Proposal for a common interface for our PIC Poisson Solvers --- PyPIC? ;-) ---

G. Iadarola, G. Rumolo

PyHEADTAIL meeting – 16/12/2014

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)

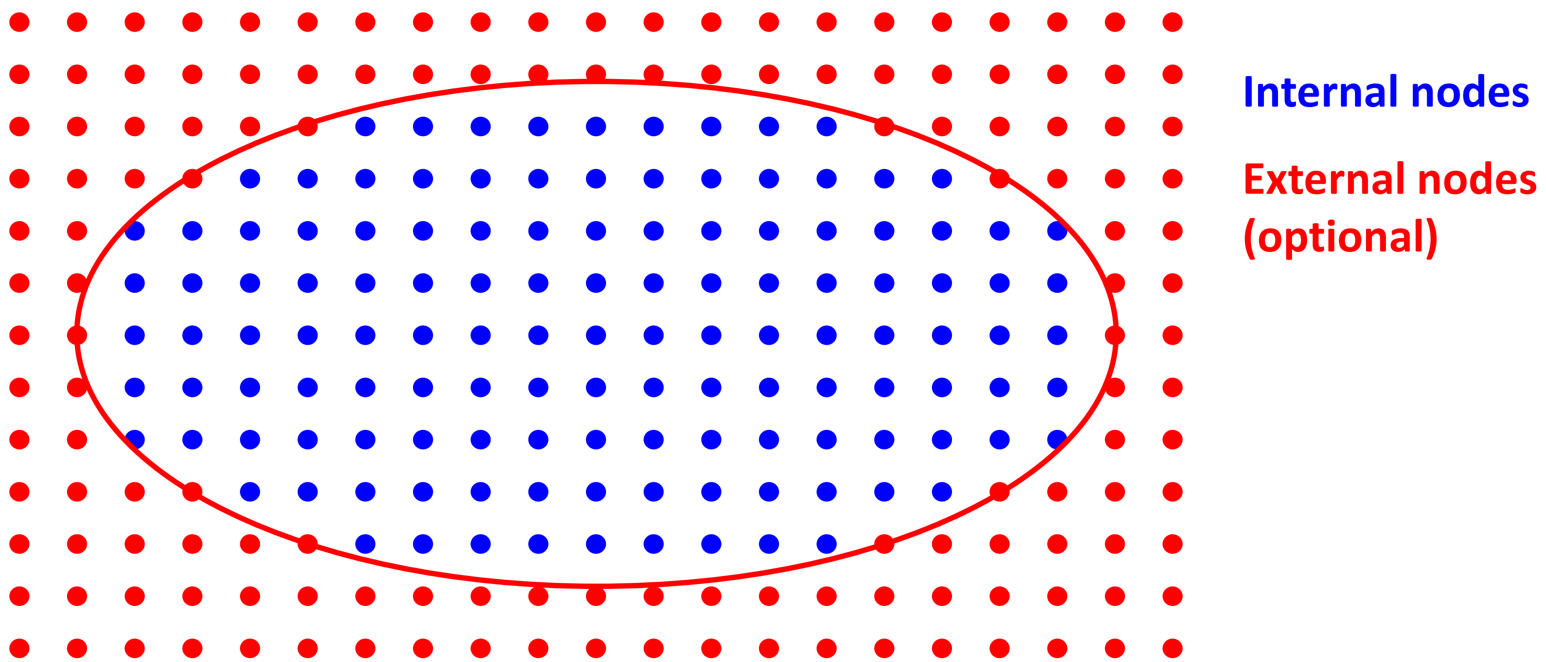
Typically 4 stages:

1. Charge scatter from macroparticles (MPs) to grid
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

Basic stages of a PIC algorithm

Standard Particle In Cell (PIC) → 4 stages:

1. Charge scatter from macroparticles (MPs) to grid
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

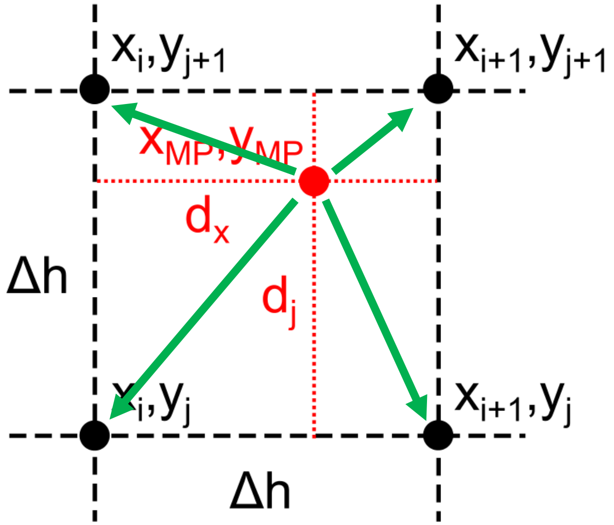


Uniform square grid

Basic stages of a PIC algorithm

Standard Particle In Cell (PIC) → 4 stages:

