EMWSD
Electromagnetic and Wake Solver Development

Meeting #02

Elena de la Fuente, Carlo Zannini, Lorenzo Giacomel

Excused: Giovanni Iadarola

Outline

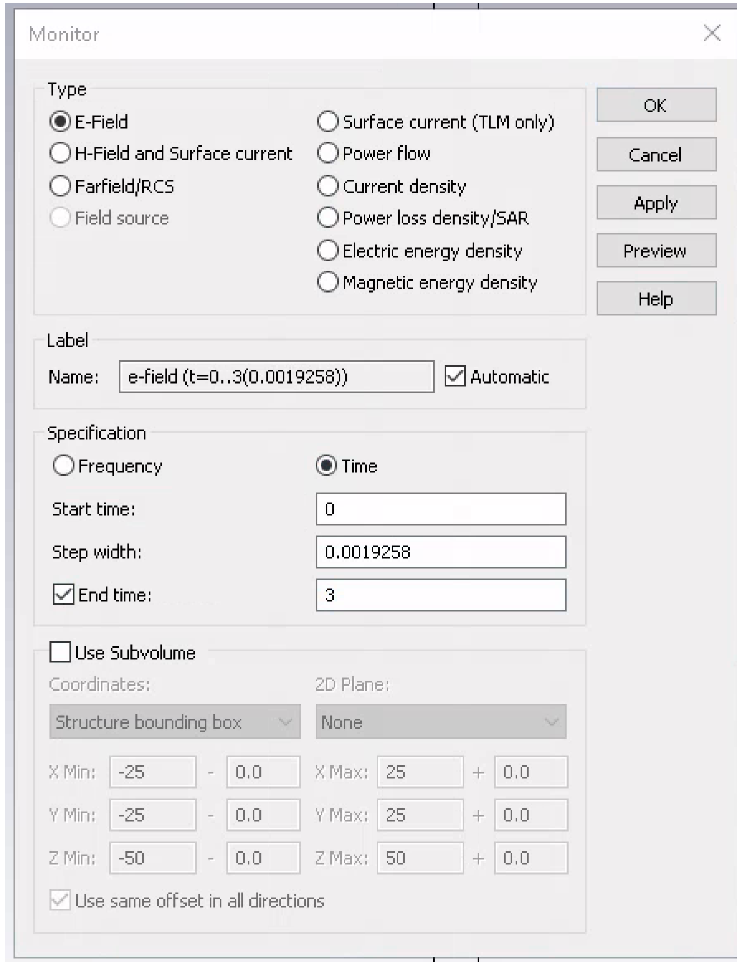
1. Ez field comparison

2. Direct algorithm results

3. Indirect algorithm results

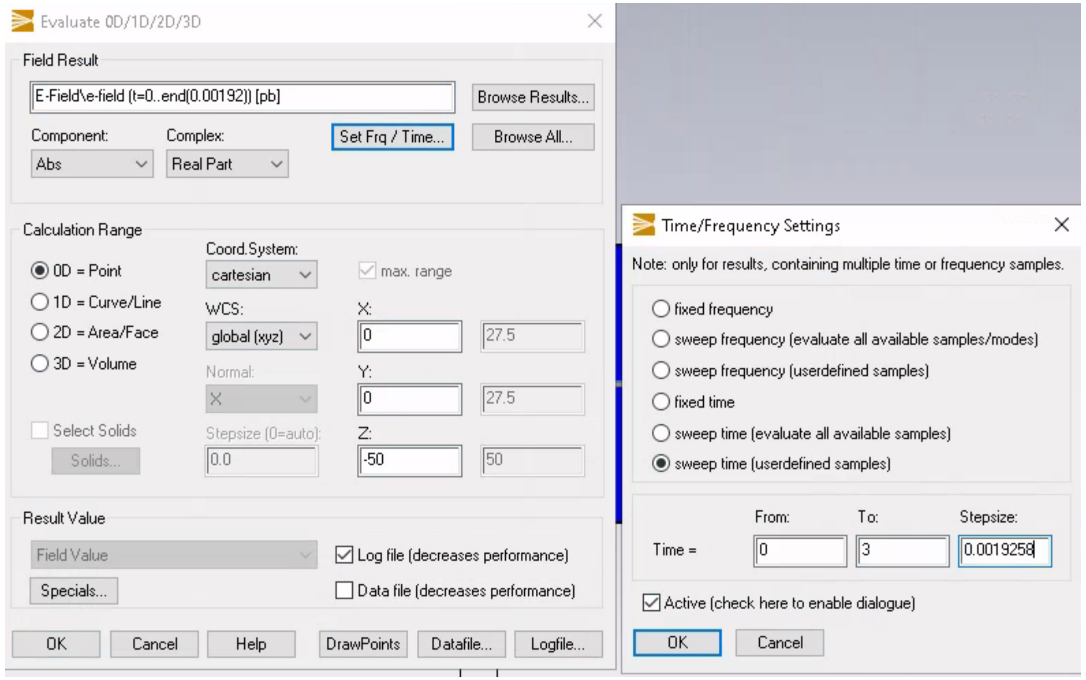
Conclusions and next steps

CST field extraction

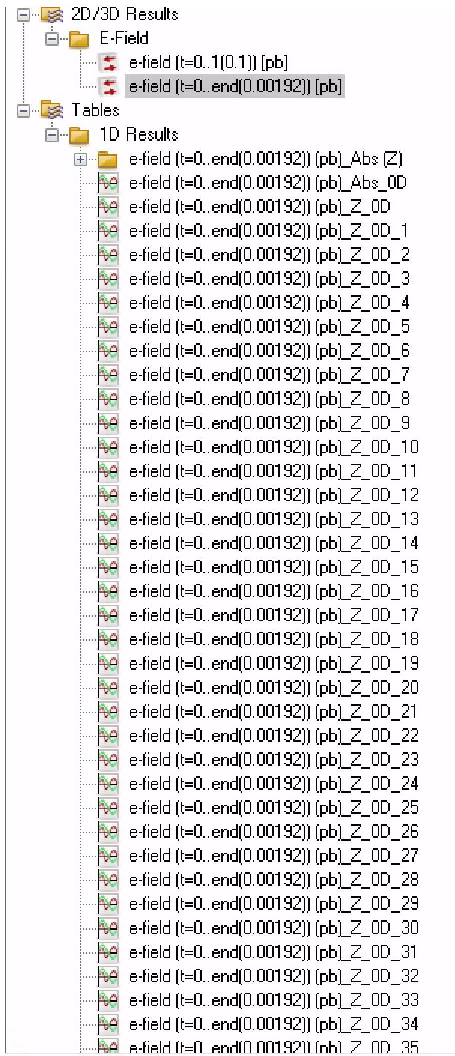


1) Field monitor with same timestep as the Warp simulation: dt=0.0019258

2) Extracting the 1D plots for all the time samples for each value of z_k



3) Generating all the plots and exporting the data to txt files (computationaly costly and time consuming)



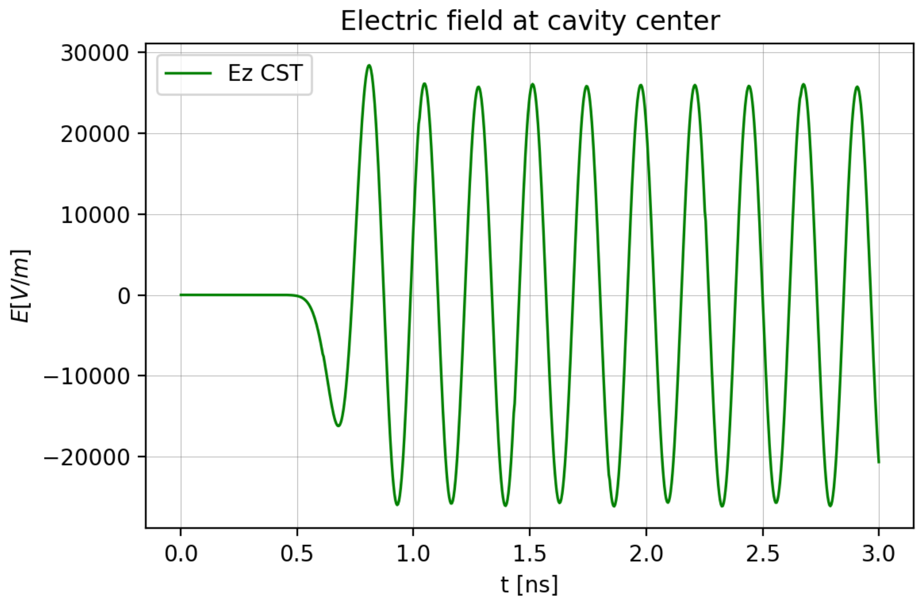
CST field extraction

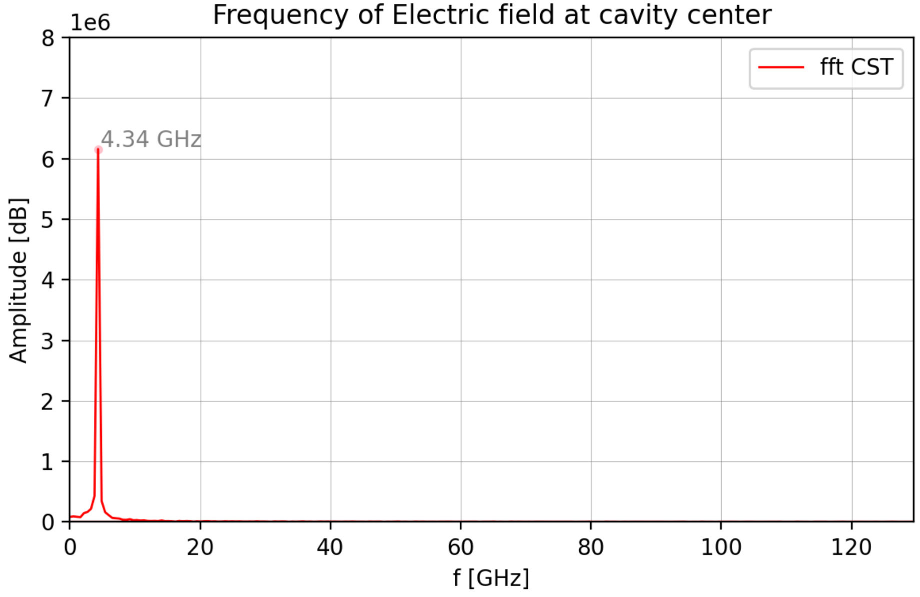
Cst_to_dict.py

4) Python script to read all the txt, plot the results and store the field matrix $E_z[z,t]$ in a dictionary with **pickle**

