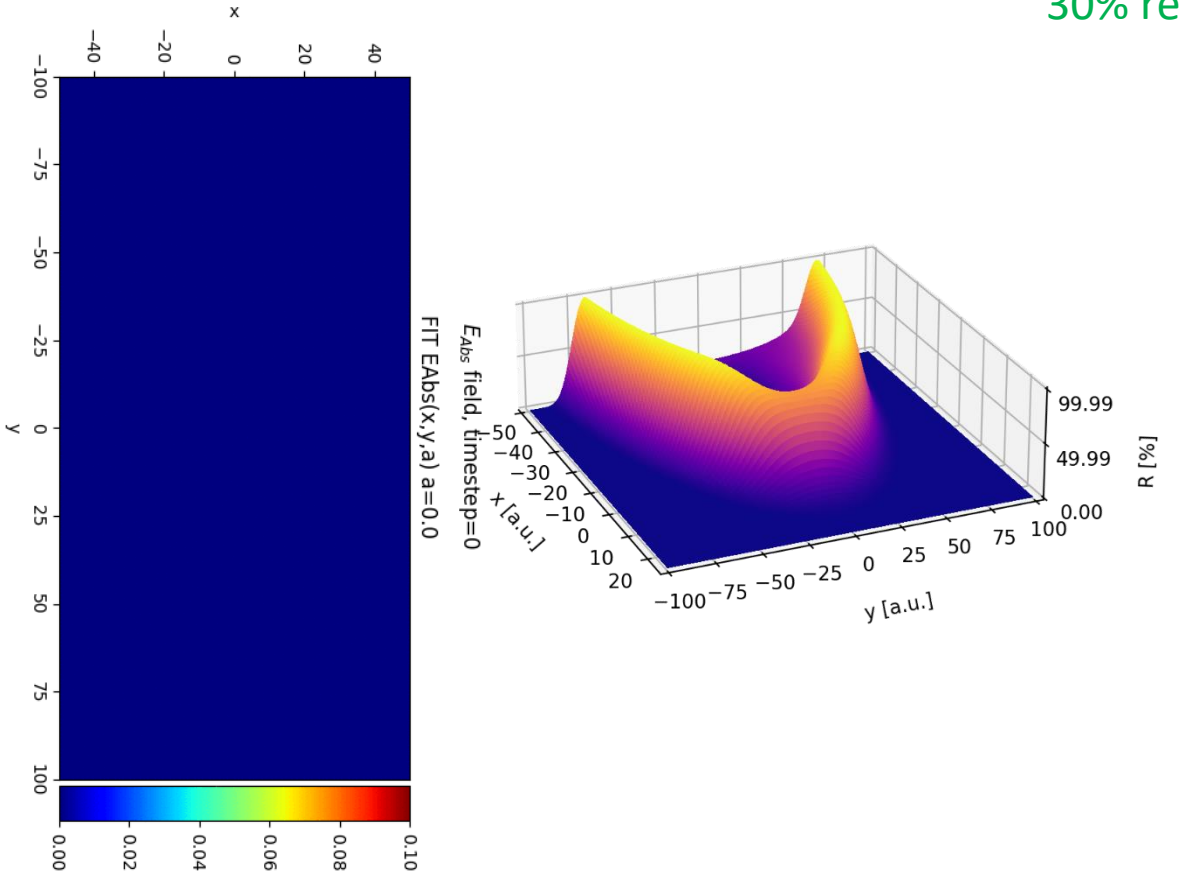



# Test PML in wakis (II)

**PEC boundaries X, Y**

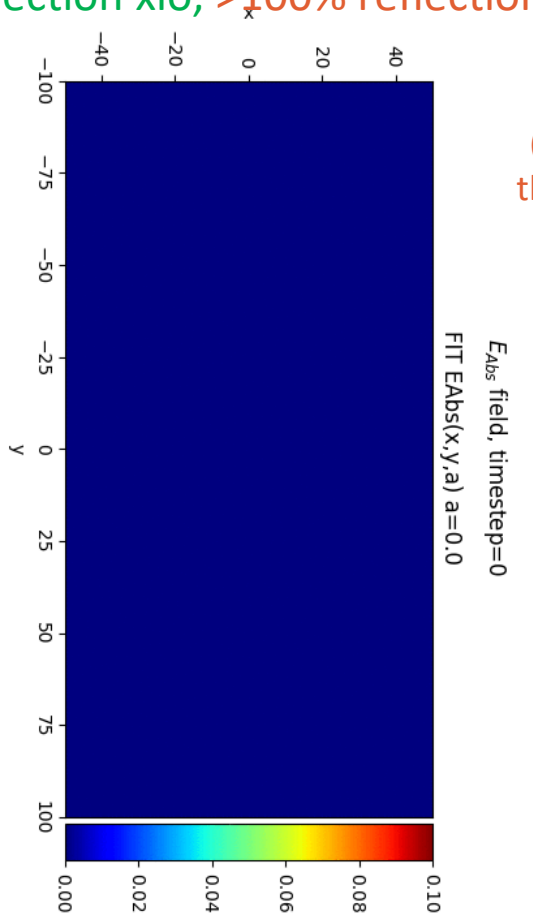
Total reflection

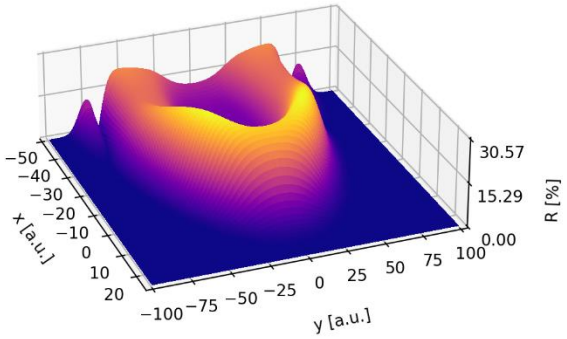



**PEC boundaries Y, ABC* boundaries X**

Absorption in Xlo, but not in Xhi → bug in ABC

30% reflection xlo, >100% reflection in xhi

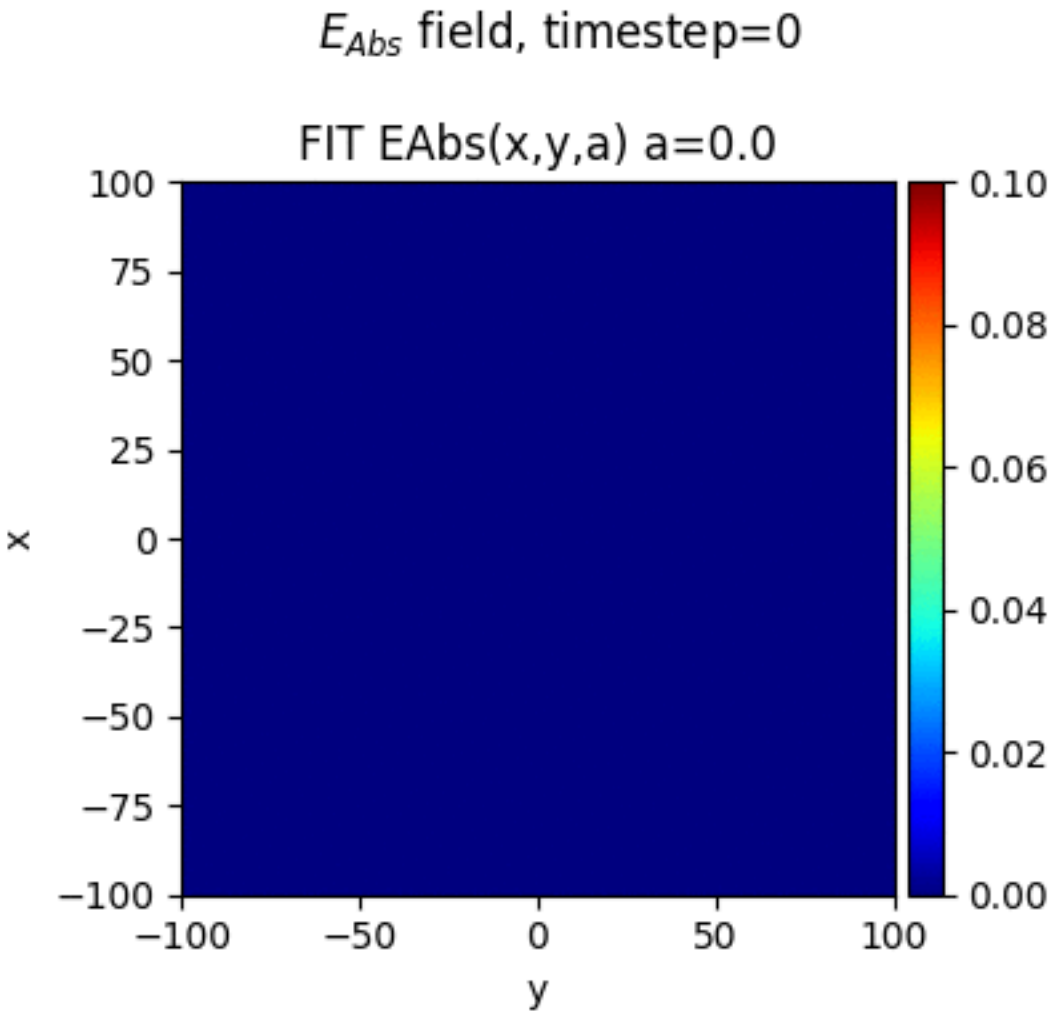



