Beam dynamics simulations with the PyHEADTAIL and PyECLYOD macroparticle codes

Michael Schenk
CERN & EPFL (Switzerland)

Largely based on presentations by K. Li & G. Iadarola

Further acknowledgements
Computational physics lies somewhere between experimental and theoretical physics and has many advantages!

**Motivation for numerical simulation codes**

### Analytical studies
- Very reduced system
- Immediate global behaviour of the system / scaling

### Machine studies
- Full real system
- Reduced flexibility for studies
- Reduced diagnostics

### Numerical studies
- Intermediate system
- Full flexibility for studies
- Full diagnostics
Outline

• Part I: PyHEADTAIL
  • Introduction
  • PyHEADTAIL protocol
  • How to build a simulation
  • Details on the model with illustrations
  • Practical information

• Part II: PyECLoud
  • Introduction to electron cloud and PyECLoud
  • Build-up simulations
  • Beam dynamics simulations with PyECLoud & PyHEADTAIL
  • Practical information
Outline

• Part I: PyHEADTAIL
  • Introduction
  • PyHEADTAIL protocol
  • How to build a simulation
  • Details on the model with illustrations
  • Practical information

• Part II: PyECLOUD
  • Introduction to electron cloud and PyECLOUD
  • Build-up simulations
  • Beam dynamics simulations with PyECLOUD & PyHEADTAIL
  • Practical information
The story

Originally the „HEADTAIL“ project

- Macroparticle simulation code to study collective effects in circular accelerators
- Initiated in 2000 by G. Rumolo at CERN
- Written in C/C++ language
- Originally used to study electron cloud effects
- Impedance effects were added later
- Model proved to be successful in various studies
- 7 publications in peer-reviewed journals (5 PRST-AB + 2 PRL)

The code grew vastly over the years and became more and more difficult to maintain and extend ...

Beginning of 2014: move to Python and implement the HEADTAIL model in object-oriented manner
PyHEADTAIL is a 6D macroparticle tracking code to study collective effects and mitigation techniques in circular accelerators

- Impedance-driven instabilities
  Head-tail modes, Transverse Mode Coupling Instability (TMCI), coupled-bunch instabilities
- Space-charge
- Electron cloud instabilities
- Mitigation systems: Feedbacks, Landau damping (incoherent tune spreads), ...
  ...

M. Schenk et al. 05.07.2017
Outline

• Part I: PyHEADTAIL
  • Introduction
  • PyHEADTAIL protocol
  • How to build a simulation
  • Details on the model with illustrations
  • Practical information

• Part II: PyECLOUD
  • Introduction to electron cloud and PyECLOUD
  • Build-up simulations
  • Beam dynamics simulations with PyECLOUD & PyHEADTAIL
  • Practical information
PyHEADTAIL protocol

Simple
- Easy to fix
- Easy to read
- Easy to maintain

Modular
- Easy to extend
- Easy to combine
- Easy to maintain

Dynamic
- Fast
- Flexible
- Interactive

... and each individual module should become a carefully engineered piece of software
Simple

Be minimalist
keep number of lines to the minimum necessary

Be pragmatic
use available libraries and do not reinvent the wheel

Be paranoid
write clear and well-formatted code

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live”

John Woods
PyHEADTAIL repository
https://github.com/PyCOMPLETE/PyHEADTAIL

feedback
gpu
impedances
machines
monitors
multipoles
particles
aperture
spacecharge
testing
trackers
...
Outline

• Part I: PyHEADTAIL
  • Introduction
  • PyHEADTAIL protocol
  • How to build a simulation
  • Details on the model with illustrations
  • Practical information

• Part II: PyECLOUD
  • Introduction to electron cloud and PyECLOUD
  • Build-up simulations
  • Beam dynamics simulations with PyECLOUD & PyHEADTAIL
  • Practical information
Building a simulation in PyHEADTAIL

With the modular design it becomes easy to make customised simulations.
Building a simulation in PyHEADTAIL
Building a simulation in PyHEADTAIL

- Import modules
- Choice of parameters
- Instantiate objects
- Assemble ring
- Tracking loop
Building a simulation in PyHEADTAIL

Import modules
Choice of parameters
Instantiate objects
Assemble ring
Tracking loop

```
from PyHEADTAIL.trackers.transverse_tracking import TransverseMap
from PyHEADTAIL.trackers.detuners import Chromaticity, AmplitudeDetuning
from PyHEADTAIL.trackers.simple_long_tracking import RFSystems
from PyHEADTAIL.particles.slicing import UniformBinSlicer
from PyHEADTAIL.impedances.wakes import CircularResonator, WakeField
from PyHEADTAIL.feedback.transverse_damper import TransverseDamper
```
Building a simulation in PyHEADTAIL

Import modules
Choice of parameters
Instantiate objects
Assemble ring
Tracking loop

```
43 R = C/(2*np.pi)
44 T0 = C/(beta*c)
45 omega0 = 2*np.pi/T0
46 alpha = 53.86**-2
47 eta = alpha - gamma**-2
48
49 V_RF = [16e6, 0*8e6]
50 h_RF = [35640, 71280]
51 dphi_RF = [0, 0*np.pi]
52
53 macroparticle_number = 50000
54 intensity = 1.3e11
55 epsn_x = 2.2e-6
56 epsn_y = 2.2e-6
57 epsn_z = 2.5
58
59 Q_x = 62.31
60 Q_y = 60.32
61 Qp_x = 0
62 Qp_y = 0
63 beta_x = R/Q_x
64 beta_y = R/Q_y
```
import modules

Choice of parameters

Instantiate objects

Assemble ring

Tracking loop

```python
rfbucket = RFBucket(
circumference=C, charge=e, mass=m_p, gamma=gamma,
alpha_array=[alpha], p_increment=0,
harmonic_list=h_RF, voltage_list=V_RF, phi_offset_list=phi_RF
)
bunch = ParticleGenerator(
macroparticleNumber=macroparticleNumber, intensity=intensity,
charge=e, mass=m_p, gamma=gamma, circumference=C,
distribution_x=gaussian2D(epsn_x/betagamma), beta_x=beta_x,
distribution_y=gaussian2D(epsn_y/betagamma), beta_y=beta_y,
distribution_z=RF_bucket_distribution(rfbucket, epsn_z=epsn_z))
bunch.x += 3.5e-4

# TRANSVERSE MAP
# =============

n_segments = 3

g = np.array([i * C/n_segments for i in range(n_segments + 1)])
alpha_x = np.zeros(n_segments + 1)
```
Building a simulation in PyHEADTAIL