cst_to_dict.py | postproc.py | cube_cavity_impedance.py | cube_cavity.py | wake_potential_cst.py

```python
'''
cst_to_dict.py

File for postprocessing logfiles from cst

--- Reads 1 log file and plots the field and the frequency
--- Reads all log files and dumps the E(z,t) matrix into a dict
--- Saves the dict in a out file 'cst.txt' with pickle

'''

import numpy as np
import matplotlib.pyplot as plt
import time
import sys
import glob, os
import scipy as sc
import pickle as pk


#--- read one file
fname = 'Ez_050'

#Initialize variables
Ez=[]
t=[]
i=0

with open('cst_files/'+fname+'.txt') as f:
    for line in f:
        i+=1
        content = f.readline()
        columns = content.split()

        if i>1 and len(columns)>1:

            Ez.append(float(columns[1]))
            t.append(float(columns[0]))

Ez=np.array(Ez) # in V/m
t=np.array(t)*1.0e-9   # in s

#close file
f.close()

#--- Plot electric field

fig1 = plt.figure(10, figsize=(6,4), dpi=200, tight_layout=True)
ax1=fig1.gca()
ax1.plot(t*1.0e9, Ez, lw=1.2, color='g', label='Ez CST')
ax1.set(title='Electric field at cavity center',
        xlabel='t [ns]',
        ylabel='$E [V/m]$',         #ylim=(-8.0e4,8.0e4)
        )
ax1.legend(loc='best')
ax1.grid(True, color='gray', linewidth=0.2)
plt.show()
```





Warp field extraction

Cube_cavity.py

1) Stores the Ez field (and other variables) in $(0,0,z_k)$ for all the timesteps in a list with $length = (nt \cdot nz)$

```
#------------#
#  time loop  #
#------------#

for n_step in range(tot_nsteps):
    picmi.warp.step()

    #Extracting the electric field from all processors
    #---z direction
    Ez=em.gatherez()    #3D matrix
    wEz=beam.wspecies.getez()   #vector with N components (for each particle) - in particles.py 1054 "Returns the Ex field applied to the particles"
    #print(wEz) is returning an empty vector
    #---x direction
    Ex=em.gatherex()    #3D matrix
    Bx=em.gatherbx()    #3D matrix
    wEx=beam.wspecies.getex()   #vector with N components (for each particle) - in particles.py 1054 "Returns the Ex field applied to the particles"
    #---y direction
    Ey=em.gatherey()    #3D matrix
    By=em.gatherby()    #3D matrix
    wEy=beam.wspecies.getey()   #vector with N components (for each particle) - in particles.py 1054 "Returns the Ex field applied to the particles"
    #get charge density
    rho=beam.wspecies.get_density()    #gathers the rho (electric density) of the beam rho[nx,ny,nz]

    #time vector
    t.append(picmi.warp.top.time) #t[0]=3.851666403092941e-12, t[300]=5.79675793665486e-10
    dt.append(picmi.warp.top.dt) #1.925833201546471e-12

    #append the 1D electric field at (ixtest,iytest,z)
    #--- z direction
    Ez_t.append(Ez[ixtest,iytest,:]) #1D vector of len=tot_nsteps*nz
    #--- x direction
    Ex_t.append(Ex[ixtest,iytest,:]) #1D vector of len=tot_nsteps*nz
    #--- y direction
    Ey_t.append(Ey[ixtest,iytest,:]) #1D vector of len=tot_nsteps*nz
    #append the 1D magnetic induction
    #--- x direction
    Bx_t.append(Bx[ixtest,iytest,:]) #1D vector of len=tot_nsteps*nz
    #--- y direction
    By_t.append(By[ixtest,iytest,:]) #1D vector of len=tot_nsteps*nz
    #append the charge density at (ixtest,iytest,z)
    rho_t.append(rho[ixtest, iytest, :]) #1D vector of len=tot_nsteps*nz
```

2) Saves the field in a dictionary for post processing and save it to a file with pickle

Warp field extraction

postproc.py

cst_to_dict.py | postproc.py | cube_cavity_impedance.py | cube_cavity.py | wake_potential_cst.py

```python
'''
File for postprocessing warp simulations

--- Reads the out file with pickle module
--- Plots the Electric field in the longitudinal direction
--- Obtains the frequency of the Electric field

'''

import numpy as np
from warp import picmi
import matplotlib.pyplot as plt
import time
import sys
import os
import scipy as sc
from copy import copy
import pickle as pk

#--- read the dictionary
with open('out_nt2000/out.txt', 'rb') as handle:
  data = pk.loads(handle.read())
  print('stored variables')
  print(data.keys())

#--- retrieve the variables

Ez_t=data.get('Ez')
Ex_t=data.get('Ex')
Ey_t=data.get('Ey')
Bx_t=data.get('Bx')
By_t=data.get('By')
rho_t=data.get('rho')
x=data.get('x')
y=data.get('y')
z=data.get('z')
t=data.get('t')
nt=data.get('nt')
nz=data.get('nz')
xtest=data.get('xtest')
ytest=data.get('ytest')

###################
#        Plots     #
###################

#--- Plot Electric field at cavity center

Ez=[]
Ez=np.reshape(Ez_t, (nz+1,nt))      #array to matrix (z,t)
Ex=np.reshape(Ex_t, (nz+1,nt))      #array to matrix (z,t)
Ey=np.reshape(Ex_t, (nz+1,nt))      #array to matrix (z,t)
t=np.array(t)
E_abs=np.sqrt(Ez[int(nz/2), :]**2+Ex[int(nz/2), :]**2+Ey[int(nz/2), :]**2)
```

3) postproc.py: reads the dict in ‘out.txt’ file with pickle module and plots the Electric field, frequency and charge distribution of the simulation performed with *cube_cavity.py*

Warp field extraction

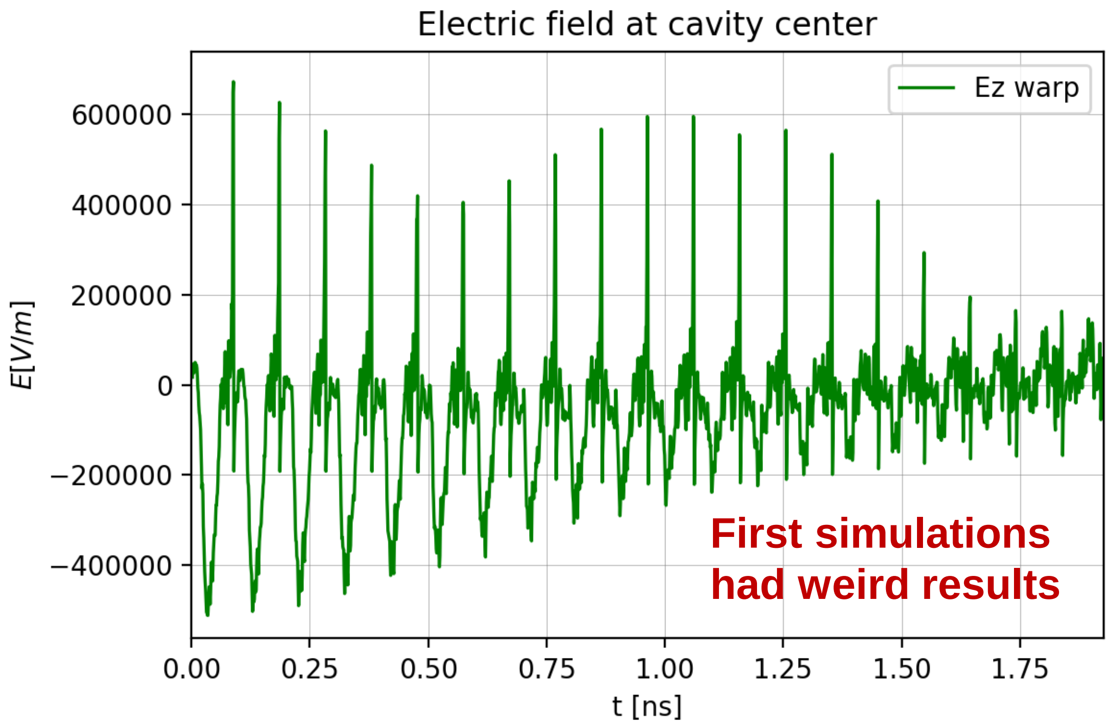
postproc.py

```
cst_to_dict.py    postproc.py    cube_cavity_impedance.py    cube_cavity.py    wake_potential_cst.py
```

