ISBN: 978-3-95450-132-8 doi:10.18429/JACoW-IPAC2014-MOPME013

# A PYTHON POISSON SOLVER FOR 3D SPACE CHARGE COMPUTATIONS IN STRUCTURES WITH ARBITRARY SHAPED BOUNDARIES

G. Pöplau*, C. Potratz, Compaec e. G., Rostock, Germany

*Abstract*

Numerical techniques in the field of particle accelerators are mainly driven by the design of next-generation accelerators: The need for higher simulation complexity and the necessity for more and more specialized algorithms arises from the ever increasing need to include a broader range of physical effects and geometrical details in a computer simulation. This, on the other hand requires fast and reliable simulation tools for a limited user base. Therefore, new approaches in simulation software development are necessary to provide useful tools that are well-suited for the task at hand and that can be easily maintained and adapted by a small user community. We show how Python can be used to solve numerical problems arising from particle accelerator design efficiently. As model problem, the computation of space charge effects of a bunch in RFQs including the vane geometry was chosen and a suited solver was implemented in Python.

## INTRODUCTION

Precise and efficient 3D space charge simulations are an important problem in accelerator design. Hereby, the requirement to include the shape of the accelerating structure comes more and more into focus. Representative problems are for instance the simulation of electron or ion cloud instabilities or the computation of the field of a bunch within an RFQ structure. On the other hand there are only a few tracking codes, that consider the true shape of the structure in space charge calculations, for example MOEVE PIC Tracking [1, 2] and OPAL [3]. Within the tracking Code MOEVE PIC Tracking, which was developed for the simulation of electron and ion cloud effects, a Poisson solver for beam pipes with elliptical cross-section was implemented.

In this paper we present a new approach for a Poisson solver applicable in structures with arbitrary shaped boundaries and present a first implementation in Python. Hereby, we take advantage of Python's capability for fast algorithm development and testing. In a subsequent step, a fast C-implementation is automatically derived by a just-in-time Compiler (JIT) [4].

For the numerical studies of the algorithm we present the problem of the space charge computation for a bunch in an 4-vane RFQ. Since this still is an open problem in the simulation effort e. g. of the Front End Test Stand (FETS) we have chosen the beam parameters of the $H^-$-ion beam of this facility [5]. The performance results are compared to a more simple shaped beam pipe with circular cross-section.

* poeplau@compaec.de

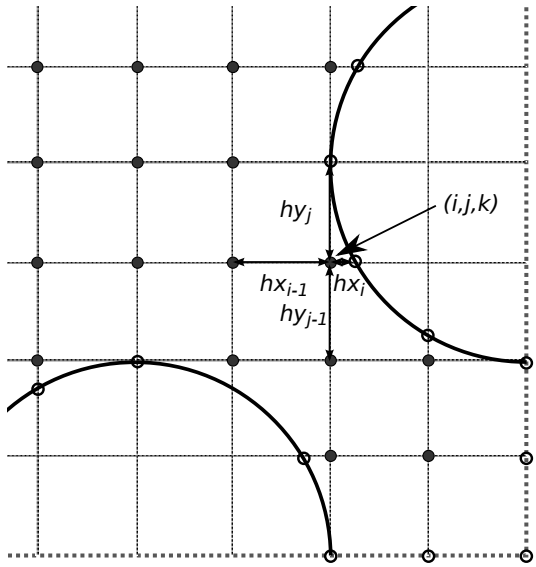


Figure 1: Schematic view of the transversal discretization of the region around the vane tips of an 4-vane RFQ. Close to metallic surfaces the 7-point-stencil, which discretizes the Laplacian, is modified using horizontal/vertical distances between grid point and surface.

## PYTHON POISSON SOLVER

The Python Poisson solver is based on the widely used particle mesh method [6]. Hereby, the Poisson equation is solved for the electrostatic potential $\varphi$ of the bunch:

$$\begin{aligned} -\Delta\varphi &= \frac{\varrho}{\varepsilon_0} \quad \text{in } \Omega \subset \mathbb{R}^3, \\ \varphi &= 0 \quad \text{on } \partial\Omega\,, \end{aligned} \tag{1}$$

where $\varrho$ denotes the space charge density, $\varepsilon_0$ the vacuum permittivity and $\Omega$ the computational domain, which is bounded by the walls of the structure and the setting of the computational bounding box. Dirichlet boundary conditions are assumed for the boundary of the structure $\partial\Omega$, i. e. the walls of the structure are supposed to be of a perfectly conducting material (PEC). On all other boundary parts of the computational domain the potential is set to zero, for instance in longitudinal direction.