114 # CREATE WAKEFIELDS
115 # ==============  
116 slicer_for_wakefields = UniformBinSlicer(300, n_sigma_z=3)  
117 wake = CircularResonator(R_shunt=1e6, frequency=700e6, Q=350,  
118 wakefields = WakeField(slicer_for_wakefields, wake, circumfere  
119  
120 121 # CREATE DAMPER  
122 # ==============  
123 damping_rate_x = 50  
124 damping_rate_y = 50  
125 damper = TransverseDamper(damping_rate_x, damping_rate_y)  
126
Building a simulation in PyHEADTAIL

Import modules

Choice of parameters

Instantiate objects

Assemble ring

Tracking loop

```
print '\n--> Begin tracking...\n'

for i in range(n_turns):
    t0 = time.clock()
    for m in one_turn_map:
        m.track(bunch)
    bunchmonitor.dump(bunch)

    if not monitorswitch:
        if (bunch.mean_x() > 1e3 or bunch.mean_y() > 1e3)
            print "--> Activate monitor"
            monitorswitch = True
    else:
        if s_cnt < 8192:
            slicemonitor.dump(bunch)
        s_cnt += 1
```
Outline

• Part I: PyHEADTAIL
  • Introduction
  • PyHEADTAIL protocol
  • How to build a simulation
  • Details on the model with illustrations
  • Practical information

• Part II: PyECLOUD
  • Introduction to electron cloud and PyECLOUD
  • Build-up simulations
  • Beam dynamics simulations with PyECLOUD & PyHEADTAIL
  • Practical information
What does the model include?

• There is a set of important dynamics that influence beam stability.

• The PyHEADTAIL model must include these mechanisms / dynamics to reproduce machine observations
  • Transverse (betatron) and longitudinal (synchrotron) motion
  • Effect of wake fields / impedances
  • Chromatiticy, detuning with amplitude, incoherent tune spreads
  • Space charge, mainly important at lower beam energies
  • Multi-bunch beams for coupled-bunch instabilities
  • Electron cloud build-up and interaction with the beam *(see part II on PyELOUD)*
  • Data acquisition / monitoring tools
  • ...

• At the same time the code must be fast enough to allow for tracking of millions of particles over several hundred thousands of turns ...
**PyHEADTAIL model – overview**

1. **Divide ring into segments separated by interaction points (IP)**
2. **Initialise beam**
   - Typically $10^6$ macroparticles
   - Various distributions
3. **Linear periodic maps for transverse tracking** from one IP to the next
4. **Adding chromaticity and amplitude detuning**
5. **(Non-)linear synchrotron motion**
6. **(Collective) effects**
   - Wake field kicks
   - Electron cloud
   - Space charge
   - Feedback system
   - Multipole kicks
     (e.g. skew quadrupole to model linear coupling)
7. **Data acquisition**
   - Bunch data
   - Slice data
   - Particle data
8. **100'000s turns**

---

**M. Schenk et al.**

05.07.2017
PyHEADTAIL model – overview

1. Divide ring into segments separated by interaction points (IP)
2. Initialise beam
   - Typically $10^6$ macroparticles
   - Various distributions
3. Linear periodic maps for transverse tracking from one IP to the next
4. Adding chromaticity and amplitude detuning
5. (Non-)linear synchrotron motion
6. (Collective) effects
   - Wake field kicks
   - Electron cloud
   - Space charge
   - Feedback system
   - Multipole kicks (e.g. skew quadrupole to model linear coupling)
7. Data acquisition
   - Bunch data
   - Slice data
   - Particle data
8. 100’000s turns ...

- Initialising the beam typically involves $10^6$ macroparticles with various distributions.
- Linear periodic maps are used for transverse tracking from one interaction point (IP) to the next.
- Adding chromaticity and amplitude detuning is a crucial step.
- Non-linear synchrotron motion affects the particle motion.
- Collective effects include wake field kicks, electron clouds, space charge, feedback systems, and multipole kicks.
- Data acquisition includes bunch data, slice data, and particle data.

Graphs show bunch centroids and head-tail amplitudes as functions of time and position.
PyHEADTAIL model – overview

1. Divide ring into segments separated by interaction points (IP)

2. Initialise beam
   - Typically $10^6$ macroparticles
   - Various distributions

3. Linear periodic maps for transverse tracking from one IP to the next

4. Adding chromaticity and amplitude detuning

5. (Non-)linear synchrotron motion

6. (Collective) effects
   - Wake field kicks
   - Electron cloud
   - Space charge
   - Feedback system
   - Multipole kicks (e.g. skew quadrupole to model linear coupling)
   - ...

7. Data acquisition
   - Bunch data
   - Slice data
   - Particle data

8. 100'000s turns ...

---

Bunch centroid [µm]

Head-tail ampl. [a.u.]

---

M. Schenk et al.

05.07.2017
Beam representation

- PyHEADTAIL uses macroparticles to describe the particle beams.
Beam representation

- PyHEADTAIL uses macroparticles to describe the particle beams.
- A macroparticle is a numerical representation of a clustered collection of physical particles.
- Such a model allows a seamless mapping of the realistic system into the computational environment, and it is fairly easy to implement.
- Using macroparticles increases the granularity of the numerical system (noise), but allows to treat large systems within limitations of computational resources.

Particle beam in PyHEADTAIL

\[
\begin{align*}
\text{transverse} & : \begin{pmatrix} x_i \\ x_i' \\ y_i \\ y_i' \end{pmatrix} & \begin{pmatrix} q_i \\ m_i \end{pmatrix} \\
\text{longitudinal} & : \begin{pmatrix} z_i \\ \delta_i \end{pmatrix}
\end{align*}
\]

Each macroparticle has 4 transverse and 2 longitudinal coordinates, an electric charge, and a mass.
Beam initialisation

Various particle distributions are available and/or being implemented

- Transverse
  - Gaussian
  - Kapchinsky-Vladimirsky (KV)

- Longitudinal
  - Gaussian with linear bucket
  - Non-linear RF bucket with matched Gaussian, parabolic, waterbag distribution
  - Multi-harmonic RF buckets

- Import from file

List can be extended further ...

*Parabolic, single RF*
PyHEADTAIL model – overview

1. Divide ring into segments separated by interaction points (IP)
2. Initialise beam
   - Typically $10^6$ macroparticles
   - Various distributions
3. Linear periodic maps for transverse tracking from one IP to the next
4. Adding chromaticity and amplitude detuning
5. (Non-)linear synchrotron motion
6. (Collective) effects
   - Wake field kicks
   - Electron cloud
   - Space charge
   - Feedback system
   - Multipole kicks (e.g. skew quadrupole to model linear coupling)
7. Data acquisition
   - Bunch data
   - Slice data
   - Particle data
8. 100'000s turns...