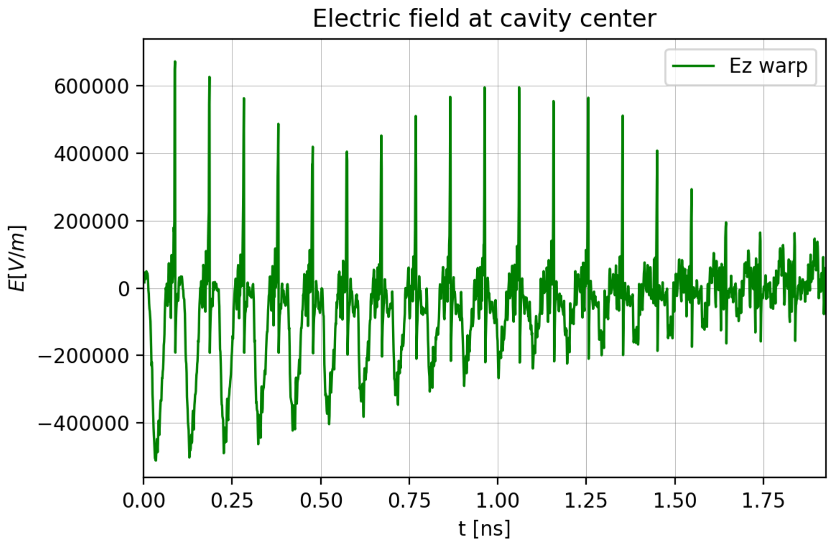
```python
'''
File for postprocessing warp simulations

--- Reads the out file with pickle module
--- Plots the Electric field in the longitudinal direction
--- Obtains the frequency of the Electric field

'''

import numpy as np
from warp import picmi
import matplotlib.pyplot as plt
import time
import sys
import os
import scipy as sc
from copy import copy
import pickle as pk

#--- read the dictionary
with open('out_nt2000/out.txt', 'rb') as handle:
    data = pk.loads(handle.read())
    print('stored variables')
    print(data.keys())

#--- retrieve the variables

Ez_t=data.get('Ez')
Ex_t=data.get('Ex')
Ey_t=data.get('Ey')
Bx_t=data.get('Bx')
By_t=data.get('By')
rho_t=data.get('rho')
x=data.get('x')
y=data.get('y')
z=data.get('z')
t=data.get('t')
nt=data.get('nt')
nz=data.get('nz')
xtest=data.get('xtest')
ytest=data.get('ytest')

##################
#      Plots     #
##################

#--- Plot Electric field at cavity center

Ez=[]
Ez=np.reshape(Ez_t, (nz+1,nt))      #array to matrix (z,t)
Ex=np.reshape(Ex_t, (nz+1,nt))      #array to matrix (z,t)
Ey=np.reshape(Ex_t, (nz+1,nt))      #array to matrix (z,t)
t=np.array(t)
E_abs=np.sqrt(Ez[int(nz/2), :]**2+Ex[int(nz/2), :]**2+Ey[int(nz/2), :]**2)
```

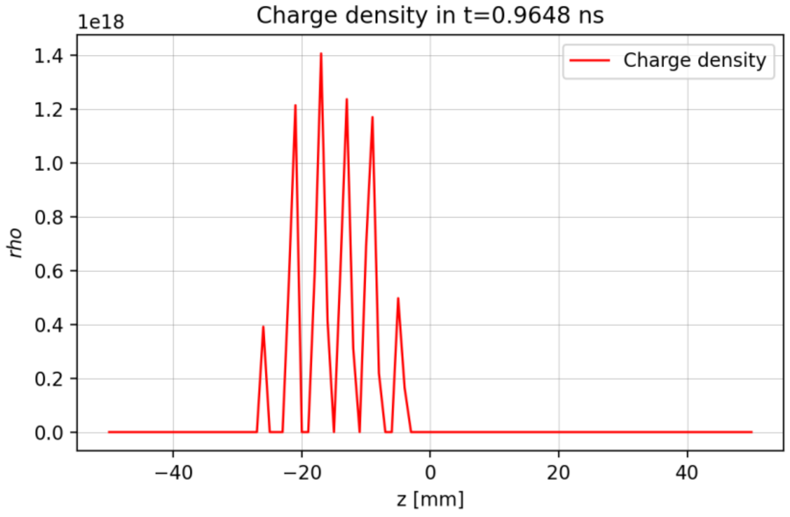
3) postproc.py: reads the dict in ‘out.txt’ file with pickle module and plots the Electric field, frequency and charge distribution of the simulation performed with *cube_cavity.py*