*Reminder:
ABC are First order extrapolation BCs (implemented with prof. Manuel Cotelo), they copy the field of the $x_{n-1}$ cell to $x_n$ at the BC

# Test PML in wakis (II)

**PEC boundaries Y**
**Periodic X**
No reflection at xlo
(but not suitable for impedance)



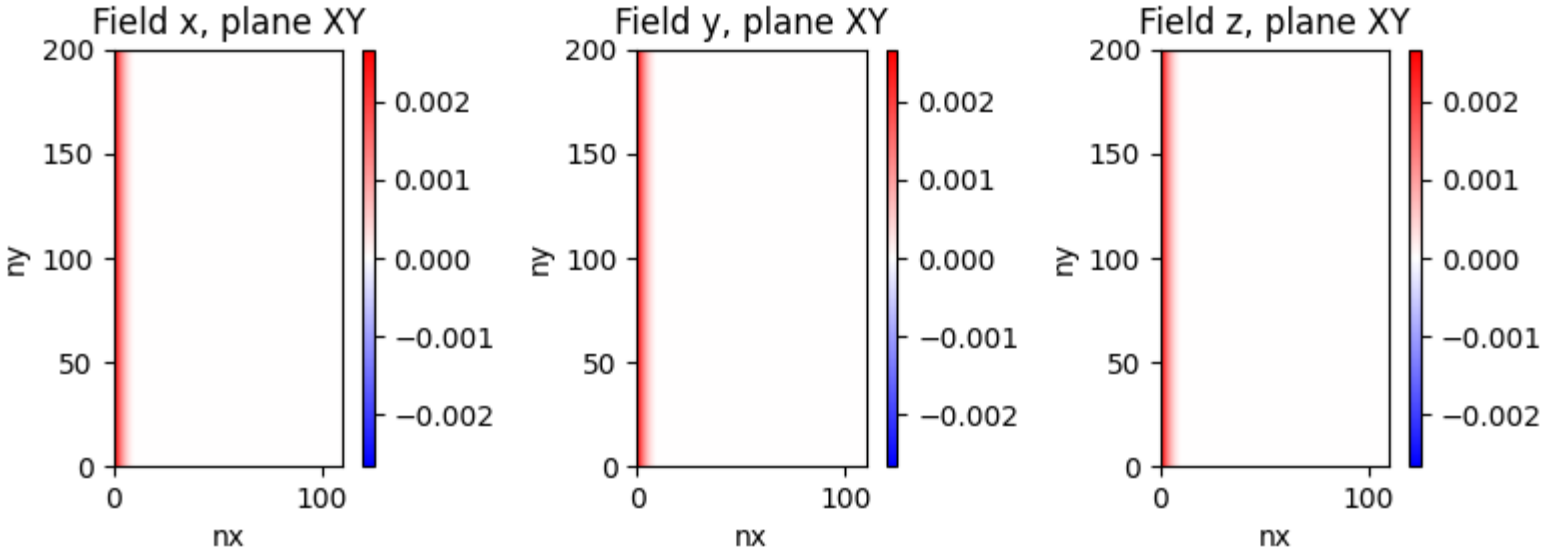
# Test PML in wakis (II)