M. Schenk et al.
• **Dipoles**: define orbit  
• **Quadrupoles**: define focusing  
• Transverse single particle dynamics along the linear periodic lattice is described by **Hill’s equation**

\[ x'' + K(s)x = 0, \quad K(s) = K(s + L) \]
Linear lattice (I)

- **Dipoles**: define orbit
- **Quadrupoles**: define focusing
- Transverse single particle dynamics along the linear periodic lattice is described by Hill’s equation

\[ x'' + K(s)x = 0, \quad K(s) = K(s + L) \]

Hill’s equation can be solved to obtain the transverse linear transfer map from one point \( p_0 \) to the next \( p_1 \) along the ring

\[
\mathcal{M} = \begin{pmatrix}
\frac{\sqrt{\beta_1}}{\sqrt{\beta_1}} & 0 \\
-\frac{1}{\sqrt{\beta_1}} & \frac{\sqrt{\alpha_1}}{\sqrt{\beta_1}}
\end{pmatrix}
\begin{pmatrix}
\cos(\Delta \mu_{0\rightarrow 1}) & \sin(\Delta \mu_{0\rightarrow 1}) \\
-\sin(\Delta \mu_{0\rightarrow 1}) & \cos(\Delta \mu_{0\rightarrow 1})
\end{pmatrix}
\begin{pmatrix}
\frac{1}{\sqrt{\beta_0}} & 0 \\
\frac{\alpha_0}{\sqrt{\beta_0}} & \sqrt{\beta_0}
\end{pmatrix}
\]

\[
Q_x = \int \frac{\Delta \mu}{2\pi}
\]

**Linear lattice (II)**

- **Optics functions** can be obtained from Twiss file (e.g. MAD-X)
- **PyHEADTAIL** uses smooth approximation (only few segments per turn – very rough optics model)

\[
\beta_x = \text{constant} \\
Q_x = \frac{C}{2\pi \beta_x}, \quad \left( Q_s = \frac{\eta C}{2\pi \beta_z} \right) \\
\Delta \mu_x = Q_x \frac{L}{C} \\
L: \text{Segment length} \\
C: \text{Ring circumference}
\]

\[
M = \begin{pmatrix}
\frac{1}{\sqrt{\beta_1}} & 0 \\
-\frac{\alpha_1}{\sqrt{\beta_1}} & \frac{1}{\sqrt{\beta_1}}
\end{pmatrix}
\begin{pmatrix}
\cos(\Delta \mu_0 \to 1) & \sin(\Delta \mu_0 \to 1) \\
-\sin(\Delta \mu_0 \to 1) & \cos(\Delta \mu_0 \to 1)
\end{pmatrix}
\begin{pmatrix}
\frac{1}{\sqrt{\beta_0}} & 0 \\
\frac{\alpha_0}{\sqrt{\beta_0}} & \sqrt{\beta_0}
\end{pmatrix}
\]

Transverse transport is the matrix product applied to transverse coordinates

\[
\begin{pmatrix} x_i \\ x'_i \end{pmatrix}
\big|_1 = M \begin{pmatrix} x_i \\ x'_i \end{pmatrix}
\big|_0
\]

Matrix is identical for all the particles

---


M. Schenk et al.
PyHEADTAIL model – overview

1. Divide ring into segments separated by interaction points (IP)
2. Initialise beam
   - Typically $10^6$ macroparticles
   - Various distributions
3. Linear periodic maps for transverse tracking from one IP to the next
4. Adding chromaticity and amplitude detuning
5. (Non-)linear synchrotron motion
6. (Collective) effects
   - Wake field kicks
   - Electron cloud
   - Space charge
   - Feedback system
   - Multipole kicks
     (e.g. skew quadrupole to model linear coupling)
7. Data acquisition
   - Bunch data
   - Slice data
   - Particle data
8. 100'000s turns...

Example graphs:
- Bunch centroid vs. time
- Head-tail amplitude vs. z
Transverse tracking with detuning

- **Chromaticity** and detuning with amplitude are implemented as a **change of phase advance** for each individual particle.

- They are important to model the **head-tail mechanism** and **Landau damping**.

- We can also include higher order chromaticity.

![Diagram](image)

\[
\Delta \mu_i \sim \Delta \mu_0 + \left( \xi \delta_i + \alpha_{xx} J_{x,i} + \alpha_{xy} J_{y,i} \right) \frac{\Delta \mu_0}{2\pi Q}
\]

\[
\mathcal{M}_i = \left( \begin{array}{cc} \sqrt{\beta_1} & 0 \\ -\frac{\alpha_i}{\sqrt{\beta_1}} & \frac{1}{\sqrt{\beta_1}} \end{array} \right) \left( \begin{array}{cc} \cos(\Delta \mu_i) & \sin(\Delta \mu_i) \\ -\sin(\Delta \mu_i) & \cos(\Delta \mu_i) \end{array} \right) \left( \begin{array}{cc} 1 & 0 \\ \frac{1}{\sqrt{\beta_0}} & \frac{\alpha_0}{\sqrt{\beta_0}} \end{array} \right)
\]

\[
\left( \begin{array}{c} x_i' \\ x_i' \end{array} \right) \bigg|_1 = \mathcal{M}_i \left( \begin{array}{c} x_i \\ x_i \end{array} \right) \bigg|_0 \\
i = 1, \ldots, N
\]

Matrix elements are now different for every particle
Example: Detuning and filamentation

Start with a beam with a transverse offset (kick)

Decoherence and emittance blow-up due to non-linearities (amplitude detuning)
Example: Detuning and filamentation

Starting from a beam with a transverse offset (kick).

Example: Detuning and filamentation

- Taking an FFT of the centroid motion (black curve) reveals the tune.
- We can also perform an FFT of every macro-particle and plot the horizontal vs. vertical tune to obtain the tune footprint.

Decoherence and emittance blow-up due to non-linearities (amplitude detuning).

M. Schenk et al.
PyHEADTAIL model – overview

1. Divide ring into segments separated by interaction points (IP)

2. Initialise beam
   - Typically $10^6$ macroparticles
   - Various distributions

3. Linear periodic maps for transverse tracking from one IP to the next

4. Adding chromaticity and amplitude detuning

5. (Non-)linear synchrotron motion

6. (Collective) effects
   - Wake field kicks
   - Electron cloud
   - Space charge
   - Feedback system
   - Multipole kicks (e.g. skew quadrupole to model linear coupling)

7. Data acquisition
   - Bunch data
   - Slice data
   - Particle data

8. 100,000s turns ...