Also plots the comparison with CST if cst_to_dict.py has been run before

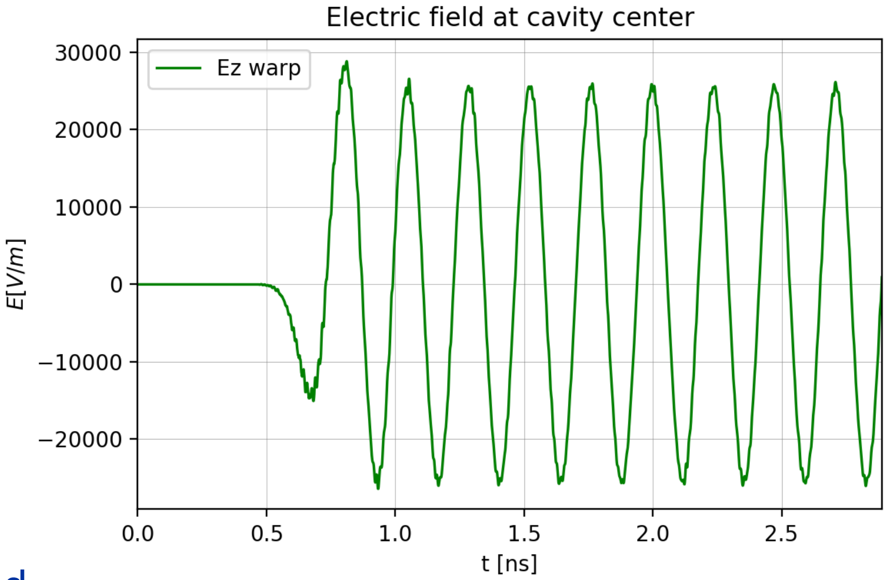


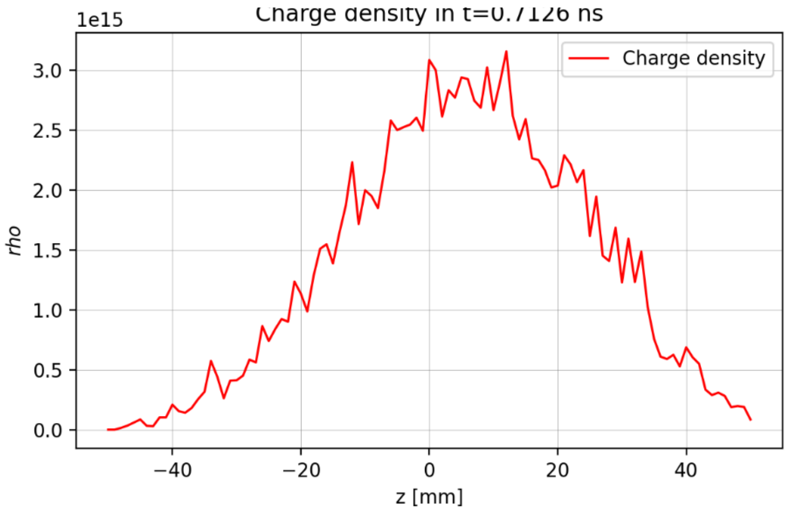
Warp fields fixed



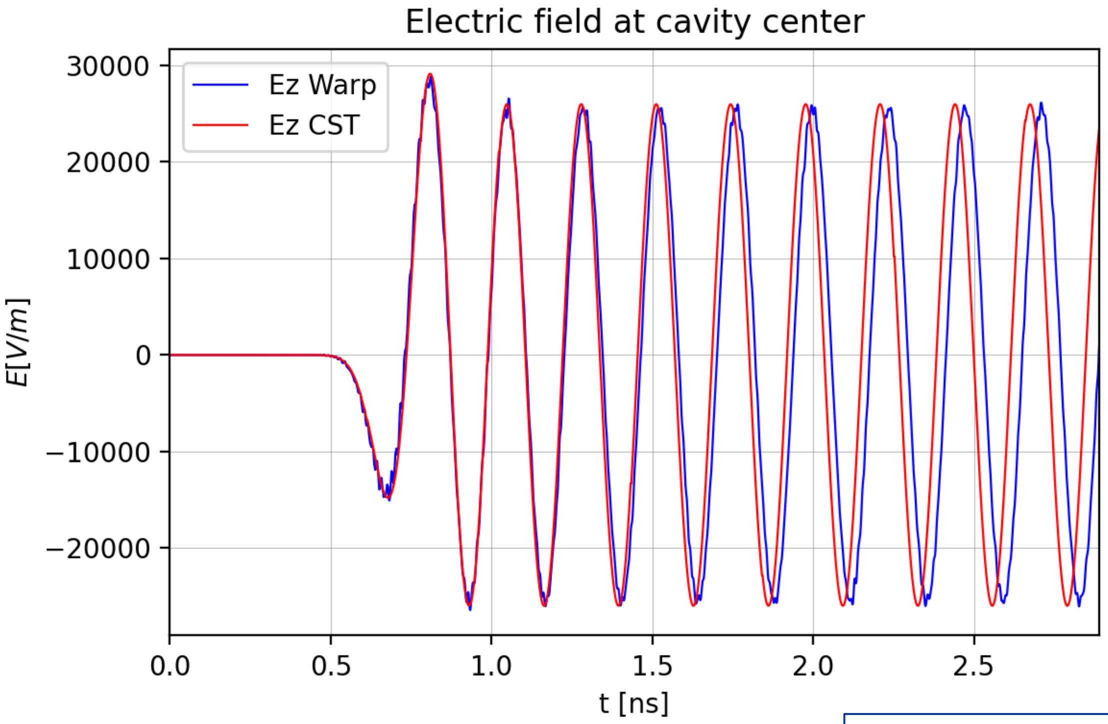


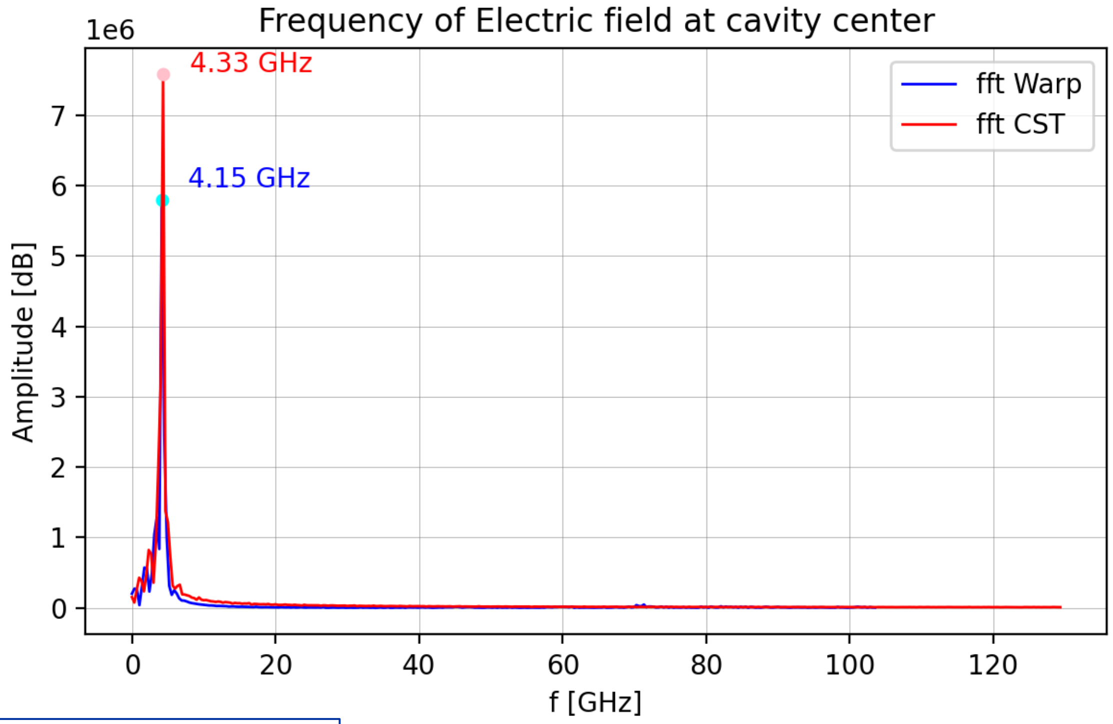
With the help of Lorenzo, we changed the **beam definition** (Macroparticles where not defined correctly) and the **time profile.** Now the results are as expected





Comparison with CST





The slight shift in frequency may be due to the Yee solver in Warp. Exploring the CKC solver may be interesting

Outline

1. Ez field comparison

2. Direct algorithm results

3. Indirect algorithm results

Conclusions and next steps

Direct algorithm

$$W_z(x,y,s) = -\frac{1}{q}\int_{-\infty}^{\infty} E_z(x,y,z,t=(z+s)/c)\mathrm{d}z$$

Integration in z over the whole domain (z_{min}, z_{max})

1) Define Wakelength and s vector

```
#--- set Wake_length, s
Wake_length=nt*dt*c - (zmax-zmin) - init_time*c
print('Max simulated time = '+str(round(t[-1]*1.0e9,4))+' ns')
print('Wake_length = '+str(Wake_length*1e3)+' mm')
ns_neg=int(init_time/dt)        #obtains the length of the negative part of s
ns_pos=int(Wake_length/(dt*c))  #obtains the length of the positive part of s
s=np.linspace(-init_time*c, 0, ns_neg) #sets the values for negative s
s=np.append(s, np.linspace(0, Wake_length,  ns_pos))
```

The negative values of s are related to the initial time in with the beam is injected (*init_time*)
The positive values discretize the wake length with a resolution related to the timestep size: $dt * c$

2) Define Wakelength and s vector

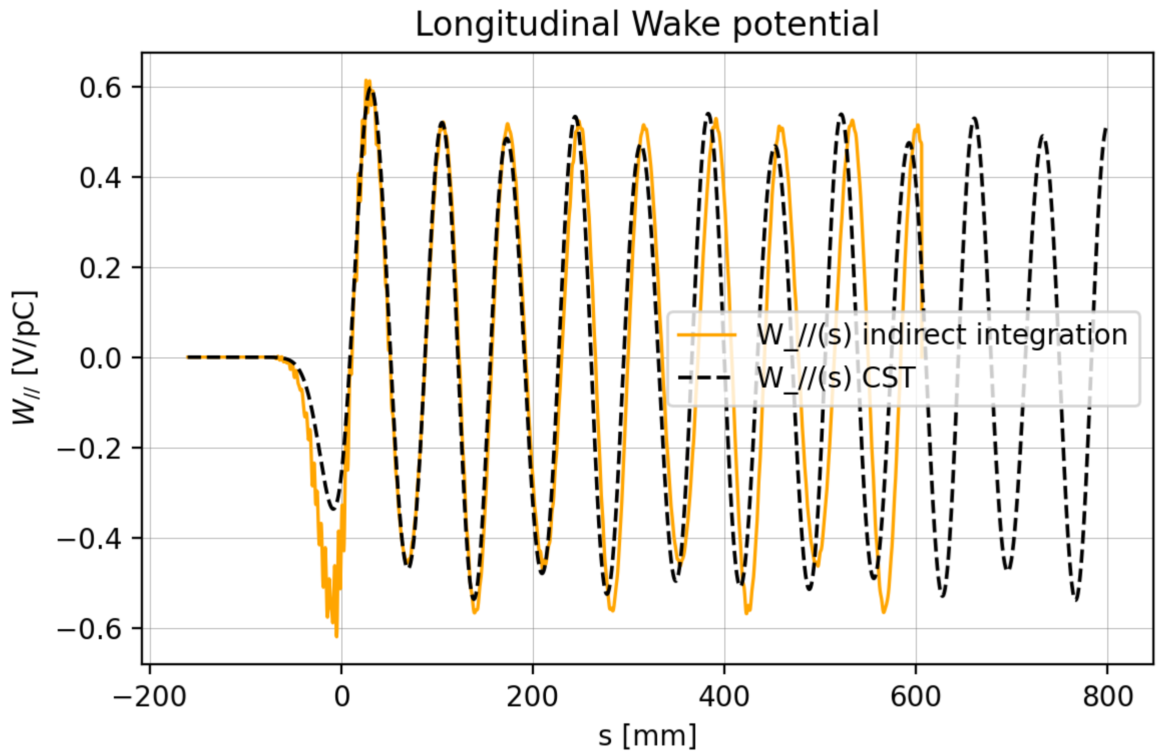
```
#------------------------#
#      Obtain W||(s)     #
#------------------------#

# s loop -------------------------------------#

for n in range(len(s)-1):


    #-------------------------------#
    # integral between zmin and zmax #
    #-------------------------------#

    #integral of (Ez(xtest, ytest, z, t=(s+z)/c))dz
    #E - the correct integral is only obtained when integrating the field in the cavity

    k=0
    for k in range(0, nt):
        t_s[k,n]=(z_interp[k]+s[n])/c-zmin/c-t[0]+init_time
        #compute integral
        if t_s[k,n]>0.0:
            it=int(t_s[k,n]/dt)     #find index for t
            Wake_potential[n]=Wake_potential[n]+(Ez_interp[k, it])*dz_interp

q=(1e-9)*1e12                       # charge of the particle beam in pC
Wake_potential=Wake_potential/q     # [V/pC]
```

2) Integral (z_{min}, z_{max}) is very close to Integral (l_1, l_2). The only differences observed are in the calculation of the *k* loss factor

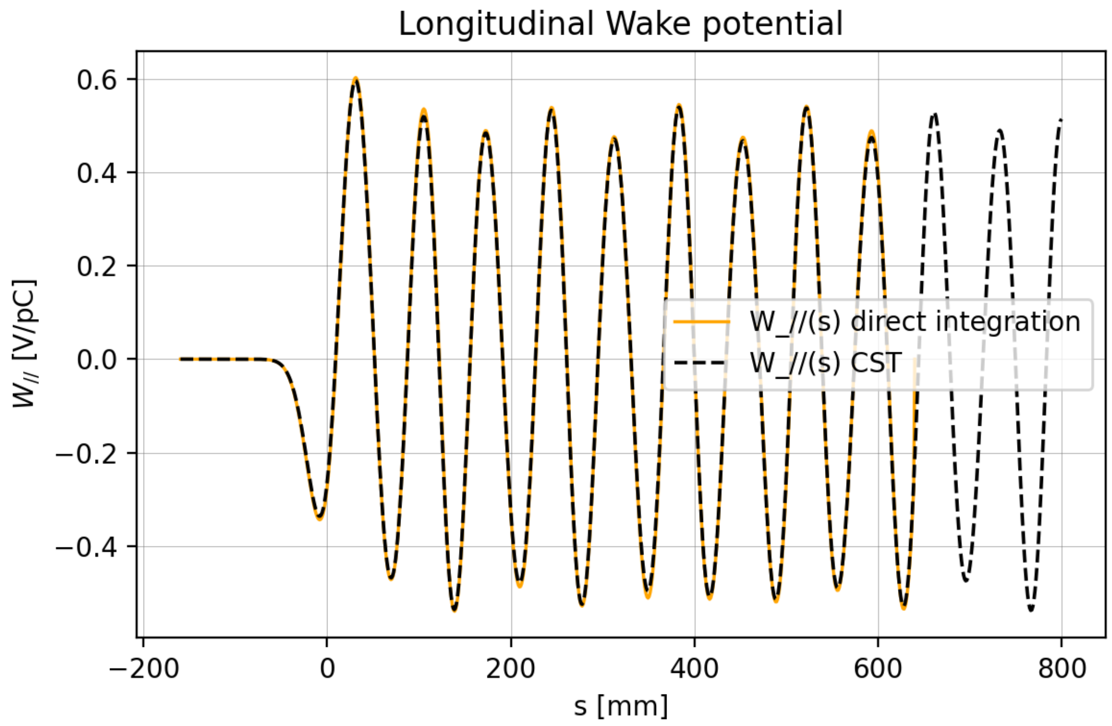
$$k = -\int_{-\infty}^{\infty} \lambda(s) W_{\|}(s)\, ds$$