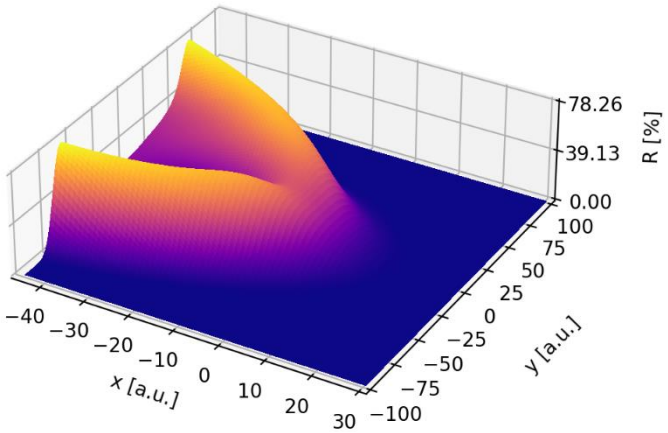
**PML boundaries X, Y**

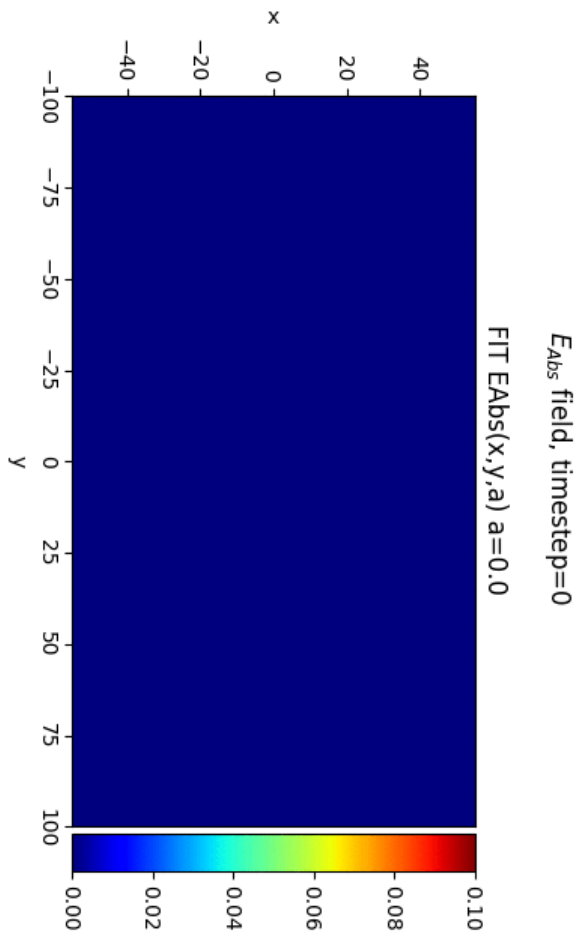
70% reflection, depends on $N_{pml}$ layers and max conductivity

$$\sigma_x = \frac{\varepsilon_0}{2\Delta t}\left(\frac{x_{N_{pml}} - x_{0_{pml}}}{N_{pml}\Delta x}\right)^2$$

$\sigma$ conductivity tensor with 10 PML layers in x



# Test PML in wakis (II)

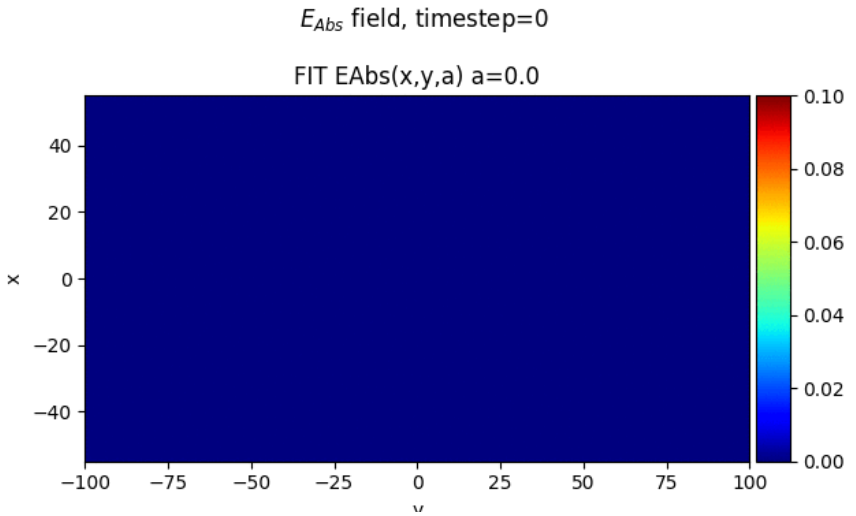
**PML boundaries X, Y**

70% reflection, depends on $N_{pml}$ layers and max conductivity

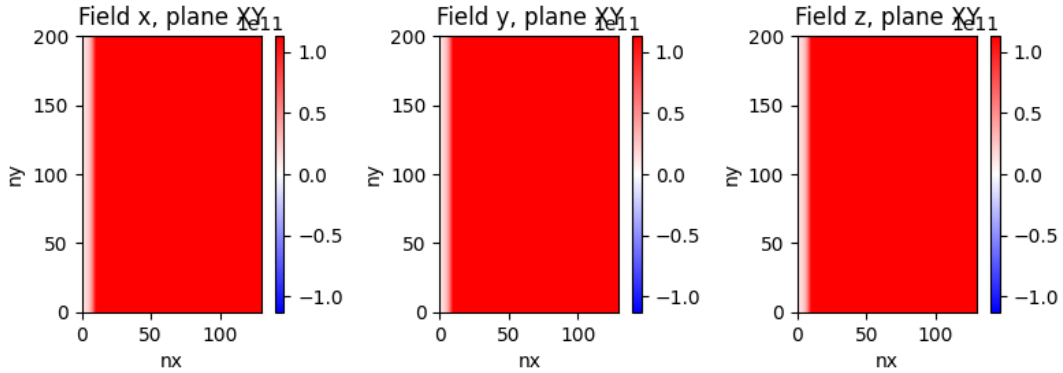
$$\sigma_x = \sigma_{max}\left(\frac{x_{N_{pml}} - x_{0_{pml}}}{N_{pml}\Delta x}\right)^2$$

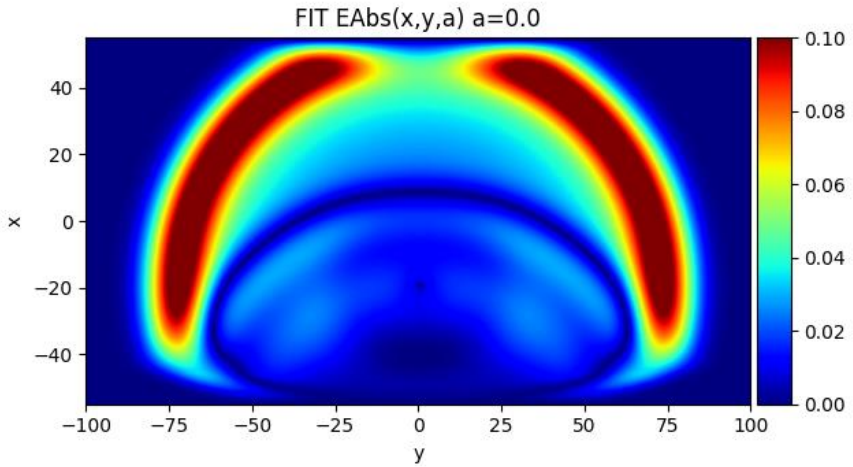
If $\sigma_{max} > \frac{\varepsilon_0}{2\Delta t}$ simulation unstable

Can be cured modifying $\bar{\bar{\varepsilon}}$ -> $\varepsilon_x = \frac{\sigma_{max}}{2}\varepsilon_0$