In our approach the Poisson equation (1) is discretized by second order finite differences. This discretization leads locally to the following equation for the potential $\varphi_{i,j,k}$ at

ISBN: 978-3-95450-132-8 doi:10.18429/JACoW-IPAC2014-MOPME013

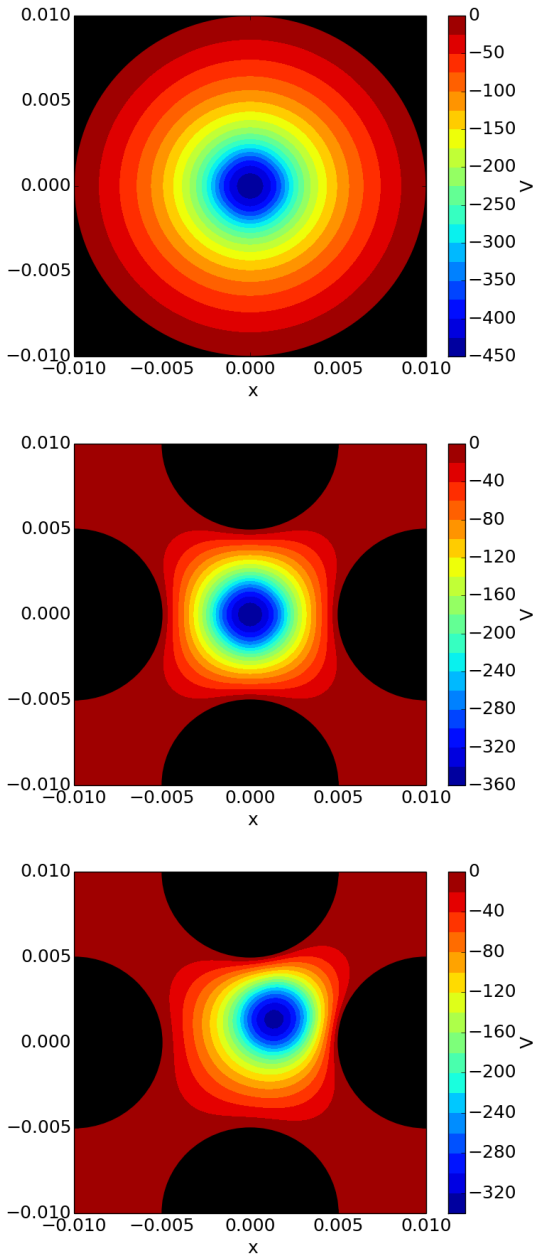


Figure 2: Potential of the same bunch in the center of a circular beam pipe (top), in the center of the vane tips of a 4-vane like RFQ (mid) and slightly off axis (bottom).

the point $(i,j,k)$:

$$
\begin{aligned}
& \tilde{h}_{y,j}\tilde{h}_{z,k}\left(\frac{1}{h_{x,i-1}}+\frac{1}{h_{x,i}}\right)\varphi_{i,j,k} \\
+\ & \tilde{h}_{x,i}\tilde{h}_{z,k}\left(\frac{1}{h_{y,j-1}}+\frac{1}{h_{y,j}}\right)\varphi_{i,j,k} \\
+\ & \tilde{h}_{x,i}\tilde{h}_{y,j}\left(\frac{1}{h_{z,k-1}}+\frac{1}{h_{z,k}}\right)\varphi_{i,j,k} \\
+\ & \tilde{h}_{y,j}\tilde{h}_{z,k}\left(-\frac{1}{h_{x,i-1}}\varphi_{i-1,j,k}-\frac{1}{h_{x,i}}\varphi_{i+1,j,k}\right) \\
+\ & \tilde{h}_{x,i}\tilde{h}_{z,k}\left(-\frac{1}{h_{y,j-1}}\varphi_{i,j-1,k}-\frac{1}{h_{y,i}}\varphi_{i,j+1,k}\right) \\
+\ & \tilde{h}_{x,i}\tilde{h}_{y,j}\left(-\frac{1}{h_{z,k-1}}\varphi_{i,j,k-1}-\frac{1}{h_{z,k}}\varphi_{i,j,k+1}\right) \\
=\ & \tilde{h}_{x,i}\tilde{h}_{y,j}\tilde{h}_{z,k}\varrho_{i,j,k}/\varepsilon_0 \,,
\end{aligned}
$$