Direct algorithm

Direct_wake_potential.py : uses the Ez field from Warp

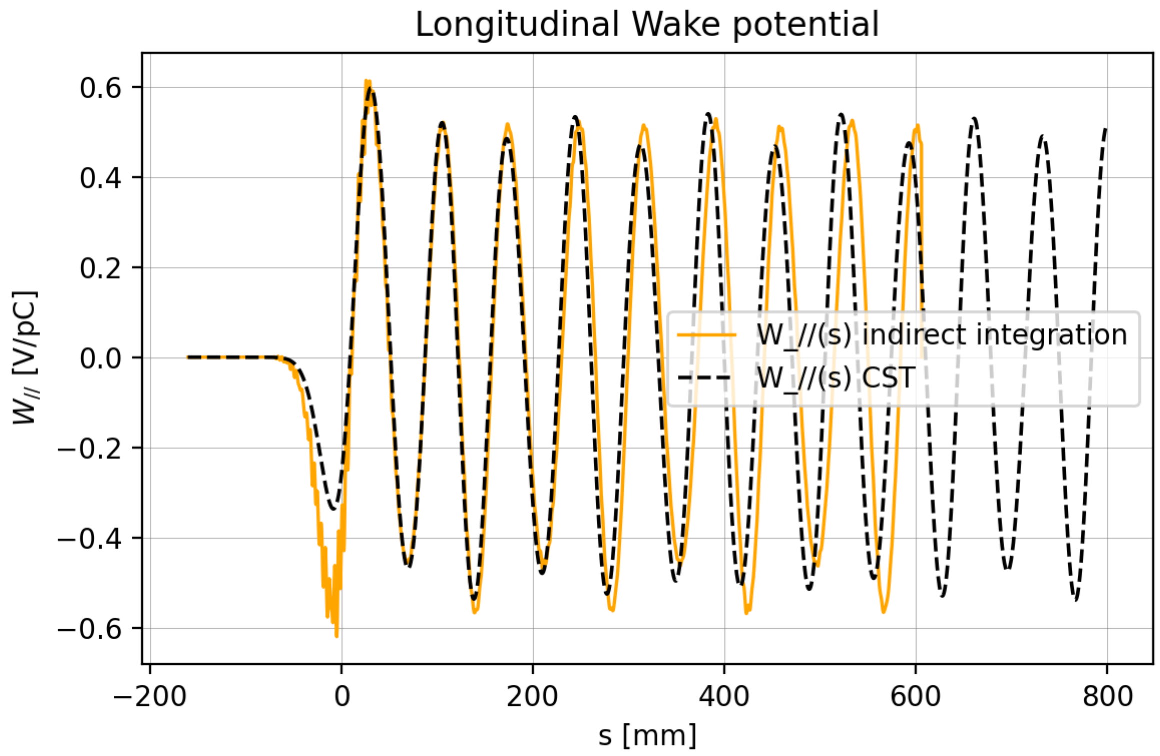


Direct_wake_potential_cst.py : used the Ez field from CST

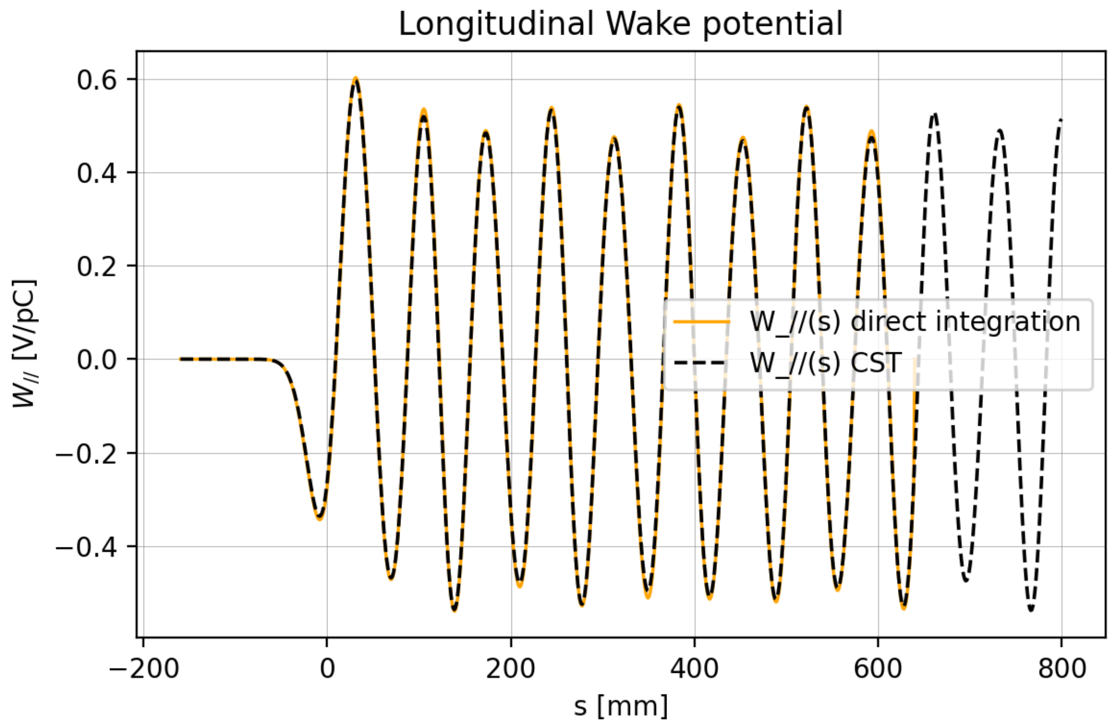


Direct algorithm

Direct_wake_potential.py : uses the Ez field from Warp



Direct_wake_potential_cst.py : used the Ez field from CST



Impedance and loss factor

3) The charge density is obtained from the σ_z and beam charge q with a Gaussian profile

```
#obtain charge distribution with a gaussian profile
charge_dist=(q*1e-12)*(1/(sigmaz*np.sqrt(2*np.pi)))*np.exp(-(0.5*(s-0.0)**2.)/(sigmaz)**2.)
#charge distribution [pC/m]
charge_dist_norm=charge_dist/(q*1e-12)
 #normalized charge distribution [-]
```

4) Perform the integral of the loss factor k

$$k = -\int_{-\infty}^{\infty} \lambda(s) W_{\|}(s)\, ds$$

```
#perform the integral int{-inf,inf}(-lambda*Wake_potential*ds)
for n in range(len(s)):
    k_factor=k_factor+charge_dist_norm[n]*Wake_potential[n]*ds

k_factor=-k_factor # [V/pC]
print('calculated k_factor = '+str(format(k_factor, '.3e')) + ' [V/pC]')
```

5) Obtain the impedance with *numpy.fft.fft()*

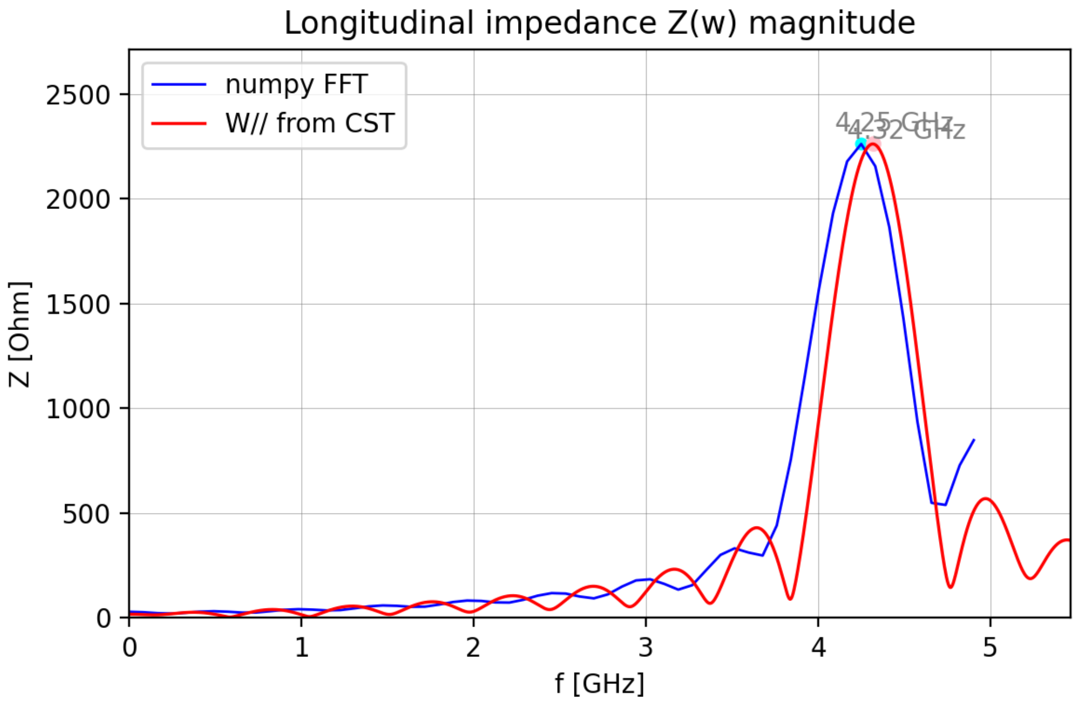
$$Z_{\|}(\omega) = -\frac{\int_{-\infty}^{\infty} W_{\|}(s)\, e^{-i\omega s}\, ds}{\int_{-\infty}^{\infty} \lambda(s)\, e^{-i\omega s}\, ds}$$

```
#------------------------------#
#      Obtain impedance Z||      #
#------------------------------#

#--- Obtain impedance Z with Fourier transform numpy.fft.fft

# to increase the resolution of fft, a longer wake length is needed
f_max=5.0*1e9
t_sample=int(1/(ds/c)/2/f_max) #obtains the time window to sample the time domain data
N_samples=int(len(s)/t_sample)
print('Performing FFT with '+str(N_samples)+' samples')
print('Frequency bin resolution '+str(round(1/(len(s)*ds/c)*1e-9,2))+ ' GHz')
print('Frequency range: 0 - '+str(round(f_max*1e-9,2)) +' GHz')

# Padding woth zeros to increase N samples = smoother FFT
charge_dist_padded=np.append(charge_dist, np.zeros(10000))
Wake_potential_padded=np.append(Wake_potential, np.zeros(10000))
charge_dist_fft=abs(np.fft.fft(charge_dist_padded[0:-1:t_sample]))
Wake_potential_fft=abs(np.fft.fft(Wake_potential_padded[0:-1:t_sample]))
Z_freq = np.fft.fftfreq(len(Wake_potential_padded[:-1:t_sample]), ds/c*t_sample)*1e-9
Z = abs(- Wake_potential_fft / charge_dist_fft)
```

