Bachelorthesis

Optimisation of electric deflectors for a storage ring experiment

---

prepared by
Alexander Heinrich
at the I Physical Institute (IIIB)
in the
RWTH-Aachen

supervised by
Prof. Dr. Achim Stahl
and
Prof. Dr. Jörg Pretz

October 8, 2018

Abstract

In der Physik stehen noch sehr viele Fragen offen. Eine unter denen ist das ungleiche Verhältnis zwischen Matierie und Antimaterie. Eine Vermutung warum dies zustande kam ist die CPT-Verletzung. Daher suchen wir nun Teilchen auf denen eine CPT-Verletzung zutrifft. Ein Teilchen bei dem wir dies herausfinden möchten ist das Proton, indem wir dessen Dipolmoment bestimmen möchten. Dies versuchen wir mithilfe eines Speicherrings zu bestimmen, der kein Magnetisches Feld beinhaltet. In diesem elektrischen Feld kann sich das Proton, falls es ein Dipolmoment hat, ausrichten, so dass wir das erkennen können. Für diesen Speicherring gibt es jedoch auch hohe Anforderungen. Das elektrische Feld sollte keine zu großen Abweichungen vom Mittelwert haben. Daher ist es wichtig zu schauen, welche Maße die einzelnen Bestandteile, die für das elektrische Feld verantwortlich sind haben sollten. Aufgabe dieser Arbeit ist es, die Maße, sowie die Anzahl einzelner Komponenten numerisch zu bestimmen, sodass das elektrische Feld dem Mittelwert so wenig wie möglich abweicht. Dabei werden verschiedene Programme und Programmiersprachen verwendet. Unter anderem Agros2D, um ein Modell des Speicherrings zu erstellen und dessen E-Feld berechnet. Andere Programiersprachen, die hierbei benutzt wurden sind C++ mit der ROOT Bibliothek, sowie Python mit weiteren Paketen, wie zum Beispiel numpy. Diese weiteren Programme sind in der Lage, das elektrische Feld, welches in Agros2D berechnet wurde auszuwerten und die Abweichung vom Mittelwert zu bestimmen. Ein Pythonskript wurde geschrieben um eine grafische Darstellung der Abweichung abhängig von verschiedenen Parametern zu erzeugen.

In physics many questions are still open. One of these is the unequal ratio of matter and antimatter. One assumtion why it became that way, is the CPT-Violation. Therefore, we now search particles which have a CPT-Violation. One possible particle where there could be a CPT-Violation is the proton. We want to examine it by looking for a dipole moment inside the proton. This can be done with the help of a storage ring which works without a magnetic field. In this electric field the proton can align if it has a dipole-moment that can be measured. But this storage ring has high expectations. The electric field should be as constant as possible without much deviations. Therefor we have to look at all measures of elements in the storage ring which are responsible for the elecric field. The goal for this thesis is to numerically detemine the number and the measurements of the different elements, so that the deviation of the field is as small as possible. By doing this, we use different programs. One program is Agros2D. It is able to create a model of the storage ring and can calculate it's electric field. Other programming languages used are C++ with the ROOT library and Python with some packages like numpy. These other programs are written for examining the field calculated from Agros2D and to see how the field deviates from the mean value. One Python script is written to have a graphical presentation of the deviation, dependant from different parameters.

Contents

1 Motivation

At the beginning of the universe, the number of matter was equal to the number of anti-matter. Now we know that the matter dominates. One assumption why this happens is the CPT-Violation. A number which can give us information about the assymetrie of matter and antimatter is the baryon asymetrie:

$$\nu = \frac{n_B - n_{\bar{B}}}{n_\gamma} \tag{1}$$

The experimental result therefore is much higher than the theoretical result. This means there are more particles which have a CPT-Violation than we know. Therefore we try to find them by finding some properties of barions which can give us information about the CPT-Violation. One information that gives us a hint about the CPT-Violation is the dipole moment inside a barion like a proton or neutron. After finding out that the neutron doesn't have a dipole moment, we now try to find a dipole moment inside the proton. The problem with the proton compared to a neutron is that it has a charge. We can't simply take a proton in an electric field we can do it with the neutron without accelerating it. But we can take the proton inside a storage ring with an electric field and see if the proton will align or not. If it aligns, that can tell us that the proton has a dipole moment. This storage ring contains no magnetic field, because this can influence the spin. Only an electric field which extends to the middle of the ring like a cylindrical capacitor. So accellerated protons in there are held in this ring by the force of the electric field. Now when protons have a dipole momentum they will align in the field like other dipoles. Then the poton aligns, the spin will align with it and we are able to measure the spin. So we can find out if there is a dipole moment or not. If there is one we found another particle with a CPT-Violation.