$\varepsilon^{-1}$ tensor with 30 PML layers in x



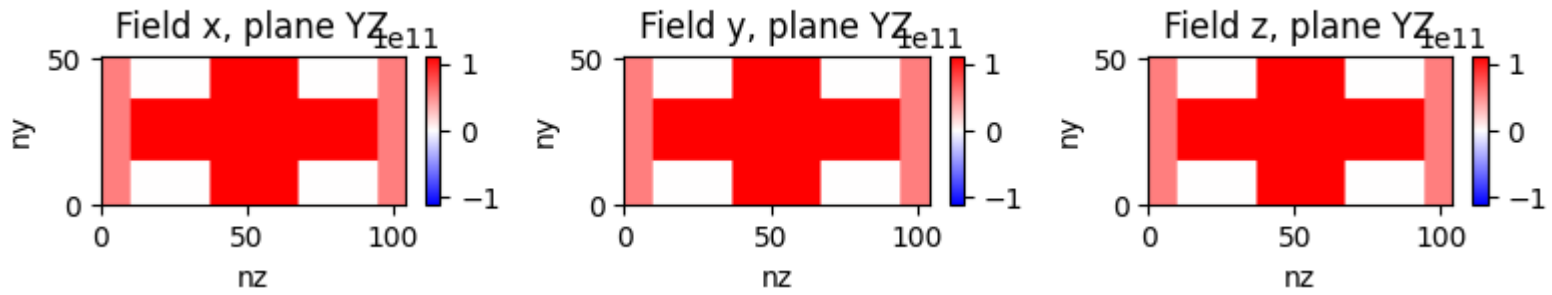



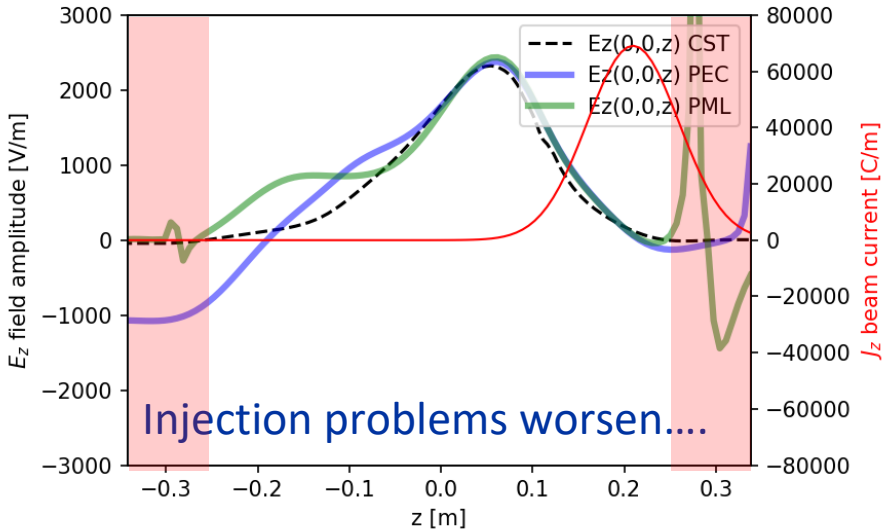

**WIP** to optimize $\sigma_{max}$ while keeping simulation stable

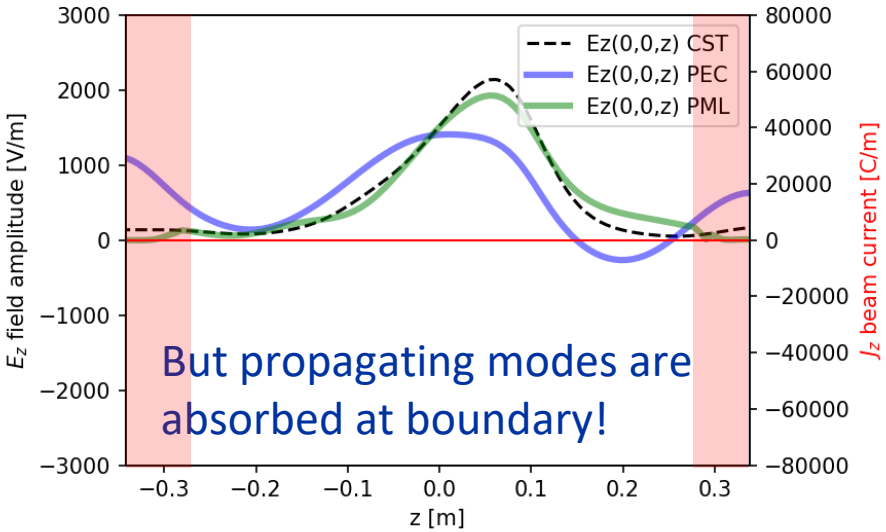
# PML on Impedance simulation

Just to try… I tested using this PML in a PEC cubic cavity with modes above cutoff