Impedance

Direct_wake_potential.py : uses the Ez field from Warp

$$k_{factor} = 1.776e-01\ [\frac{V}{pC}]$$

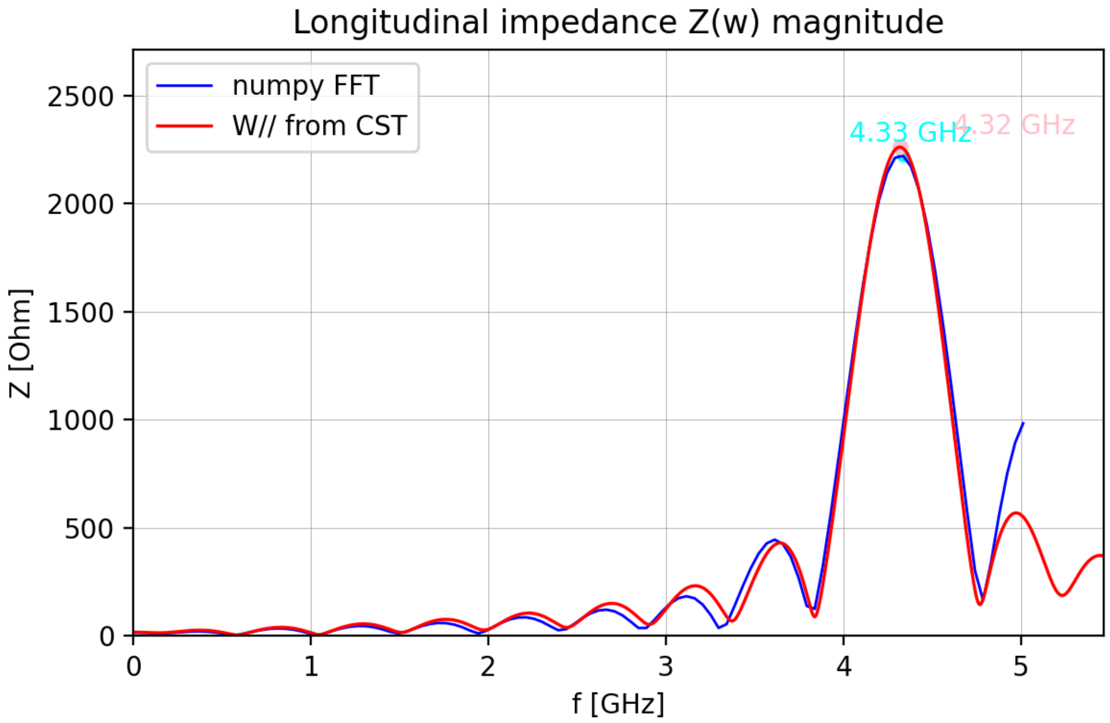


Direct_wake_potential_cst.py : used the Ez field from CST

$$k_{factor} = 5.740e-02\ [\frac{V}{pC}]$$

value from CST: 5.790052e-02

Difference <1%



Impedance with CST's DFT

Using a 1000 simples in frequency DFT

A single sided DFT with 1000 frequency samples is used for the numerical fourier transformation:

$$S(\omega) = \frac{\Delta t}{\sqrt{\pi}} \sum_{k=0}^{N} s(k) \exp(-jk\Delta t\, \omega)$$

with:

dt: time sampling width.

N: number of time samples

w: angular frequency (1000 samples)

Source: CST's wake solver manual

```
#-------------------------------#
#      Obtain impedance Z||      #
#-------------------------------#

#--- DFT function definition like in CST [not working]

class Fourier:
    def __init__(self, dft, freqs):
        self.dft = dft
        self.freqs = freqs

def DFT(F, dt, N):
        #function to obtain the DFT with 1000 samples
        #--F: function in time domain
        #--dt: time sampling width
        #--N: number of time samples

        #define frequency domain
        N_samples=1000  # same number as CST
        f_max = 5.0     # maximum freq in GHz
        freqs=np.linspace(-f_max,f_max,N_samples)*1e9 #frequency range [Hz]
        dft=np.zeros_like(freqs)*1j
        padding=1     #length of the padding with zero
        F=np.append(F,np.zeros(padding))
        print('Performing DFT with '+str(N_samples)+'samples')
        print('Frequency bin resolution'+str(round(1/(N*dt)*1e-9,3))+ 'GHz')
        print('Frequency range')

        for m in range(N_samples):
            for k in range(N+padding):
                dft[m]=dft[m]+F[k]*np.exp(-1j*k*dt*freqs[m])

        dft=dt/np.sqrt(np.pi)*dft #Magnitude in [Ohm]
        freqs=freqs*1e-9 #in [GHz]
        return Fourier(dft,freqs)
```

Outline

1. Ez field comparison

2. Direct algorithm results

3. Indirect algorithm results

Conclusions and next steps

Indirect algorithm

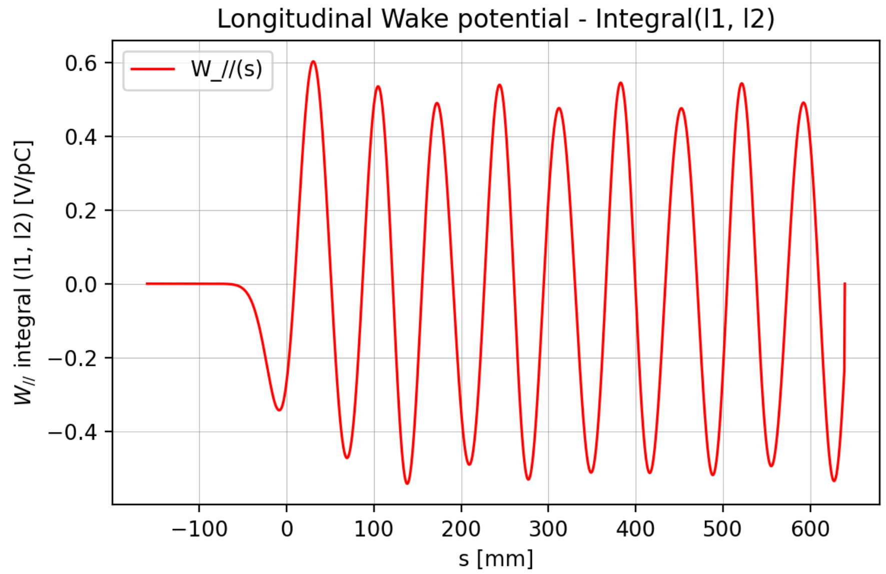
$$qW_z(x,y,s) = \\ = \left[\varphi(x,y=b_1) - \varphi(x,y)\right]_{z=-l_1} \\ - \int_{-l_1}^{l_2} E_z(x,y,z,t=(z+s)/c)dz \quad (15) \\ + \left[\varphi(x,y) - \varphi(x,y=b_2)\right]_{z=l_2} .$$

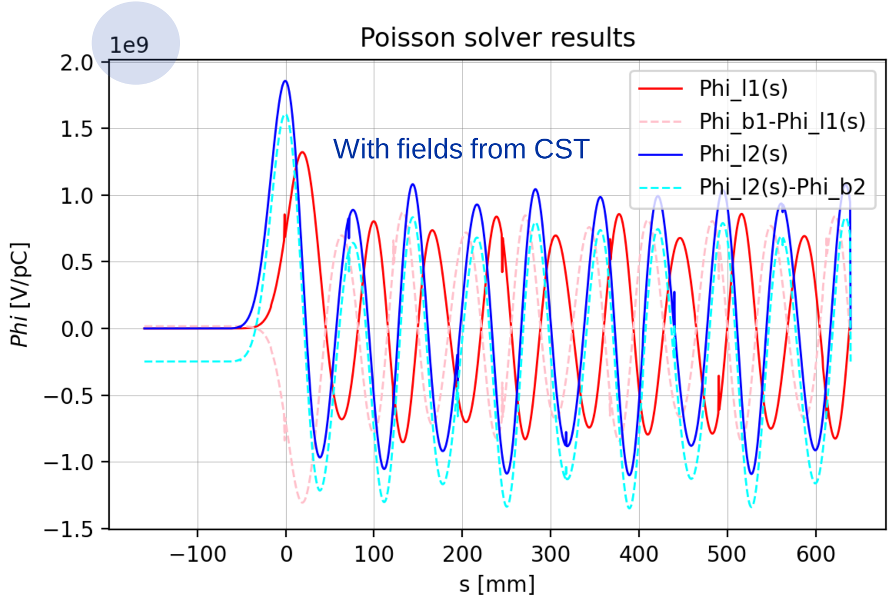
Poisson in z=-l1 surface t=(-l1+s)/c

Poisson in z=l2 surface t=(l2+s)/c

The dominant part seem to be the **poisson phi φ.** According to the results of the direct integration, $\boldsymbol{\varphi_{l1}}$ **should cancel with $\boldsymbol{\varphi_{l2}}$** ...