2 Physical Basics

2.1 Electric Field

By describing the strength of the coulomb force in a physical system we have to use a quantity. Therefore the electric field stregth was defined. It could be defined the following way: The electric field strength $\vec{E}$ is the coulomb force of a test charge divided by the charge of the test charge. It's unit is $\frac{\vec{F}}{C}$ or Vm. The electric field gives us information about the electric field stregth in the whole sytstem. In a statical system which means no current and no chage of the system dependant from the time, the electric field stregth can be described by the first Maxwell equation:

$$\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\epsilon} \tag{2}$$

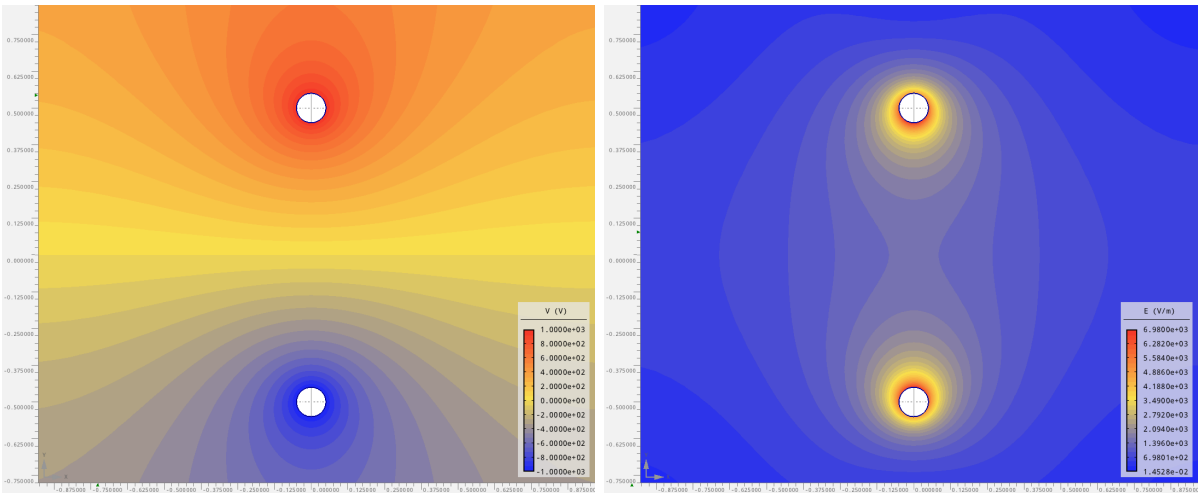
2.2 Elektrical Dipole

An electrical dipole is an arrangement of two opposite charges. An electrical dipole can be descriped by the electric dipole moment $\vec{p}$ which is dependent of the electrical charge q and the distance between the charges $\vec{d}$.

$$\vec{p} = q \cdot \vec{d} \tag{3}$$

For electrical dipoles you can differentiate between mathematical dipoles and physical dipoles. A physical dipole has a finite charge and a finite distance. In a mathematical dipole the distance is infinitisemal small and the charge infinitisemal big but has a finite electric dipole momentum. Like others object with a electrical charge, the electric dipole will also be influenced in an electric field. Therefore we look what happens with a free mathematical dipole in an electric field. Because the sum of the charges in a dipole is zero, the dipole will not change it's translation.

But there will be a rotation. The dipol will turn along the lines of the E-field so that $\vec{p}$ and $\vec{E}$ are parrallel. The difference between a mathematical and a physical dipole in an electric fiel is, that the physical dipole can change it's translation in an inhomogeneous electric field.
For this thesis mainly the caracteristics of the physical dipolefield between the two charges are important. The equation of the field inside the dipole can be described with the coulombs law



(a) Potential of a electric dipole (b) Elektric field of a electric dipole

Figure 1: Potential and electric field of an electric dipole

and for two charges q and -q, it is:

$$E(r) = \frac{q}{4\pi\epsilon} \cdot \frac{4}{2r^2 - d^2} \tag{4}$$

where r is the distance from the middle of the dipole and d is the distance between two dipoles. When you have many parrallel dipoles the arrangement can be seen as a planar capacitor, so the dipole field is constant.