1. Charge scatter from macroparticles (MPs) to grid
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



$$\rho_{i,j} = \rho_{i,j} + \frac{q\, n_{\mathrm{MP}}}{\Delta h}\left(1 - \frac{d_x}{\Delta h}\right)\left(1 - \frac{d_y}{\Delta h}\right)$$

$$\rho_{i+1,j} = \rho_{i+1,j} + \frac{q\, n_{\mathrm{MP}}}{\Delta h}\left(\frac{d_x}{\Delta h}\right)\left(1 - \frac{d_y}{\Delta h}\right)$$

$$\rho_{i,j+1} = \rho_{i,j+1} + \frac{q\, n_{\mathrm{MP}}}{\Delta h}\left(1 - \frac{d_x}{\Delta h}\right)\left(\frac{d_y}{\Delta h}\right)$$

$$\rho_{i+1,j+1} = \rho_{i+1,j+1} + \frac{q\, n_{\mathrm{MP}}}{\Delta h}\left(\frac{d_x}{\Delta h}\right)\left(\frac{d_y}{\Delta h}\right)$$

Basic stages of a PIC algorithm

Standard Particle In Cell (PIC) → 4 stages:

1. Charge scatter from macroparticles (MPs) to grid
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

$$\begin{cases} \nabla^2 \phi(x,y) = -\dfrac{\rho(x,y)}{\varepsilon_0} \\ \textbf{Boundary conditions} \text{ (e.g., perfectly conducting, open, periodic)} \end{cases}$$

Different numerical approaches to the solution with different advantages and drawbacks.

Already implemented in our codes:

- **FASTION, HEADTAIL and PyHEADTAIL**: Open space FFT solver (explicit, very fast but open boundaries)
- **HEADTAIL**: Rectangular boundary FFT solver (explicit, very fast but only rectangular boundaries)
- **FASTION**: dual grid (see Lotta's presentation)
- **PyECLOUD (and PyEC4PyHT)**: Finite Difference implicit Poisson solver (arbitrary chamber shape, sparse matrix, possibility to use Shortley Weller boundary refinement, KLU fast routines, computationally more demanding)

Standard Particle In Cell (PIC) → 4 stages:

1. Charge scatter from macroparticles (MPs) to grid
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

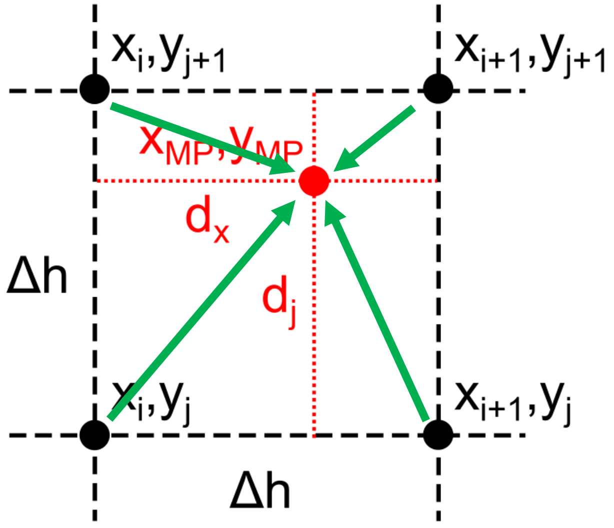
$$\mathbf{E} = -\nabla\phi$$

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

$$(E_y)_{i,j} = -\frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta h}$$

Standard Particle In Cell (PIC) → 4 stages:

1. Charge scatter from macroparticles (MPs) to grid
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



$$\mathbf{E}\left(x_{\mathrm{MP}}, y_{\mathrm{MP}}\right) =$$

$$\mathbf{E}_{i,j}\left(1-\frac{d_x}{\Delta h}\right)\left(1-\frac{d_y}{\Delta h}\right)+\mathbf{E}_{i+1,j}\left(\frac{d_x}{\Delta h}\right)\left(1-\frac{d_y}{\Delta h}\right)$$
$$+\mathbf{E}_{i,j+1}\left(1-\frac{d_x}{\Delta h}\right)\left(\frac{d_y}{\Delta h}\right)+\mathbf{E}_{i+1,j+1}\left(\frac{d_x}{\Delta h}\right)\left(\frac{d_y}{\Delta h}\right)$$