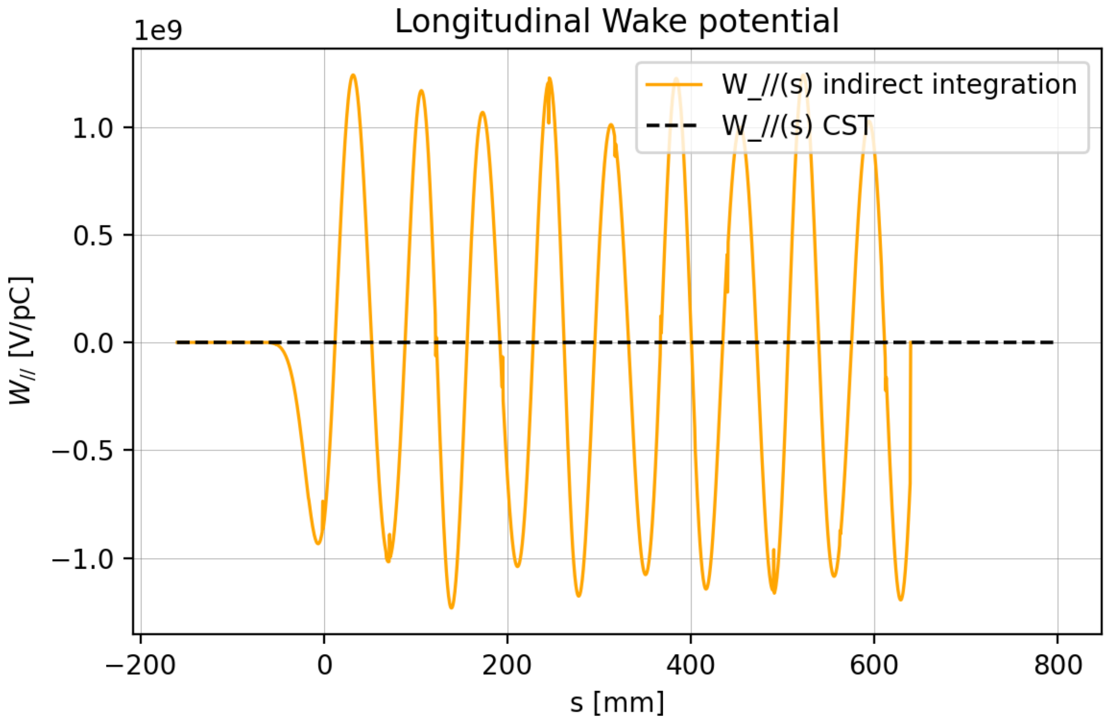
A **normalization factor** might be missing since $\boldsymbol{\varphi}$ is in the order of 1e9

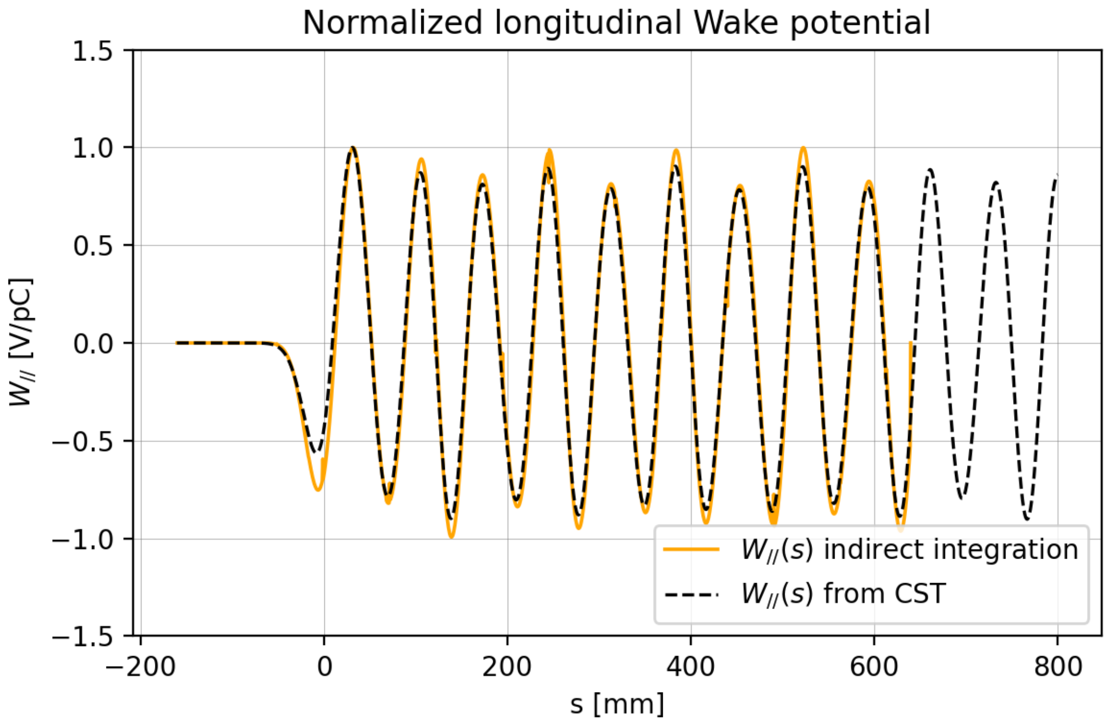




Indirect algorithm results (CST)

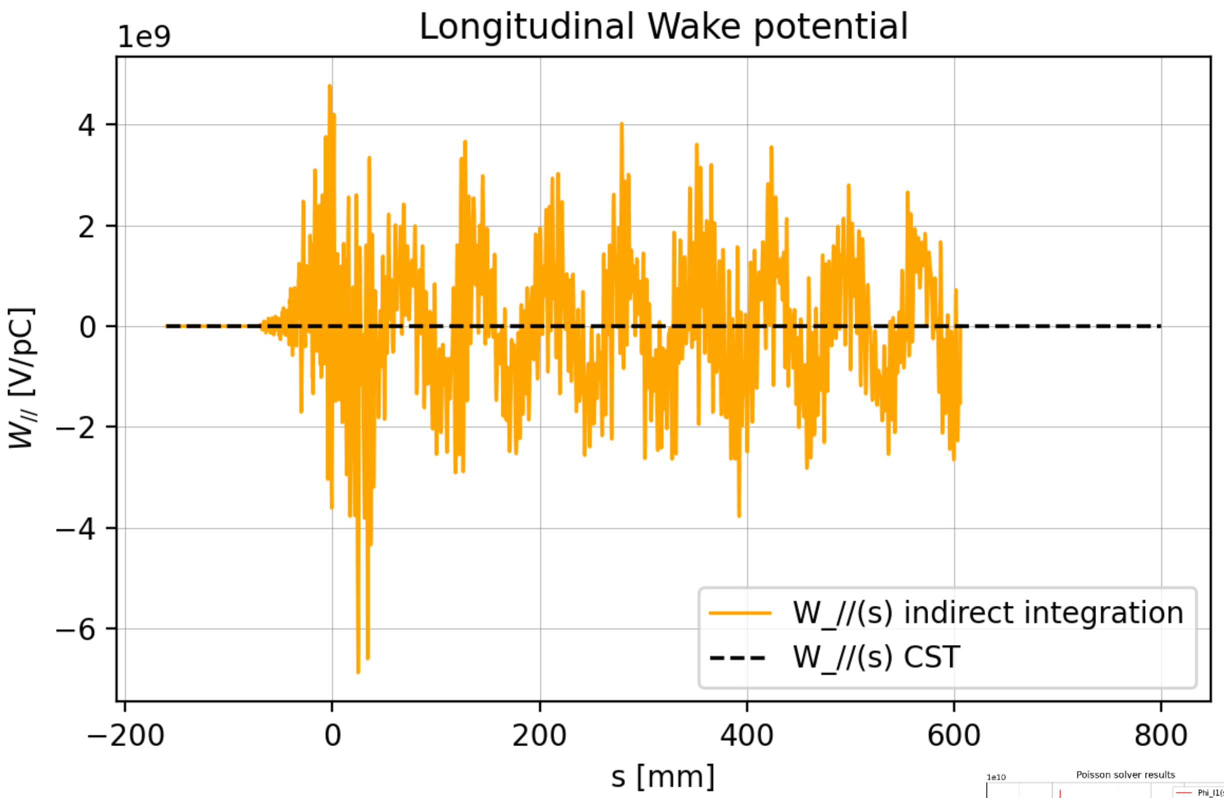
Indirect_wake_potential_cst.py : uses the Ez field from CST