...
Longitudinal tracking

- Longitudinal particle dynamics is described by the **longitudinal equations of motion**

  \[
  z' = -\eta \delta \\
  \delta' = \frac{e V_{RF}}{m\gamma\beta^2c^2C} \sin \left( \frac{2\pi h}{C} z \right)
  \]

  - $V_{RF}$: RF voltage
  - $h = \frac{\omega_{RF}}{\omega_0}$: harmonic number
  - $\omega_0$: Revolution frequency
  - $C$: circumference

- PyHEADTAIL also models **multi-harmonic RF systems** and **particle acceleration**

- Equations of motion are solved numerically via a symplectic integration scheme (drift – kick – drift), i.e. an **iterative turn-by-turn advancement** of the coordinates

  \[
  z_{i,k+1/2} = z_{i,k} - \frac{\eta C}{2} \delta_{i,k} \\
  \delta_{i,k+1} = \delta_{i,k} + \frac{e V_{RF}}{m\gamma\beta^2c^2} \sin \left( \frac{2\pi h}{C} z_{i,k+1/2} \right) \\
  z_{i,k+1} = z_{i,k+1/2} - \frac{\eta C}{2} \delta_{i,k+1} \\
  i = 1, \ldots, N \\
  k$: iteration/turn

Example: Non-linear longitudinal tracking

Longitudinal distribution matched to stationary RF bucket
Example II: Simplified transition crossing
Example III: CERN PS bunch triple splitting
Example IV: CERN PS Booster hollow bunches

- Generation of ‘hollow’ bunches for mitigation of transverse space charge
- Idea is to flatten the charge distribution (lower peak charge line density)
- Simulations are capable of reproducing machine observations as well as predicting them

A. Oeftiger et al.
Example IV: CERN PS Booster hollow bunches

PyHEADTAIL

A. Oeftiger et al.
PyHEADTAIL model – overview

1. Divide ring into segments separated by interaction points (IP)

2. Initialise beam
   - Typically $10^6$ macroparticles
   - Various distributions

3. Linear periodic maps for transverse tracking from one IP to the next

4. Adding chromaticity and amplitude detuning

5. (Non-)linear synchrotron motion

6. (Collective) effects
   - Wake field kicks
   - Electron cloud
   - Space charge
   - Feedback system
   - Multipole kicks (e.g. skew quadrupole to model linear coupling)
   -...

7. Data acquisition
   - Bunch data
   - Slice data
   - Particle data

8. 100’000s turns...

- Bunch centroid [µm]
- Time [turns]
- Head-tail ampl. [a.u.]
- z [m]
Wake fields

Our first ‘real’ collective interaction from impedances ...
The general problem

Equations of motion of the beam particles

Interaction with the external environment

\( (\vec{E}, \vec{B}) \)

Additional electromagnetic field acting on the beam, besides RF and external magnetic fields

Multi-bunch beam

\[ s \]
The general problem

When the loop closes, either the beam will find a new stable equilibrium configuration ...
The general problem

Interaction with the external environment

Equations of motion of the beam particles

Noise

Multi-bunch beam

\[
\left( \vec{E}, \vec{B} \right)
\]

Additional electromagnetic field acting on the beam, besides RF and external magnetic fields

... or it might develop an instability along the bunch train ...
The general problem

---

Interaction with the external environment

Equations of motion of the beam particles

\( \left( \vec{E}, \vec{B} \right) \)

Additional electromagnetic field acting on the beam, besides RF and external magnetic fields

... or also an instability affecting different bunches independently of each other
**Reminder of wake functions**

**Longitudinal wake fields**

\[
\int F_s(z, s) \, ds = -q_1 q_2 \left( W_{||}(z) + O(\Delta x_1) + O(\Delta x_2) \right)
\]

Zeroth order with source and witness centred – usually dominant

Higher order terms – usually negligible for small offsets
Reminder of wake functions

Transverse wake fields

\[ \int F_x(z, s) \, ds = \]

We have truncated to the first order, thus neglecting

- First order coupling terms between x and y planes
- All higher order terms in the wake expansion (including mixed higher order terms)

Zeroth order for asymmetric structures
\( \Rightarrow \) Orbit offset

Dipole wakes – depend on source particle
\( \Rightarrow \) Orbit offset

Quadrupole wakes – depend on witness particle
\( \Rightarrow \) Detuning
Numerical implementation of wake fields

Wake functions are obtained externally from electromagnetic codes such as ACE3P, CST, GdfidL, HFSS, IW2D, ...

‘Full’ resolution

- In tracking codes, wake fields need to update the macroparticle momenta (i.e. they provide a kick)
- Wake kick on macroparticle ‘i’ created by all macroparticles ‘j’ via the wake fields is

\[ \Delta x'_i = -\frac{e^2}{m\gamma^2 c^2} \]

\[ \times \sum_{j=0}^{n_{\text{macroparticles}}} \left\{ \begin{array}{ll} W_{Cx}(z_i - z_j) \\
\Delta x_j \cdot W_{Dx}(z_i - z_j) \\
W_{Qx}(z_i - z_j) \Delta x_i \end{array} \right\} \]

Performance-optimised (PyHEADTAIL)

- For numerical efficiency, the beam is longitudinally divided into a set of slices
- If slices are thin enough, wakes can be assumed constant within slice
- Wake kick on macroparticles in slice ‘i’ created by macroparticles in slices ‘j’ is

\[ \Delta x'[i] = -\frac{e^2}{m\gamma^2 c^2} \]

\[ \times \sum_{j=0}^{n_{\text{slices}}} \left\{ \begin{array}{ll} N[j] \cdot W_{Cx}[i - j] \\
N[j]\langle x \rangle[j] \cdot W_{Dx}[i - j] \\
N[j] \cdot W_{Qx}[i - j] \Delta x[i] \end{array} \right\} \]
Numerical implementation in PyHEADTAIL

Bunch & slices

Wake field $W_{C^\circ}(\theta)$

Slice index

$n$  $i$  $0$
1. Bin particles into slices – binning needs to be fine enough to sample the wake function.
1. Bin particles into slices – binning needs to be fine enough to sample the wake function

2. Perform convolution to obtain wake kicks

\[ \Delta x'[i] = -\frac{e^2}{m\gamma\beta^2 c^2} \sum_{j=0}^{i} N[j] \cdot W_{Cx}[i - j] \]
Numerical implementation in PyHEADTAIL

1. Bin particles into slices – binning needs to be fine enough to sample the wake function
2. Perform convolution to obtain wake kicks
3. Apply wake kicks (momentum update)

$$
\Delta x'[i] = -\frac{e^2}{m\gamma^2\beta^2 c^2} \sum_{j=0}^{i} N[j] \cdot W_{C\phi}[i-j], \quad x'[i] \rightarrow x'[i] + \Delta x'[i], \quad i = 1, \ldots, n_{\text{slices}}
$$
Intermediate summary