Advantages of a common interface

We could collect all the code we already have in a **common tool** (PyPIC? ;-)), with a **common interface**. This would mean:

- A **single implementation for common features** (scatter, gather, etc…)
- **All implementations become naturally available for all users** (fast rectangular or dual grid in PyECLOUD and PyHEADTAIL, Finite Difference in FASTION)
- With a bit of tweaking we could get a first **space charge module for PyHEADTAIL** (?) (practically already debugged…)
- **New implementations easier to test and debug** and immediately available for all usages

Something like this…

First idea on a possible interface

```
class ACertainPIC:

        def  __init__ (self, geometry, grid, numerical_param):

                ........

        def scatter(self, x_mp, y_mp, q_mp):

                ........

        def solve(self)

                ........

        def gather (self, x_mp, y_mp):

                ........

                return Ex_mp, Ey_mp

        ........
```

First idea on a possible interface

Example of usage:

```
import PyPIC.PIC_FFT as PyPICFFT

mypic = PyPICFFT(xmin, xmax, dx, ymin, ymax,dy)



<some code which gives x_mp, y_mp, q_mp>


mypic.scatter(x_mp, y_mp, q_mp)
mypic.solve()
Ex_mp, Ey_mp = mypic.gather(x_mp, y_mp)
```

Now I want to change the solver:

```
import PyPIC.PIC_FFT as PyPICFFT

mypic = PyPICFFT(xmin, xmax, dx, ymin, ymax,dy)



<some code which gives x_mp, y_mp, q_mp>


mypic.scatter(x_mp, y_mp, q_mp)
mypic.solve()
Ex_mp, Ey_mp = mypic.gather(x_mp, y_mp)
```

First idea on a possible interface

Now I want to change the solver:

```
#import PyPIC.PIC_FFT as PyPICFFT
import PyPIC.PIC_FD as PyPICFD
#mypic = PyPICFFT(xmin, xmax, dx, ymin, ymax,dy)
mypic = PyPICFD(chamber_object, dx, dy)

<some code which gives x_mp, y_mp, q_mp>

mypic.scatter(x_mp, y_mp, q_mp)
mypic.solve()
Ex_mp, Ey_mp = mypic.gather(x_mp, y_mp)
```

First idea on a possible interface

If I want compare the two:

```
import PyPIC.PIC_FFT as PyPICFFT
import PyPIC.PIC_FD as PyPICFD
mypic1 = PyPICFFT(xmin, xmax, dx, ymin, ymax,dy)
mypic2 = PyPICFD(chamber_object, dx, dy)

<some code which gives x_mp, y_mp, q_mp>

mypic1.scatter(x_mp, y_mp, q_mp)
mypic1.solve()
Ex1_mp, Ey1_mp = mypic1.gather(x_mp, y_mp)

mypic2.scatter(x_mp, y_mp, q_mp)
mypic2.solve()
Ex2_mp, Ey2_mp = mypic2.gather(x_mp, y_mp)

print norm(Ex1-Ex2)/norm(Ex1)
print norm(Ey1-Ey2)/norm(Ey1)
```

Thanks for your attention!

Basic stages of a PIC algorithm

Standard Particle In Cell (PIC) → 4 stages:

1. Charge scatter from macroparticles (MPs) to grid
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

$$\begin{cases} \nabla^2 \phi(x,y) = -\dfrac{\rho(x,y)}{\varepsilon_0} \\ \phi(x,y) = 0 \qquad \text{on the boundary} \end{cases}$$

Internal nodes:

$$\frac{\phi_{i-1,j} + \phi_{i,j-1} - 4\phi_{i,j} + \phi_{i+1,j} + \phi_{i,j+1}}{\Delta h^2} = -\frac{\rho_{i,j}}{\varepsilon_0}$$

External nodes:

$$\phi_{i,j} = 0$$

Can be written in matrix form:

$$\underline{\underline{A}}\,\underline{\phi} = \frac{1}{\varepsilon_0}\underline{\rho}$$

A is sparse and depends only on chamber geometry and grid size

→ It can be computed and LU factorized in the initialization stage to speed up calculation