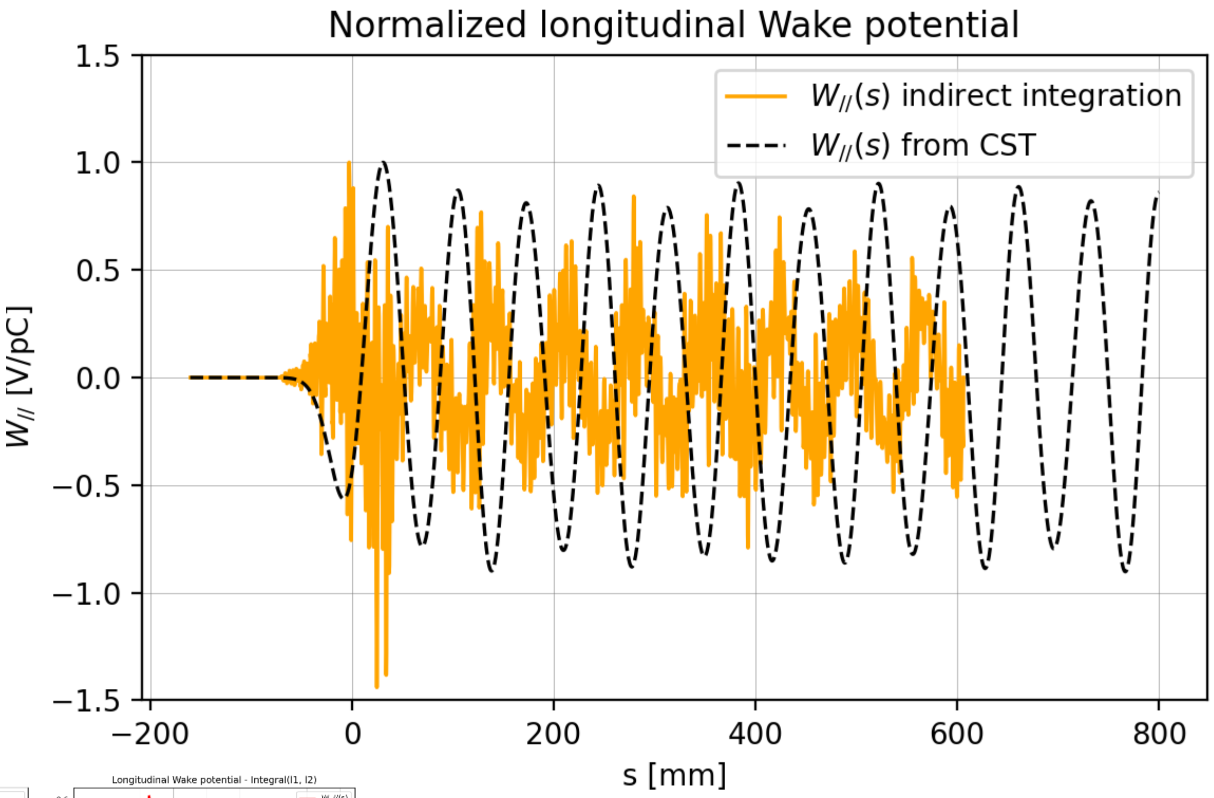


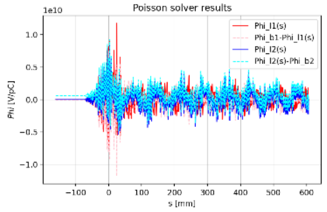


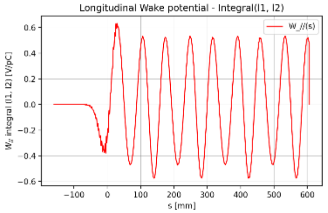
Indirect algorithm results (Warp)

Indirect_wake_potential.py : uses the Ez field from Warp



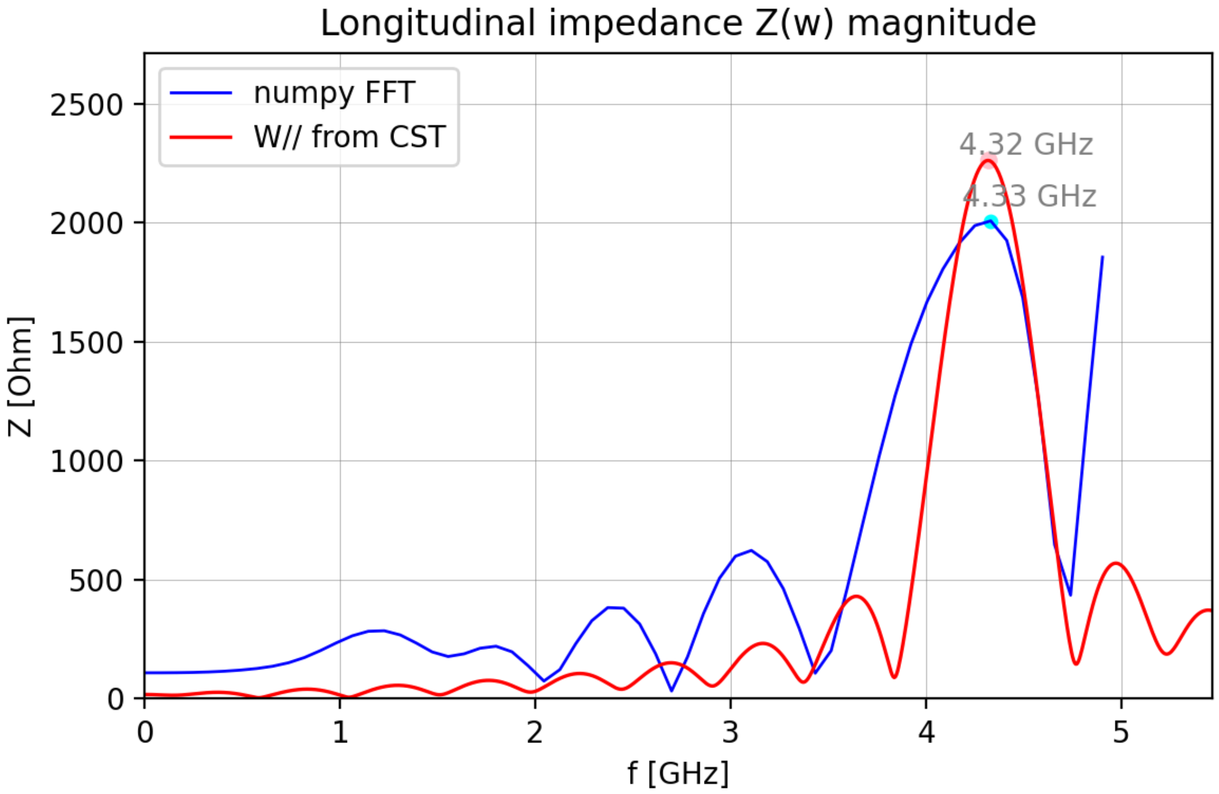




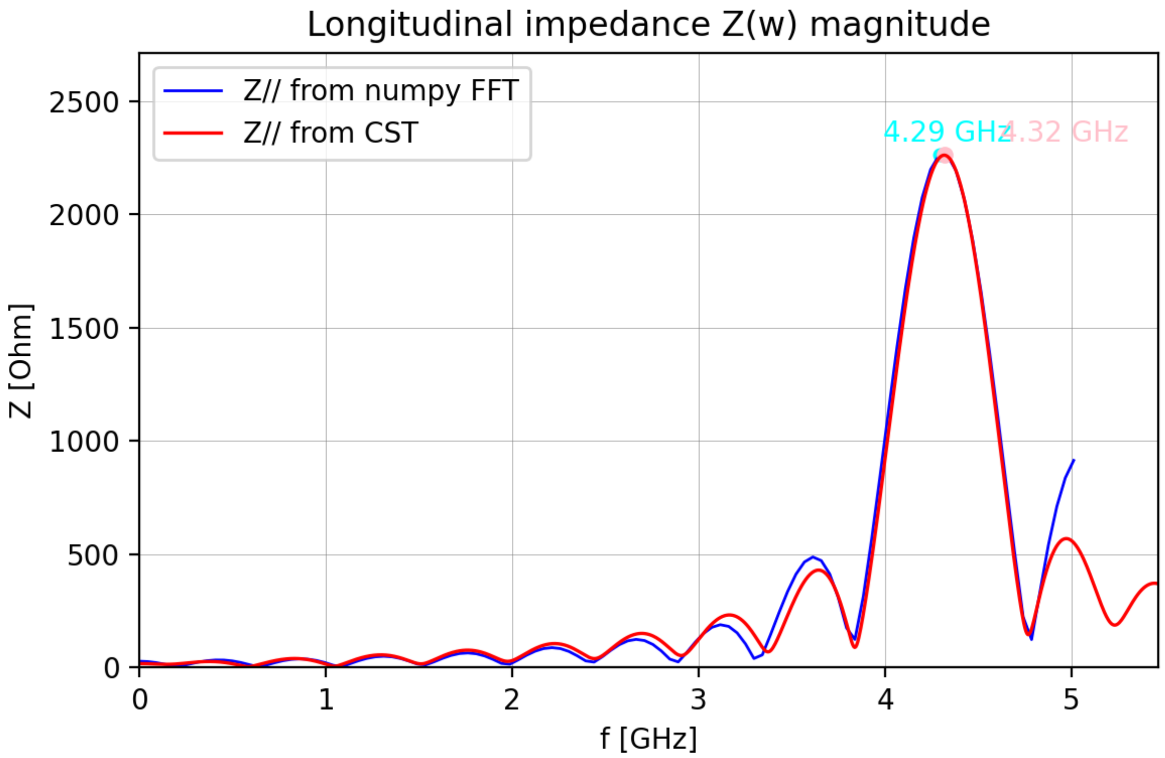


Impedance results

Indirect_wake_potential.py : uses the Ez field from Warp



Indirect_wake_potential.py : uses the Ez field from Warp



*Impedance amplitude is normalized for the indirect algorithm

Outline

1. Ez field comparison

2. Direct algorithm results

3. Indirect algorithm results

<u>Conclusions and next steps</u>

Next steps?

- Fix the poisson results from the indirect algorithm. Compare the fields from Warp and CST in different locations
- Compare the indirect integration from CST with the indirect algorithm
- Try the other CKC solver in Warp
- Perform a convergence analysis?
- Continue with the transverse Wake potential and impedance through Panofsky-Wenzel
- Try the EMcLAW cube cavity? (--> long term…)

CERN