We are now ready to track a full turn including the interaction with wake fields

1. **Initialise** a macroparticle distribution

2. Update **transverse coordinates** and momenta according to the **linear transfer map** – adjust the individual phase advance according to **chromaticity** and **amplitude detuning**

3. Update the **longitudinal coordinates** and momenta according to the leap-frog integration scheme

4. Update momenta only (apply kicks) according to **wake field** generated kicks

5. Repeat turn-by-turn ...

\[
\begin{pmatrix}
  x_i \\
  x'_i \\
\end{pmatrix}
_{k+1}
 =
\mathcal{M}_i
\begin{pmatrix}
  x_i \\
  x'_i \\
\end{pmatrix}
_{k}
\]
Example I: Dipole wakes

\[ \Delta x'[i] = -\frac{e^2}{m\gamma\beta^2c^2C} \sum_{j=0}^{i} N[j] \langle x \rangle[j] \cdot W_{Dx}[i-j] \]

- Without synchrotron motion
  - kicks accumulate turn after turn – the beam is unstable → like beam break-up in linacs

Dipolar term → orbit kick

Offset dependent orbit kick → kicks can accumulate
Example I a: Dipole wakes – beam break-up

Without synchrotron motion
Example II: Dipole wakes

\[ \Delta x'[i] = -\frac{e^2}{m\gamma^2 c^2 C} \sum_{j=0}^{i} N[j] \langle x \rangle[j] \cdot W_{Dx}[i-j] \]

- **Without synchrotron motion**
  - Kicks accumulate turn after turn – the beam is unstable → like beam break-up in linacs

- **With synchrotron motion**
  - Chromaticity = 0
    - Synchrotron sidebands are well separated → beam is stable
    - Synchrotron sidebands couple → (transverse) mode coupling instability
  - Chromaticity ≠ 0
    - Headtail modes → beam is unstable (can be very weak and often damped by non-linearities, i.e. Landau damping)
Example II a: Dipole wakes – TMCI below threshold

As the intensity increases the coherent modes shift – here, modes A and B are approaching each other.
When the two modes merge, a fast coherent instability arises – the transverse mode coupling instability (TMCI) which often is a hard intensity limit in many machines.
Example II b: Dipole wakes – TMCI above threshold
Raising the TMCI threshold – SPS Q20 optics

- In PyHEADTAIL we have the possibility to perform scans of variables, e.g. we can run many simulations in parallel each with different beam intensity.
- We can then perform a spectral analysis of each simulation and stack them all behind one another to obtain the typical visualisation plots of TMCI.

![Graph showing Mode A and Mode B with TMCI threshold highlighted.](image_url)
Raising the TMCI threshold – SPS Q20 optics

• In PyHEADTAIL we have the possibility to perform scans of variables, e.g. we can run many simulations in parallel each with different beam intensity
• We can then perform a spectral analysis of each simulation and stack them all behind one another to obtain the typical visualisation plots of TMCI
• By lowering the transverse tunes, the slip factor $\eta$ can be increased and therewith the TMCI threshold (analytically: $I_{TMCI} \propto \eta$)
**Relativistic vs. non-relativistic wakes**

**Fully relativistic wakes**
- Relativistic wakes only affect **trailing particles** following the source particle.
- Finite values range for negative distances, i.e. \((-L, 0)\) or “tail – head”.

**Non-relativistic wakes**
- Nonrelativistic wakes can also affect particles ahead of the source particle.
- Finite values extend from \((-L, L)\) or “tail – head” & “head – tail”.

---

![Graph showing the comparison between relativistic and non-relativistic wakes](graph.png)

- **Source**
- **Witness trailing**
- **Witness ahead**

---

*M. Schenk et al.*

05.07.2017
Example non-relativistic wakes

- PS injection oscillations show intra-bunch modulations
- Since $\beta \approx 0.91$, wakes can affect witnesses ahead of the source.
- Intra-bunch modulations are only reproduced when adding non-relativistic wakes caused by indirect space charge fields

A. Huschauer et al
Example non-relativistic wakes

- PS injection oscillations show intra-bunch modulations
- Since $\beta \approx 0.91$, wakes can affect witnesses ahead of the source.
- Intra-bunch modulations are only reproduced when adding non-relativistic wakes caused by indirect space charge fields

A. Huschauer et al
Example IV: Head-tail modes

• As soon as chromaticity is non-zero, another ‘resonant’ condition can be met as particles now ‘synchronise’ their betatron motion with the synchrotron motion.

• Head-tail modes arise – the order of the respective mode depends on the chromaticity together with the impedance and bunch spectrum.
Example IV: Head-tail modes
Example IV: Head-tail modes in the LHC

LHC measurement

PyHEADTAIL

Q'' = 0
m = 0

Q'' = -40'000
m = -1
Example V: Benchmarks against theory (airbag beam)

- Head-tail modes with an airbag beam
- Excellent agreement between
  - PyHEADTAIL (*red*)
  - Circulant matrix model (*blue*)
  - Analytical formula (*solid lines*)
PyHEADTAIL model – summary

1. Divide ring into segments separated by interaction points (IP)
2. Initialise beam
   - Typically $10^6$ macroparticles
   - Various distributions
3. Linear periodic maps for transverse tracking from one IP to the next
4. Adding chromaticity and amplitude detuning
5. (Non-)linear synchrotron motion
6. (Collective) effects
   - Wake field kicks
   - Electron cloud
   - Space charge
   - Feedback system
   - Multipole kicks (e.g. skew quadrupole to model linear coupling)
7. Data acquisition
   - Bunch data
   - Slice data
   - Particle data
8. 100’000s turns...
Outline

• Part I: PyHEADTAIL
  • Introduction
  • PyHEADTAIL protocol
  • How to build a simulation
  • Details on the model with illustrations
  • Practical information

• Part II: PyECLOUD
  • Introduction to electron cloud and PyECLOUD
  • Build-up simulations
  • Beam dynamics simulations with PyECLOUD & PyHEADTAIL
  • Practical information
What I did not talk about …

• PyHEADTAIL also has an extensive space charge module
• Recently it has gotten multi bunch support making simulations of coupled-bunch instabilities possible
• Feedback systems module is undergoing big upgrades (ongoing work)
• Longitudinal wake fields and instabilities (microwave, energy loss, …)
• Multipolar kicks (e.g. for more realistic amplitude detuning model following from octupolar kicks)
• Radiation damping module
• A large part of our efforts go into performance optimisations of the code: MPI, openMP, GPU acceleration with PyCUDA, cython, …