where $h_{x,i-1}$, $h_{x,i}$, $h_{y,j-1}$,$h_{y,j}$, $h_{z,k-1}$, $h_{z,k}$ are the step sizes related to point $(i,j,k)$. For the transversal plane this is shown in Figure 1. The so-called dual step size $\tilde{h}_{x,i}$ is defined as $\tilde{h}_{x,i} = (h_{x,i-1} + h_{x,i})/2$; the dual step sizes in $y$- and $z$- direction are computed analogously. This difference equation is only computed, if a grid point is located inside the vacuum space. Points in close proximities to the PEC surfaces require a special treatment. If a point is located next to the conducting surface (e.g. the surface of a vane), the corresponding step size is reduced to weight the boundary effect at the surface properly. An example of this treatment is presented in Figure 1. In the update equation for the point $(i,j,k)$ the step size $h_{x,i}$ is reduced.

The update equation can be translated directly to Python (the code is shown as a direct translation of the update equation, no additional optimizations were performed to keep the example instructive):

Listing 1: Simple exemplary parts of straight forward implementation of the SSOR algorithm without optimizations.

```
def iteration_ssor(...):
...
  for k in range(1,p):
   for j in range(1,m):
    for i in range(1,n):
     if pipe[i,j,k] == 1:
      hx_dual = (hx[i-1,j,k] + ... )/2
          ...
      ax1 = hy_dual*hz_dual/(hx[i,j,k])
      ....
      U[i,j,k] = ( F[i,j,k] + Ugs)
```

The direct translation consists of multiple nested loops that update single field entries based on certain conditions. Unfortunately, the native Python implementation is too slow for practical usage. On an up to date PC (Intel Core I5-3210M, 2.5 GHz), a single iteration on a $[64]^3$ grid requires more than 7 s CPU time (one core). To speed up the execution, the just-in-time compiler Numba [4] was used. Simply by adding two lines of code, the Numba infrastructure can be used to generate automatically an optimized C implementation of the above function:

Listing 2: Using Numba to automatically build an C implementation

```
# Import numba module
from numba import autojit
# Use decorator to instruct numba
# to create a compiled function
@autojit
def iteration_ssor(...):
...
```

The decorator *autojit* instructs the just-in-time compiler to analyze the function during the first call. A C replacement function is generated and compiled in the background. All subsequent function calls to that function are replaced with calls to the compiled C function. In the case of the simple SSOR implementation the JIT-compiler generated an

ISBN: 978-3-95450-132-8 doi:10.18429/JACoW-IPAC2014-MOPME013



optimized function that reduced the execution time for one iteration on the $[64]^3$ grid used here from 7 s CPU time to 30 ms CPU time which is comparable to a native C implementation without the need to actually write low level code.

Using the JIT-compiler, two iterative Poisson solvers were implemented in Python: Symmetric Successive Over Relaxation with relaxation factor $\omega$ (SSOR($\omega$)) and preconditioned conjugate gradients with Jacobi precondition (PCG-Jacobi). For more details of discretization and solvers see [1, 7] and citations therein.