$\varepsilon^{-1}$ tensor



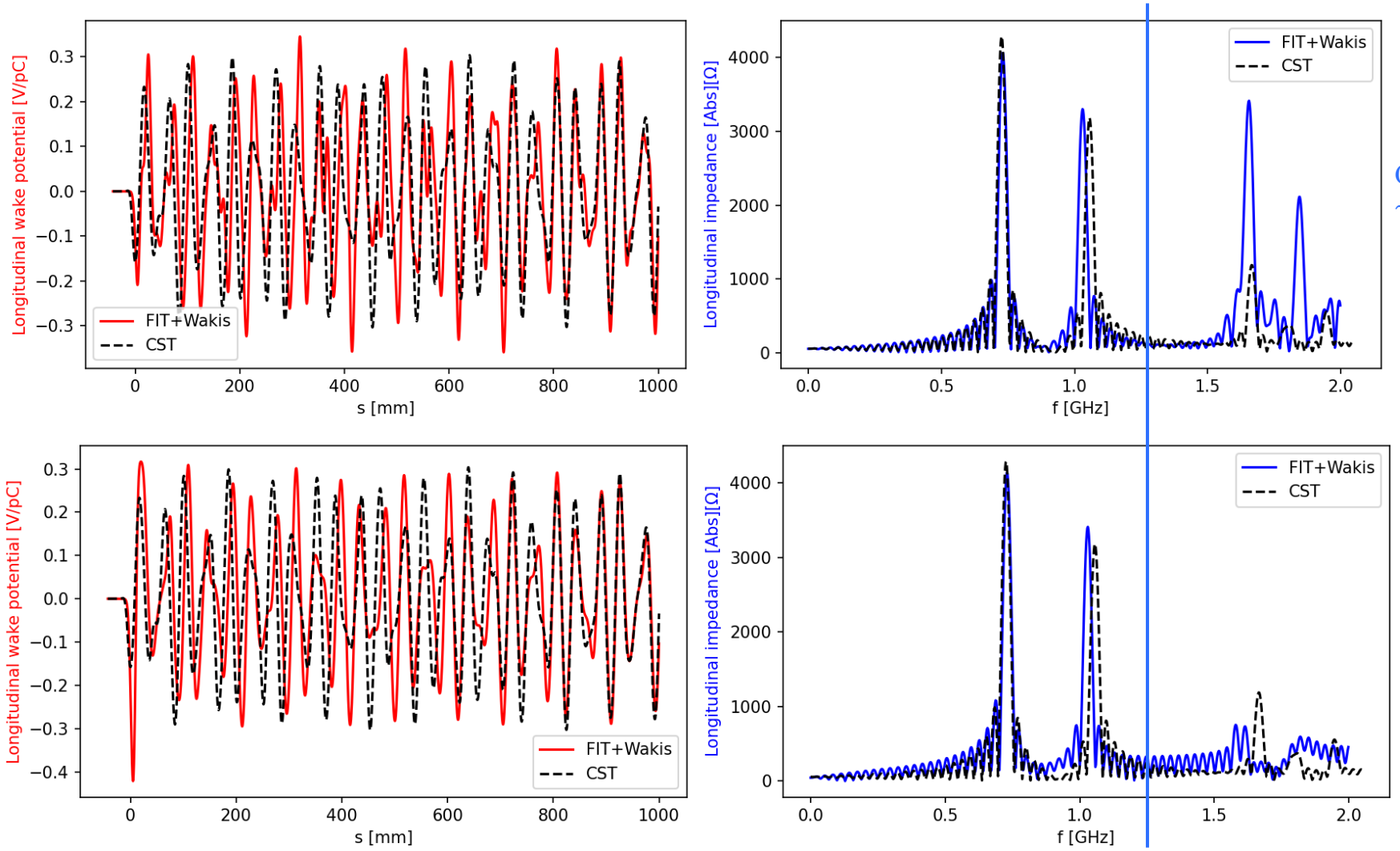



# PML on Impedance simulation (II)

Just to try… I tested using this PML in a PEC cubic cavity with modes above cutoff

PEC

PML

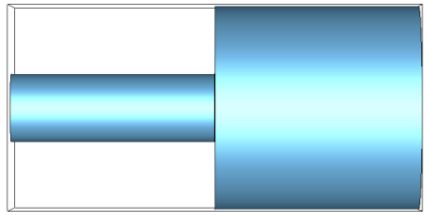


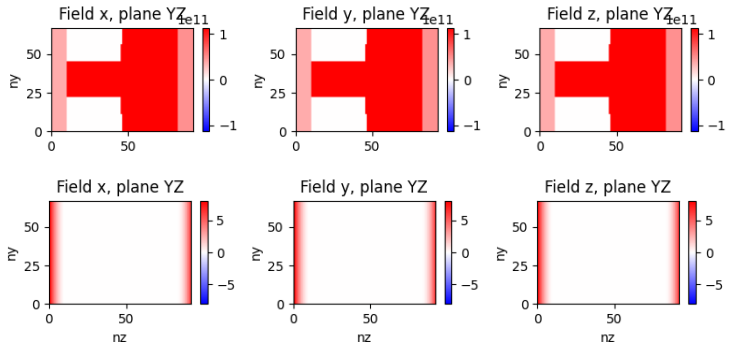

Cutoff frequency: ~1.25 GHz

# PML on Impedance simulation (III)

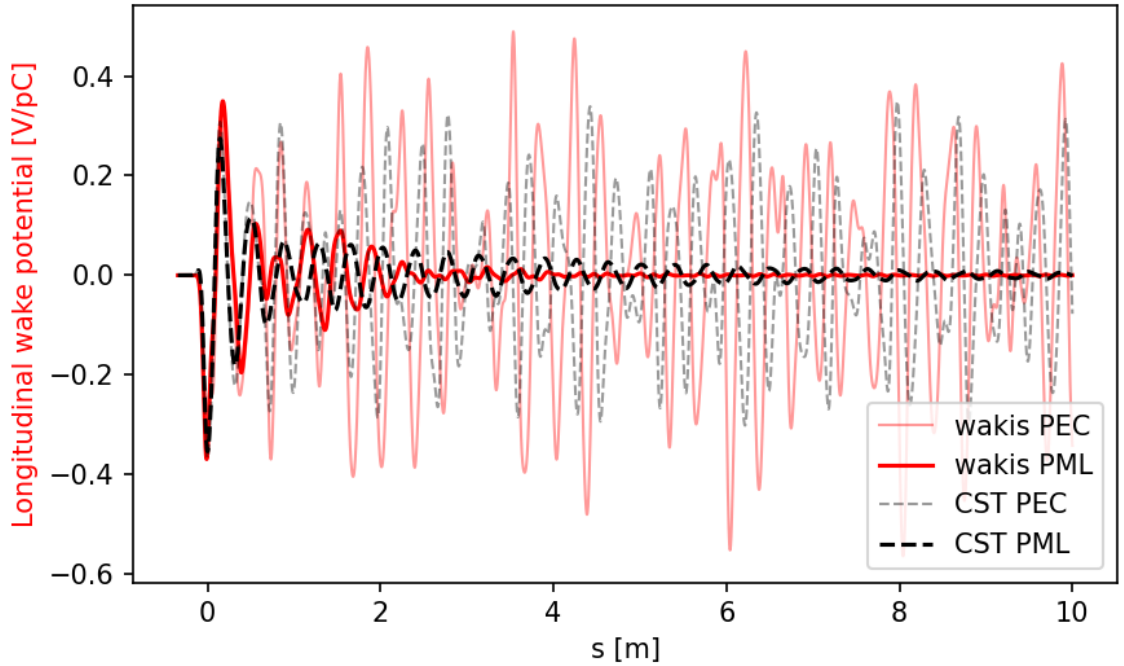
Worst case for PML: **PEC step-out transition**