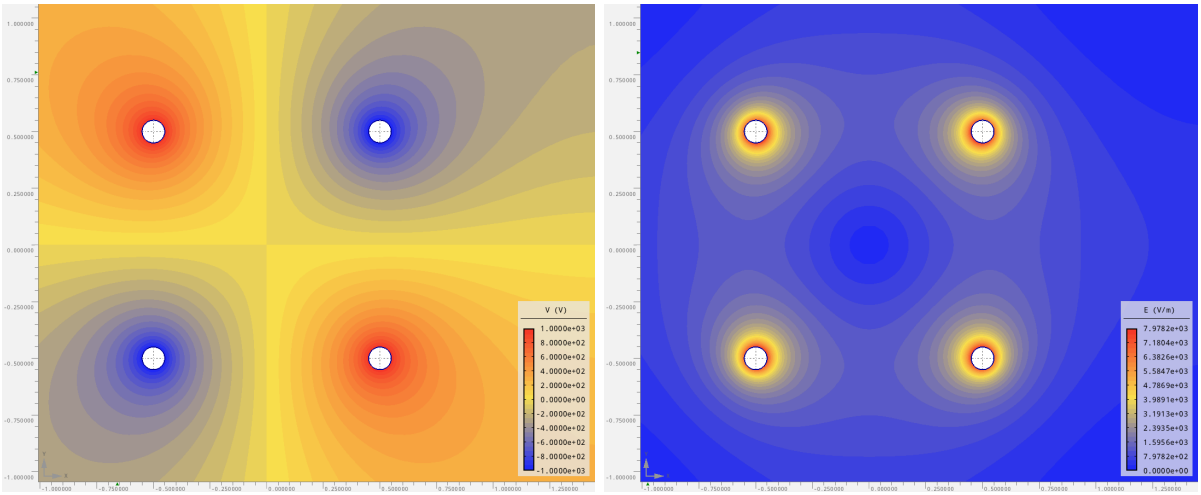
2.3 Elektrical Quadrupole

The electrical quadrupole is an arangement of two dipoles. The particles in a quadrupole are disposed so that the momentum of the two dipolemomentums are antiparrallel. You can see the field and the potential in figure 2

Also the Field inside can be described with the coulombs law by the sum of the electric field strength of each charge. Also with a quadrupole there is more than one kind of quadrupole. The mathematical description where the quadrupole is a point and the physical where the quadrupole is a finite object. By using plates instead of pointed charges you also can create a quadrupole. Compared with the dipole which consits of plates, the field of this quadrupole isn't constant. The x-component of the field increases linear in x-direction and the y-component of the electric field increases linear in y-direction.

3 Basics in Programming

Programms, how they are stored in the hard disk drive and how the cpu reads them are very difficult to write and to understand. So in the early times when computers were invented people tried to make it easier for humans to write programs. First by using mnenonics in an assembler language, where a single word represents a specific byte or some bytes in the program. For converting the more readable language to the binary code assemblers were created. The advantage of this languages is the possibility to tell the computer exactly what to do. So it is a



(a) Potential of a electric quadrupole (b) Elektric field of a electric quadrupole

Figure 2: Electric Quadrupole

good language for writing kernels and drivers and also today these languages are used for specific cases. The disadventage is the dependancy of the operating system, the processor-architecture and also the librarys you use. Also the error-rate can be higher compared with some other programming languages.
Later programming language like Fortran and C were developed. They are called the higher programming languages where every line can represent many bytes of the assembler language. There also came the possibility to define variables classes and different kinds of loops. They used a compiler to convert the program code into binary code. An other advantage of the compiler languages are that they are mostly platform independant. So with the exception of some librarys the code in the different operating systems is the same.
The next generation of programing are interpreter languages like python. This languages were mostly designed for more simplicity and doing more complex things with fewer lines of codes. They offer many librarys for all kinds of operations. They were called interpreter language, because they don't need a compiler. The main advantage is the ability of very exact calculations without having to take attention to memory management. The disadvantage is the efficiency of these languages which run slower than compiler languages.
Programming languages are used in all kinds of sectors. In physics, languages like python or C++ are often used. C++ mainly when you have much data to examine and Python for doing many things with a few lines of code.

3.1 Python

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.

Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one– and preferably only one –obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!

How you can see in the Zen of Python, python is designed for simplicity and readibility. The Zen of Python says in general that the scrips should be written so that it is as simple as possible to understand the code. The Zen of Python says that there is only one way to do it. Python should be an easy to learn language. It is easier to learn one way than many ways.
Python is a multiplatform, opensource, simple and modern programming language. It is usable for objectorientated, functional but also aspectorientated programming. It is a interpreter language so it doesn't need a compiler to convert the script into machine language. So after you have written the code, you only need to excecute it. You don't have to take care of the exactness of each number. Python will take care of it and it can store numbers of a very wide range with a very high exactness. Even without packages, it contains a lot of build in functions and classes. For more complex operations it is possible to import packages. For Python there are many packages for all kind of programming. The most important packages used for this thesis are numpy, matplotlib and re.