... and more …
PyHEADTAIL is open source, available at https://github.com/PyCOMPLETE/PyHEADTAIL

Programming languages
- Mostly Python
- Computationally intensive parts are implemented in cython, or are GPU (PyCUDA) / OpenMP / MPI parallelised

Operating systems
- Best on Linux (tested extensively on Ubuntu and SLC 5 and 6)
- Also runs on Windows, but installation is not as straightforward

Other prerequisites
- Python 2.7 (move to Python 3 is planned for the nearer future)
- Open source python libraries: numpy, scipy, matplotlib, cython, h5py, (pycuda)
- CERN PyPIC library

Available documentation and resources
- Git Wiki pages including instructions for installation and further links
- A set of examples and test scripts, showing amongst others some functionailty and usage of the code

Your contributions are very welcome 😊
Outline

• Part I: PyHEADTAIL
  • Introduction
  • PyHEADTAIL protocol
  • How to build a simulation
  • Details on the model with illustrations
  • Practical information

• Part II: PyECLOUD
  • Introduction to electron cloud and PyECLOUD
  • Build-up simulations
  • Beam dynamics simulations with PyECLOUD & PyHEADTAIL
  • Practical information
Secondary Electron Emission can drive an avalanche multiplication effect filling the beam chamber with an electron cloud.

Trailing bunches of the train interact with a dense electron cloud:
- Transverse instabilities
- Transverse emittance blow-up
- Particle losses

Electron cloud induces other unwanted effects like:
- Heat load on the beam chambers
- Vacuum degradation
**PyECloud**

**PyECloud** is the simulation code for the simulation of electron cloud (e-cloud) effects

- **2D Macroparticle (MP) code**
- Developed and maintained at CERN since 2011 following the legacy of the ECLOUD code (F. Zimmermann et al., 1997...)

It can be used to simulate different effects

- **Build-up simulations**: in stand-alone mode for the simulation of the e-cloud formation at a certain section of an accelerator
- **Effects on the beam dynamics** (e.g. instabilities, e-cloud detuning) in combination with the PyHEADTAIL code
- **Fast beam-ion instabilities** in combination with the PyHEADTAIL code
Outline

• Part I: PyHEADTAIL
  • Introduction
  • PyHEADTAIL protocol
  • How to build a simulation
  • Details on the model with illustrations
  • Practical information

• Part II: PyECLOUD
  • Introduction to electron cloud and PyECLOUD
  • Build-up simulations
  • Beam dynamics simulations with PyECLOUD & PyHEADTAIL
  • Practical information
Simulation of the **electron multipacting** process

- The **beam is rigid** (fixed charge distribution)
- Electrons are tracked and **secondary emission** models are applied when particles impact on the wall
Evaluate the electric field of beam at each MP location

Evaluate the e⁻ space charge electric field

Compute MP motion \( (t \rightarrow t + \Delta t) \)

Detect impacts and generate secondaries
“Primary electrons” are generated by:

- **Ionization of the residual gas** in the vacuum chamber
- **Photoemission** from the chamber’s walls due to **synchrotron radiation**
PyECLoud build-up simulations

1. t = t + Δt
2. Generate seed e^-
3. Evaluate the electric field of beam at each MP location
4. Evaluate the e^- space charge electric field
5. Compute MP motion (t -> t + Δt)
6. Detect impacts and generate secondaries

Beam field evaluated on a fixed grid and interpolated at each MP location

Several options implemented:
- Bassetti – Erskine formula with image charges (elliptic chambers)
- Finite Difference Poisson Solver (arbitrary shaped chambers)
- Field map imported from file

Multiple beams can be simulated (e.g. LHC triplets)
Evaluate the electric field of beam at each MP location

Evaluate the \( e^- \) space charge electric field

Compute MP motion \((t \rightarrow t + \Delta t)\)

Detect impacts and generate secondaries

Effect of the electrons on themselves evaluated by Particle In Cell (PIC) to solve the Poisson equation

- **Arbitrary shaped** chambers can be simulated
- **Shortley-Weller** method for refined approximation of curved boundaries on a rectangular grid
- Important **speed-up** achieved through C-implemented **KLU factorization algorithm** (cython)
- Available as a **standalone module (PyPIC)** for different usages (e.g. space charge module for PyHEADTAIL)
Evaluate the electric field of beam at each MP location

Evaluate the $e^-$ space charge electric field

Compute MP motion ($t \rightarrow t + \Delta t$)

Detect impacts and generate secondaries

The dynamics equation is integrated to update MP position and momentum.

Two algorithms are implemented:

- **Semi-analytic** algorithm inherited from the ECLoud code (effective for dipole and field free regions)
- **Boris algorithm** with substeps (phase space volume preserving) → proved to be necessary for quadrupoles and combined function magnets
  - Speedup using f2py and cython
  - Dipole and quadrupole natively available, arbitrary B-field maps can be loaded from file
Evaluate the electric field of beam at each MP location

Evaluate the $e^-$ space charge electric field

Compute MP motion ($t \rightarrow t + \Delta t$)

Detect impacts and generate secondaries

When a MP hits the wall Secondary Electron Emission models are employed to generate charge, energy and angle of the emitted charge

- **Secondary Electron Yield (SEY)** can be non-uniform on the chamber surface

According to the number of emitted electrons, the MP can be simply rescaled or new MPs can be generated

- Part of a quite complex MP size management ...
PyE CLOUD build-up simulations – Usage at CERN

Used for studies on many machines (e.g. PS, SPS, LHC, CLIC-DR, FCC-ee, FCC-hh) to

- Identify surface (SEY) requirements on newly installed elements (multipacting thresholds)
- Compute expected heat loads on cryogenic components
- Compute expected electron flux on beam chamber walls (for vacuum estimates)
- Reconstruct SEY from measured heat loads (measured bunch-by-bunch properties can be loaded for simulations)
- Simulate electron cloud detectors (e.g. shielded pickup)
Example I: E-clouds in a drift section

Beam passage leads to a **pinch of the cloud** which in turn acts back on the beam – differently each turn
Example II: E-clouds in a dipole magnet

Beam passage leads to a pinch of the cloud which in turn acts back on the beam – differently each turn.
Example III: E-clouds in a quadrupole magnet

Beam passage leads to a pinch of the cloud which in turn acts back on the beam – differently each turn.
Outline

• Part I: PyHEADTAIL
  • Introduction
  • PyHEADTAIL protocol
  • How to build a simulation
  • Details on the model with illustrations
  • Practical information