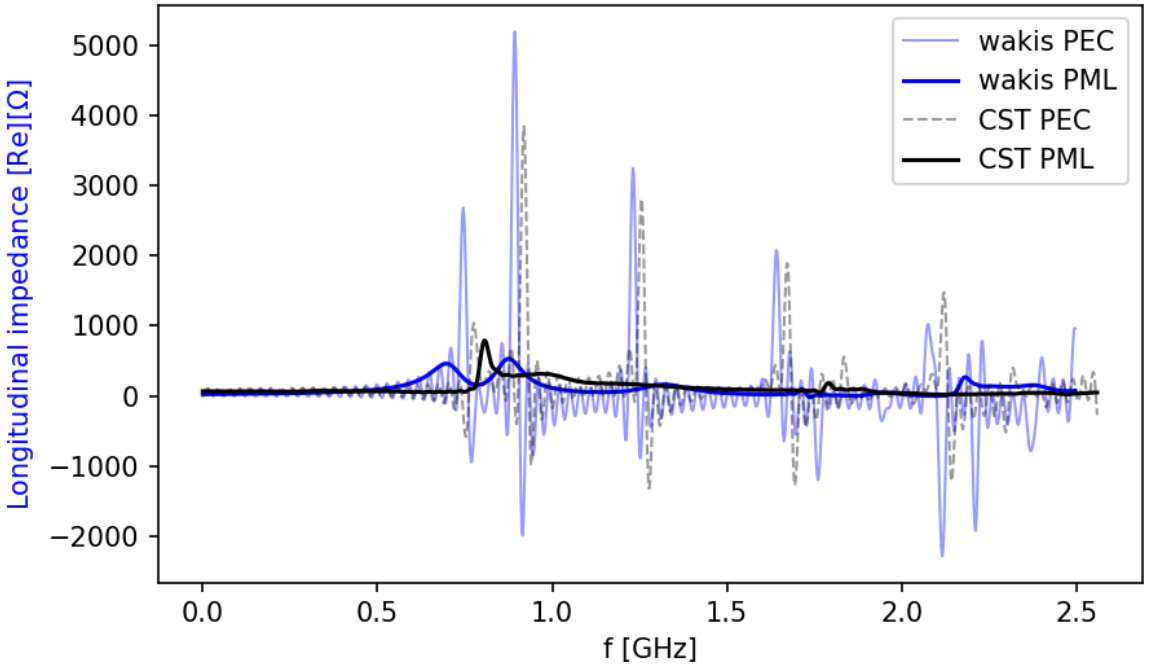
- Same mesh as CST
- 10 layer PML






Benchmark with CST Wakefield Solver



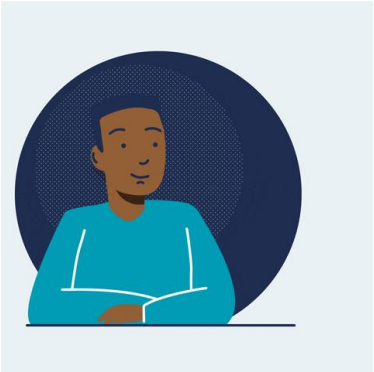

Surprisingly good results

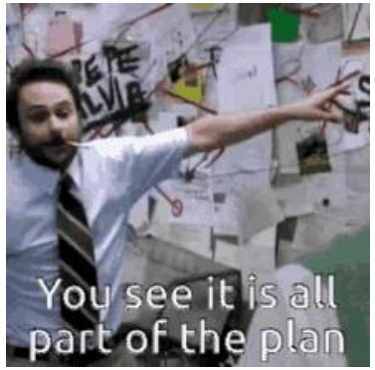
# Conclusions

- ✓ Progress on the code:
  - First users, documentation, work on Gitub workflow to keep a stable version
  - GPU acceleration successful, speedup x120 vs CPU for limit case 20 million cells
    - New bottleneck: memory alloc -> parallelize?
  - Low beta simulations:
    - Potential to understand space charge problem, create model to remove it (?)
    - Remove injection perturbation by enlarging domain only during excitation (?) only needed for beta=1
  - PML:
    - Mathematical derivation and 1st implementation done
    - Under reflection test -> 70% reflection
    - **Seems to work already for impedance simulations?**

# Conclusions & Future

- ✓ Progress on the code:
  - First users, documentation, work on Gitub workflow to keep a stable version
  - GPU acceleration successful, speedup x120 vs CPU for limit case 20 million cells
    - New bottleneck: memory alloc -> parallelize?
  - Low beta simulations:
    - Potential to understand space charge problem, create model to remove it (?)
    - Remove injection perturbation by enlarging domain only during excitation (?) only needed for beta=1
  - PML:
    - Mathematical derivation and 1st implementation done
    - Under reflection test -> 70% reflection
    - **Seems to work already for impedance simulations?**
- Still many things to do:
  - Good conductors, staircased geometry, grid refinement, moving window, frequency dependent materials, frequency domain monitors, TESTS …. Where to focus? IPAC in October





# Wake solver milestones:

PhD %

| 1st year | 2nd year | 3rd year |
|---|---|---|

Jul '23 — Jul '24 — Jul '25 — Jul '26

meetings #17, #18, #19, CEI

- 0. Wake potential and impedance
- 1. FIT Maxwell Equations in 3D
- 2. PEC, Periodic, PMC boundaries
- 3. Embedded Boundaries: geometry import from **.stl** files
- 4. Beam injection $\mathbf{J_z}$
- 5. Materials: $\varepsilon, \mu, \sigma$

In progress

- x. Documentation
- 6. on GPU & memory opt.
- 7. Low beta & SC
- 8. Open boundaries & PML
- 9. Numerical tests vs analytic

To Do

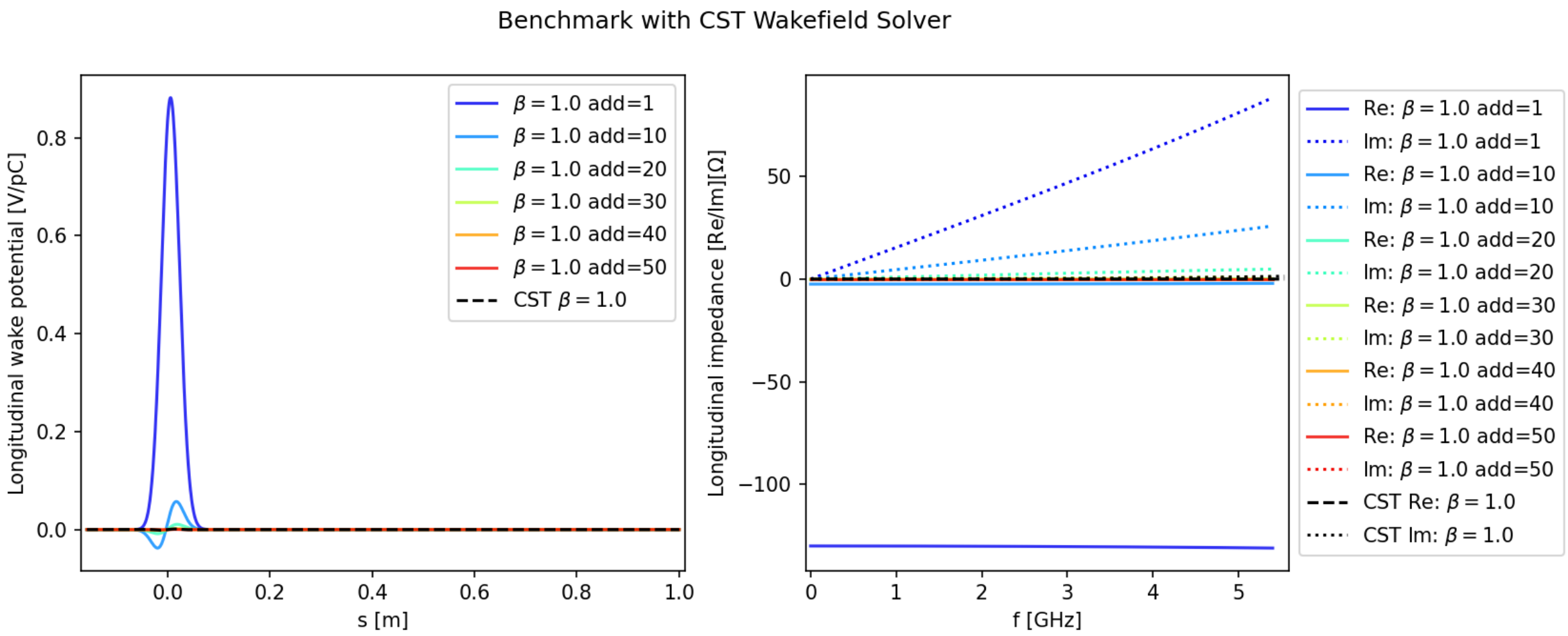
- 10. Leontovich condition for good conductors >100 S/m
- 11. Staircased geometry: PBA/grid refinement
- 12. Moving window solver (?)
- 13. Frequency-dependent materials
- 14. …