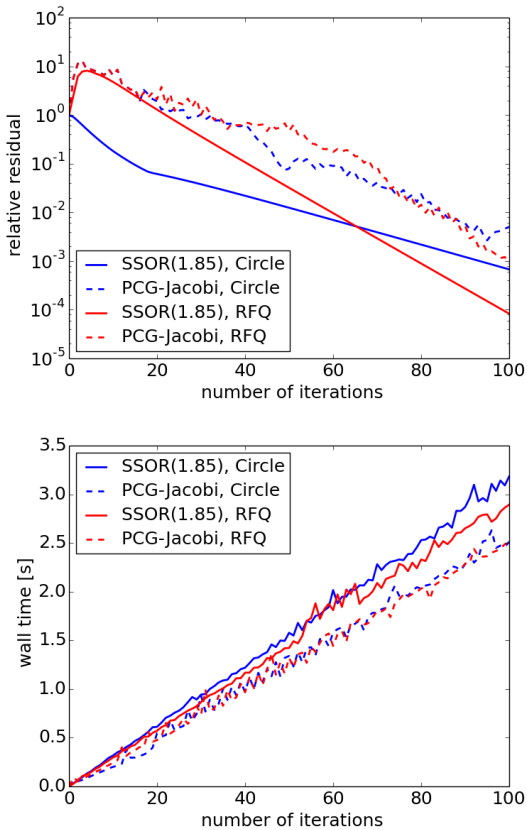


Figure 3: Convergence and performance of both Poisson solvers in both cross-sections.

## NUMERICAL RESULTS

The new Python Poisson Solver was investigated with a bunch within differently shaped structures: a circular beam pipe and the region of the vanes tips of a 4-vane like RFQ structure. The model bunch is a uniformly charged ellipsoid with the parameters of the $H^-$-bunch of FETS, i. e. the current was taken with $C = 60$ mA, the transverse radius with $r = 2$ mm and the pulse length with 2 ms.

Figure 3 compares the performance of the two solvers applied for the two different domains. It turns out that the SSOR solver with relaxation parameter $\omega = 1.85$ shows a better performance than PCG-Jacobi.

Figure 2 demonstrates the influence of the structure on the potential shape of the same bunch for a circular beam pipe, the region of the vane tips of a 4-vane like RFQ and a bunch in the same region but with a shift of 1.5 mm in both $x$- and $y$- direction. The electric field within this structure is represented in Figure 4.

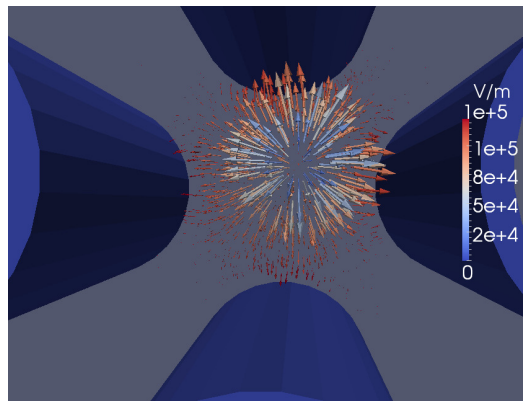


Figure 4: Electric field of a bunch in a beam pipe in the vicinity of the vane tips of a 4-vane like RFQ with a shift in both $x$- and $y$-direction of 1.5 mm.

## CONCLUSIONS

In this paper we presented a new Poisson solver for 3D space charge computations in structures with arbitrary shaped boundaries. The solver is implemented in Python which makes further adoptions quiet easy. To overcome the severe performance problems of a native implementation, a JIT-compiler was used to generate optimized C code for the computational intensive parts of the code. Thus the easy to write, read and maintain properties of the Python programming language can effectively be used to write performant numerical software that solves real world problems.

## REFERENCES

[1] A. Markoviḱ. *Simulation of the Interaction of Positively Charged Beams and Electron Clouds*. PhD thesis, Rostock University, 2013.

[2] G. Pöplau, A. Meseck, and U. van Rienen. Simulation of the interaction of an electron beam with ionized residual gas. *Proceedings of IPAC 2011*, 2250–2252, 2011.

[3] A. Adelmann, P. Arbenz, and Y. Ineichen. A fast parallel poisson solver on irregular domains applied to beam dynamics simulations. *J. Comput. Phys.*, 229(12):4554–4566, 2010.

[4] *Numba, Version 0.13*. `http://numba.pydata.org`, 2014.

[5] A.Letchford et al. Current status of the RAL Front End Test Stand (FETS) project. *Proceedings of LINAC 2012*, 846–848, 2012.

[6] R.W. Hockney and J.W. Eastwood. *Computer Simulation Using Particles*. Institut of Physics Publishing, Bristol, 1992.

[7] G. Pöplau, U. van Rienen, S.B. van der Geer, and M.J. de Loos. Multigrid algorithms for the fast calculation of space-charge effects in accelerator design. *IEEE Transactions on Magnetics*, 40(2):714–717, 2004.