• Part II: PyECLOUD
  • Introduction to electron cloud and PyECLOUD
  • Build-up simulations
  • Beam dynamics simulations with PyECLOUD & PyHEADTAIL
  • Practical information
Electrons are attracted by the proton bunches and ‘fly’ through it exerting **significant electromagnetic forces on beam particles** → impact on beam dynamics

- **Coherent effects**: tune shift, transverse instabilities
- **Incoherent effects**: tune spreads, non-linear forces → losses, emittance blow-up

- To simulate this kind of effects, **we need to track both the e-cloud particles and the beam particles**
PyE CLOUD beam dynamics simulations

- Since 2014, we dropped the approach of having separate tools for build-up and instability
- Use PyE CLOUD also to simulate the interaction beam/e-cloud within PyHEADTAIL
  → Possible thanks to the highly modular structure of the two codes

---

**Legend:** From PyHEADTAIL / From PyE CLOUD / Developed ad hoc

---

PyHEADTAIL

- Transverse tracking → with Q’, octupoles etc.
- Longitudinal tracking
- Transverse feedback
- Impedances
- Space charge
- ...

PyHEADTAIL slicer

**PyE CLOUD (PyEC4PyHT object)**

- Initial e-distribution (from PyE CLOUD buildup sim.)
- For each slice:
  - Generate seed e⁻
  - Evaluate beam slice electric field (Particle in Cell)
  - Evaluate the e⁺electric field (Particle in Cell)
  - Apply kick on the beam particles
  - Compute e⁺ motion (t→t+Δt) (possibly with substeps)
  - Detect impacts and generate secondaries

PyHEADTAIL bunch

- Transverse tracking → with Q’, octupoles etc.
- Longitudinal tracking
- Transverse feedback
- Impedances
- Space charge
- …
PyECLoud beam dynamics simulations

- Since 2014, we dropped the approach of having separate tools for build-up and instability
- **Use PyECLoud** also to simulate the interaction beam/e-cloud within PyHEADTAIL
  → Possible thanks to the highly modular structure of the two codes

In this framework we could significantly improve our modeling of e-cloud effects on beam dynamics:

- Accurate **tracking** of the electrons in dipoles and quadrupoles
- **Multipacting** can be included
- Realistic **boundary conditions**
- Possibility to record the e-cloud pinch to evaluate detuning along the bunch (tune footprint)
- Combine **different e-clouds** in the same simulation (e.g. dipoles & quadrupoles)
- Non-uniform **optics functions** (e.g. LHC Insertion Region), off-centered beams...
- We could generalise it for **fast-ion instabilities**

And of course more efficient in terms of development and maintainance compared to having two separate tools with largely duplicated code
Example: LHC and SPS beam dynamics with e-cloud

Used up to now especially for SPS and LHC to estimate:

- **Stability** thresholds
- **Tune shifts and tune spreads**

Including several effects:

- **e-cloud in different machine elements** (dipoles and/or quadrupoles and/or drifts)
- And all **beam dynamics** modeling available in PyHEADTAIL
  - Non uniform optics
  - Transverse feedback
  - Chromaticity
  - Octupoles
Example: E-cloud coherent instability threshold

- Typical e-cloud simulation: try to identify the e-cloud central density threshold for an instability
- Scans in the central density are performed until exponential growth in emittance is observed

Coherent instabilities occur when a certain central cloud density threshold is breached
- This leads to coherent intra bunch motion which grows exponentially
- A consequence is emittance blow-up and losses
Example: E-cloud instability measured in LHC

- First injection of 48 bunches of 25 ns beam into the LHC in 2011
- Beam was dumped twice due to a violent instability in the vertical plane, causing losses above the interlock threshold
**Example: E-cloud instability in LHC – measured and simulated**

48b injection test (26/08/11)

Some motion only for last bunches ...  
Bunch 25 is the first unstable  
up to ±5mm  

48x PyECLoud e^ distribution (δ_{max} = 2.1)

48x HEADTAIL simulations reveal the onset of instability
Example: Incoherent effect from e-cloud

- With LHC tunes at $Q_v = 0.305$, beam lifetime drops as $Q'$ is increased to 15 units.
- Reason is the third order resonance line that is crossed by some particles ...
- ... due to large tune spread from high $Q'$, e-cloud, and octupoles.
Example: Incoherent effect from e-cloud

- PyHEADTAIL & PyE CLOUD: allows to obtain tune footprint for all effects separately ...
- ... as well as for all of them combined

A. Romano et al.
Example: Incoherent effect from e-cloud

• PyHEADTAIL & PyECLOUD: allows to obtain tune footprint for all effects separately ...
• ... as well as for all of them combined
-> allows to identify the source of the losses.
-> Lower the tune to have more margin.

A. Romano et al.
PyECLoud is also open source and available at
https://github.com/PyCOMPLETE/PyECLoud

Programming languages
- Mostly Python (95%)
- Fortran (linked via f2py) and C (linked via cython) for computationally intensive parts

Operating systems
- Linux (tested extensively on Ubuntu and SLC 5 and 6)
- Should run also on Windows, but not tested

Other prerequisites
- Python 2.7+
- Open source python libraries: numpy, f2py, scipy, matplotlib, cython
- CERN PyPIC library

Available documentation and resources
- Reference manual (documenting input and output)
- Git Wiki pages including instructions for installation and further links
- Examples and test scripts
Main material

- K. Li et al., *HEADTAIL code*, CERN ABP information meeting, Geneva (CH), March 2015. [https://indico.cern.ch/event/378615/](https://indico.cern.ch/event/378615/)
- K. Li et al., *PyHEADTAIL*, CERN ABP-CWG meeting, Geneva (CH), November 2016, [https://indico.cern.ch/event/580876/](https://indico.cern.ch/event/580876/)
- G. Iadarola et al., *PyECLOUD*, CERN ABP-CWG meeting, Geneva (CH), March 2017, [https://indico.cern.ch/event/580885/](https://indico.cern.ch/event/580885/)

Some further reading

- L. Mether et al., *FASTION*, CERN ABP-CWG meeting, Geneva (CH), November 2016, [https://indico.cern.ch/event/580878/](https://indico.cern.ch/event/580878/)
- A. Romano et al., Electron cloud instability studies for the LHC, CERN, Geneva (CH), March 2017, [https://indico.cern.ch/event/605497/](https://indico.cern.ch/event/605497/)
Backup
Electron clouds in a bending magnet

• The electrons exhibit different transverse (x,y) distributions, according to the type of region in which the electron cloud is formed.
  • In dipole regions, the electron motion is confined along the lines of the magnetic field. Example: snapshots of multipacting in the dipole of an LHC arc cell during bunch passage and including secondary production.
Electron clouds in a quadrupole magnet