# Thank you ☺ !!!

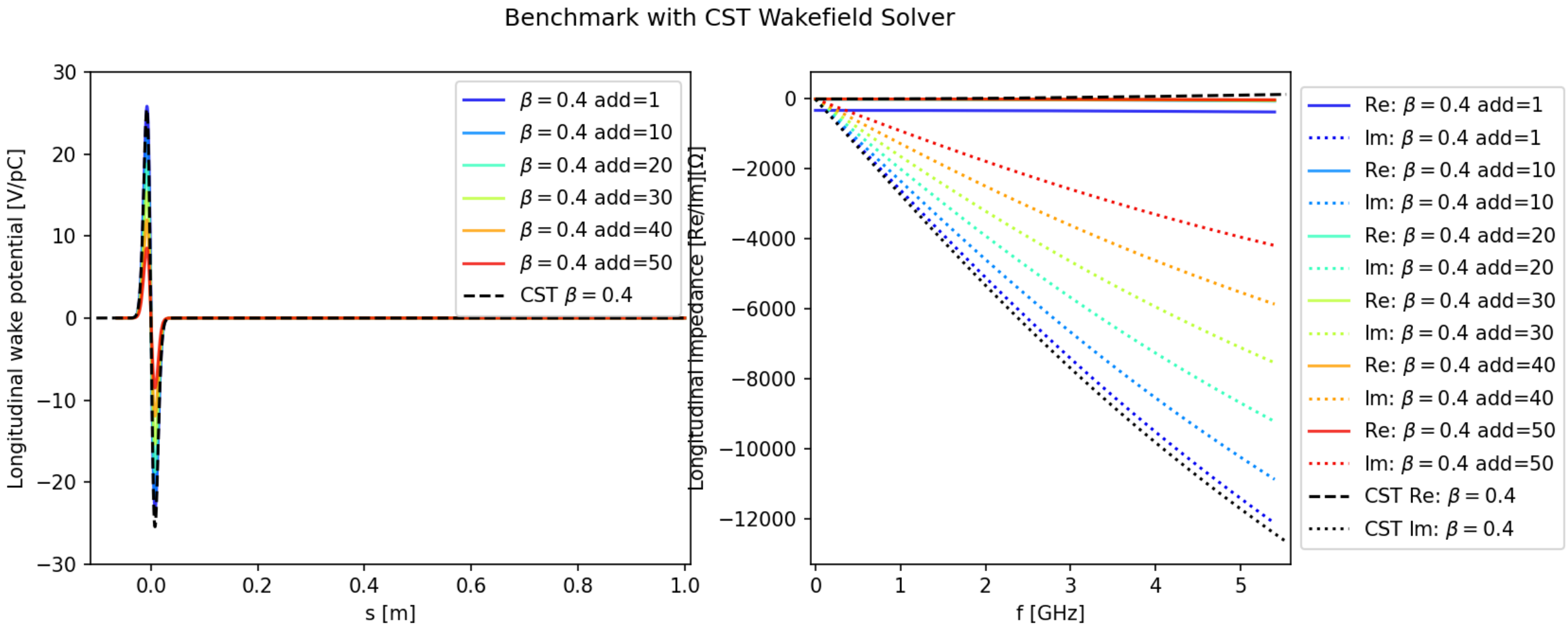
Electromagnetic and Wake Solver Development
meeting #20

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

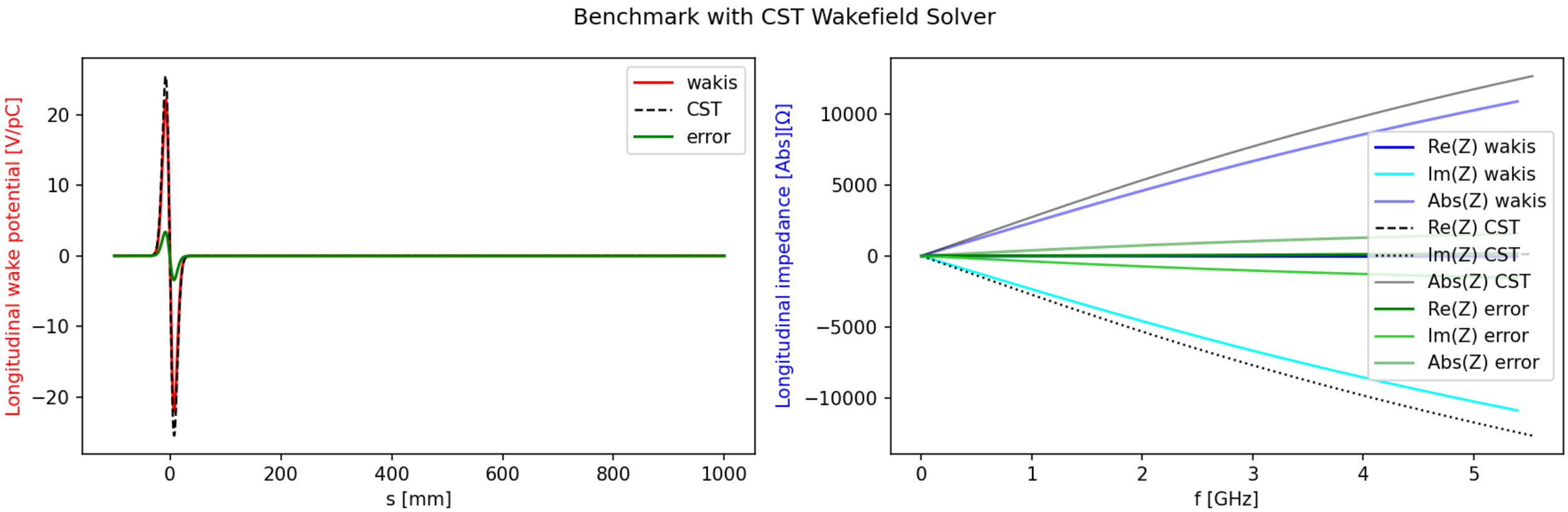
# Backup: beta=1, different add_space



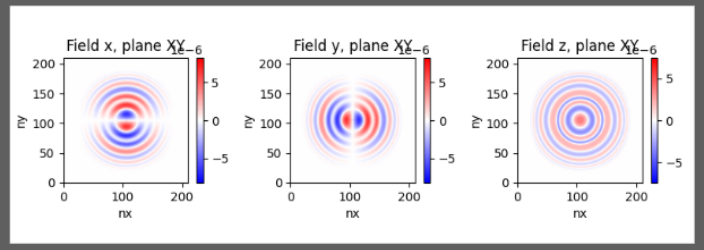
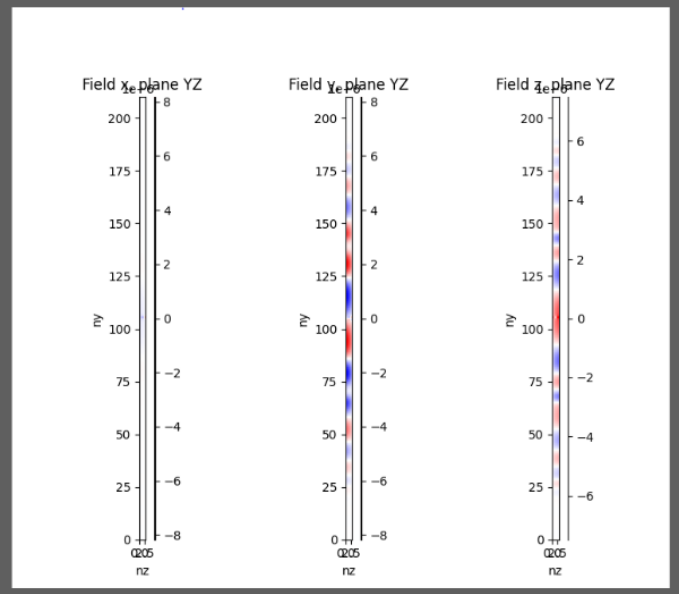
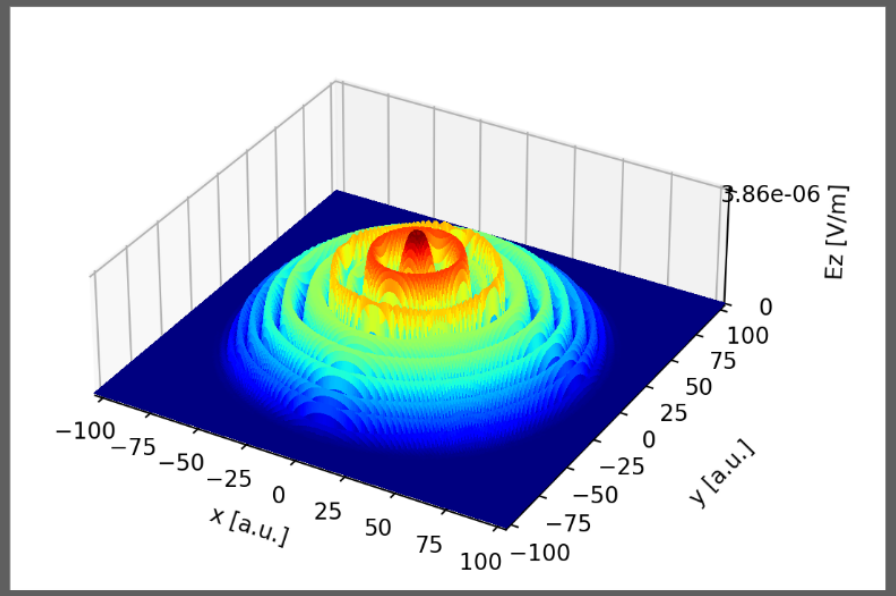
# Backup: beta=0.4, different add_space



# Backup: beta=0.4, error add_space=10



# Backup: issues 3D to 2D sim

| plane XY | plane YZ | 3D view |
|---|---|---|
|  |  |  |

# Berenguer PML update equations

Not working, all fields = 0

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

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

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

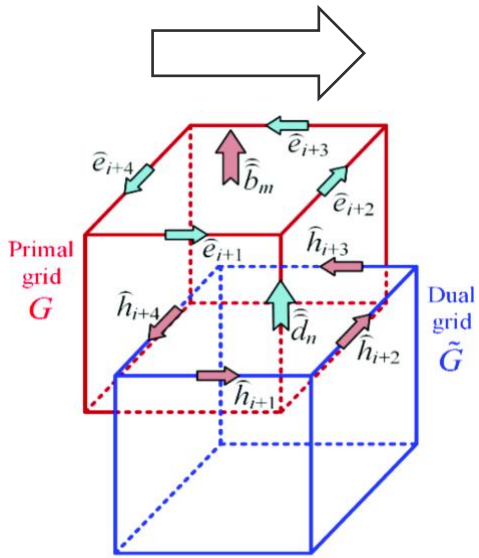
$$h^{n+1} = h^n - \Delta t\, \tilde{D}_s D_\mu^{-1} D_A^{-1} C e^{n+0.5}$$

```python
def one_step_etd(self):
    '''[TODO] Not working
    '''
    if self.step_0:
        self.set_ghosts_to_0()
        self.step_0 = False

        #cleanup
        del self.itDaiDepsDstC

        #pre-compute
        a, b = 1.0, self.sigma.toarray()
        isigma = np.divide(a, b, out=np.zeros_like(b), where=b!=0)

        self.iDsigma = diags(isigma, shape=(3*self.N, 3*self.N), dtype=float)
        self.Dexp = diags(np.exp(-self.ieps.toarray()*self.sigma.toarray()*self.dt),
                          shape=(3*self.N, 3*self.N), dtype=float)
        self.oneMinusDexp = diags(1.0-np.exp(-self.ieps.toarray()*self.sigma.toarray()*self.dt),
                          shape=(3*self.N, 3*self.N), dtype=float)
        self.itDaiDsigmaDstC = self.itDa * self.iDsigma * self.Ds * self.C.transpose()

        del a, b, isigma
        self.attrcleanup()

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

    self.E.fromarray(self.Dexp*self.E.toarray() +
                     (self.oneMinusDexp)*self.itDaiDsigmaDstC*self.H.toarray() -
                     (self.oneMinusDexp)*self.iDsigma*self.J.toarray()
                     )

    self.J.fromarray(self.Dsigma*self.E.toarray())

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

# FIT theory: Grid Maxwell Equations

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

**FIT**




Grid Maxwell Equations

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

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

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

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

- Operators
- Spatial matrixes
- Material properties

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

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

What we care about:
**Update equations**

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

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

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