- The electrons exhibit different transverse (x,y) distributions, according to the type of region in which the electron cloud is formed.
  - In quadrupole regions, the electrons tend to multipact along the pole-to-pole lines of the cross section (example: snapshots of multipacting in an LHC arc quadrupole). Multipacting thresholds are usually lower in quadrupoles because electrons survive long thanks to trapping due to the magnetic gradient.

\[
\Delta t = 0 \quad \Delta t = 0.5 \text{ ns} \quad \Delta t = 1 \text{ ns} \quad \Delta t = 1.5 \text{ ns} \quad \Delta t = 2 \text{ ns} \\
\Delta t = 2.5 \text{ ns} \quad \Delta t = 3 \text{ ns} \quad \Delta t = 5 \text{ ns} \quad \Delta t = 7 \text{ ns} \quad \Delta t = 9 \text{ ns}
\]
A key component of the e-cloud simulator is the Particle In Cell (PIC) Poisson solver

Initially PyE CLOUD included a simple Particle-In-Cell solver

→ We decided to reorganize our Particle In Cell (PIC) Poisson solvers

→ We wrote a Python library (PyPIC) including different PIC solvers having the same interface which can be used as plug-in modules for PyE CLOUD but also for other applications (e.g. space charge, beam-beam)

PyPIC is now available on the PyCOMPLETE git repository: https://github.com/PyCOMPLETE/PyPIC/
A key component of the e-cloud simulator is the Particle In Cell (PIC) Poisson solver.

PyPIC includes the following solvers:

- Open boundary FFT
- Perfect Electric Conductor (PEC) rectangular FFT
- PEC arbitrarily shaped boundary – Finite Differences, staircase approx. of curved boundaries
- PEC arbitrarily shaped boundary – Finite Differences, Shortley-Weller method for curved boundaries (more accurate)
- Fast Polar Poisson solver
- GPU FFT solvers

- **Telescopic grid mode** introduced since PyPIC 2.0 → several grids with different cell size can be nested, for example to have increased resolution only where needed (e.g. at the beam location)
PIC solvers in brief

• In many of our codes, Particle in Cell (PIC) algorithms are used to compute the electric field generated by a set of charged particles in a set of discrete points (can be the locations of the particles themselves, or of another set of particles)

• The solution typically consists of 4 stages:
  1. Charge scatter from macroparticles (MPs) to grid (reduction of macroparticles)
  2. Calculation of the electrostatic potential at the nodes
  3. Calculation of the electric field at the nodes (gradient evaluation)
  4. Field gather from grid to MPs
PIC solvers in brief

• The solution typically consists of 4 stages:
  1. **Charge scatter** from macroparticles (MPs) to grid (reduction of macroparticles)
  2. Calculation of the **electrostatic potential at the nodes**
  3. Calculation of the **electric field at the nodes** (gradient evaluation)
  4. **Field gather** from grid to MPs

![Uniform square grid](image_url)
PIC solvers – basic steps

- The solution typically consists of 4 stages:
  1. **Charge scatter** from macroparticles (MPs) to grid (reduction of macroparticles)
  2. Calculation of the **electrostatic potential** at the nodes
  3. Calculation of the **electric field at the nodes** (gradient evaluation)
  4. **Field gather** from grid to MPs

\[ \rho_{i,j} = \rho_{i,j} + \frac{q n_{MP}}{\Delta h} \left( 1 - \frac{d_x}{\Delta h} \right) \left( 1 - \frac{d_y}{\Delta h} \right) \]
\[ \rho_{i+1,j} = \rho_{i+1,j} + \frac{q n_{MP}}{\Delta h} \left( \frac{d_x}{\Delta h} \right) \left( 1 - \frac{d_y}{\Delta h} \right) \]
\[ \rho_{i,j+1} = \rho_{i,j+1} + \frac{q n_{MP}}{\Delta h} \left( 1 - \frac{d_x}{\Delta h} \right) \left( \frac{d_y}{\Delta h} \right) \]
\[ \rho_{i+1,j+1} = \rho_{i+1,j+1} + \frac{q n_{MP}}{\Delta h} \left( \frac{d_x}{\Delta h} \right) \left( \frac{d_y}{\Delta h} \right) \]
PIC solvers – basic steps

• The solution typically consists of 4 stages:
  1. Charge scatter from macroparticles (MPs) to grid (reduction of macroparticles)
  2. Calculation of the electrostatic potential at the nodes
  3. Calculation of the electric field at the nodes (gradient evaluation)
  4. Field gather from grid to MPs

\[ \nabla^2 \phi(x, y) = -\frac{\rho(x, y)}{\varepsilon_0} \]

Boundary conditions (e.g., perfectly conducting, open, periodic)

• Different numerical approaches exist to solve these types of equations each with its own advantages and drawbacks:
  • Open space FFT solver (explicit, very fast but open boundaries)
  • Rectangular boundary FFT solver (explicit, very fast but only rectangular boundaries)
  • Finite Difference implicit Poisson solver (arbitrary chamber shape, sparse matrix, possibility to use Shortley Weller boundary refinement, KLU fast routines, computationally more demanding)
  • Dual or multi-grid in combination with direct or iterative solvers
PIC solvers – basic steps

• The solution typically consists of 4 stages:
  1. **Charge scatter** from macroparticles (MPs) to grid (reduction of macroparticles)
  2. Calculation of the **electrostatic potential at the nodes**
  3. **Calculation of the electric field at the nodes** (gradient evaluation)
  4. Field **gather** from grid to MPs

\[
E = - \nabla \phi
\]

\[
(E_x)_{i,j} = - \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta h}
\]

\[
(E_y)_{i,j} = - \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta h}
\]
PIC solvers – basic steps

• The solution typically consists of 4 stages:
  1. Charge scatter from macroparticles (MPs) to grid (reduction of macroparticles)
  2. Calculation of the electrostatic potential at the nodes
  3. Calculation of the electric field at the nodes (gradient evaluation)
  4. Field gather from grid to MPs

\[
E(x_{\text{MP}}, y_{\text{MP}}) =
\]

\[
E_{i,j} \left(1 - \frac{d_x}{\Delta h}\right) \left(1 - \frac{d_y}{\Delta h}\right) + E_{i+1,j} \left(\frac{d_x}{\Delta h}\right) \left(1 - \frac{d_y}{\Delta h}\right)
\]

\[
+ E_{i,j+1} \left(1 - \frac{d_x}{\Delta h}\right) \left(\frac{d_y}{\Delta h}\right) + E_{i+1,j+1} \left(\frac{d_x}{\Delta h}\right) \left(\frac{d_y}{\Delta h}\right)
\]