3.1.1 numpy

Numpy is a very usefull package which is mainly used for arrayoperations. It contains the n-dimensinal array structure. In this structure the whole array is saved in one chain in the memory, instead saving the elements in separate regions of the memory. Like this, array operations are faster in a nd-array than in a Python list. Numpy offers many vector and matrix operations for linear algebra like vectorproduct or matrix multiplication. It also offers to create arrays in a special form like creating arrays filled with zeros or ones. The created arrays can be saved in an ascii file. Ascii files in tabular form can be read from numpy and saved into an array. Other functions of numpy are the creation of random numbers or the fourier transformation. For integration of C/C++ or fortran code in Python, numpy can also be used.

3.1.2 matplotlib

Matplotlib is a python package to generate all kinds of plots. It is able to generate two dimensional plots like a simple line plot or histogram or heatmaps, but also animated or three dimensinal plots. You can set a title, describe the axis, add some text in the plot and also make a legend. You can also draw elements in the graph like arrows or circles. Matplotlib is also able to set the scale of the axis like you want. Matplotlib has a very detailed documentation how to use all the classes and functions of matplotlib. So it is possible to understand this package with only some basic knowledge of Python.

3.1.3 re

The name re stands for regular expression. Like the name says the package helps to find structures in a string. It only needs a pattern which you have to define. If re find this pattern in the string it is able to return the part of the string where it founds the pattern but also the parts of the string. You can use it after reading from an ascii file to extract some values or other information of this file. For the thesis it is used to find the values of an ascii file which Python

opened. The advantage of using re compared to numpy is that it is able to find the values of a more complex ascii file than a tabular.

3.2 C++

C++ is a very fast, efficient, open-source and multiplatform compilerlanguage. It is invented by Bjarne Stroustrup in 1979 as a successor of C. In C++ you have a lot of control of what the computer is finally doing. You also have some control over the memory for example how much memory is saved for a variable, or reserve memory for an array. Therefore it is suitable for systemprogramming like operation systems, drivers and also programs which have to run as fast as possible. It is easier than the assembler language which was used for systemprogramming and drivers before the developpment of C/C++, but with an optimized compiler, it is nearly as fast as an assembler. In C++ you can acces a lot of functions of the standard C++ library which are often usefull like iostream(reading and writing to the terminal), fstream (reading and writing to a file), cmath (mathematical functions), vector (a kind of array which supports many array operations) and a lot of others. It is also possible to implement other library in C++ like the ROOT library, which is also used for this thesis. But this control also means responsability. Even if the compiler do not give an error that doesn't mean that the program runs without problems. It can produce a segmentation fault in C++ very fast for example when you reserve less memory for an array than you need. In this case the compiler doesn't give any error but the program will not exit succesfully. Because of this you have to take into account all possible cases which can happen and perhaps use exception handling in the script.

3.3 ROOT

ROOT is developped by CERN for data Analysis. It is an open-source project written 1994 in C++. Root can manly be used as a python library or C/C++ library but other programming languages are also supported. It can also be used as an own language which supports the C commands with the root extensions. Root is able to generate plots from arrays or ascii files. It is also designed to create 2d or 3d plots and histogramms. These plots can be fitted to every function with and also without predefined parameters. Every plot and fit can be exported as a pdf file.

3.4 bash

Bash (Bourne-again shell) is a scripting language which can be used to automatisate some processes. Bash is mainly used in unix like systems for example linux or macintosh but there is also a equivalent for windows computers. The main adventage in bash is the ability to launch other programs within this script as a main thread or a subproces. It also supports variables, loops, conditions functions and other things. In general it is simple to use even without much knowledge, it is possible to write simple scripts. Unlike python or C++ bash is designed for straight forward programming. So the scripts are very readable. Bash is a good script for automatisation processes or to connect some programs, so that one is excecute after one other. You don't need to always be there to launch every single programm when one is finished. Bash supports the commands from terminal but it also has it's own commands. You can also use regular expressions in bash when you will do one operation with to many files. In the thesis we are using bash but also sh which is similar to bash.

3.5 Agros2D

Agros2D is an open source application for Windows and Linux operation systems. It is designed to numerically calculate properties of a system like the electric field or the temperature of a

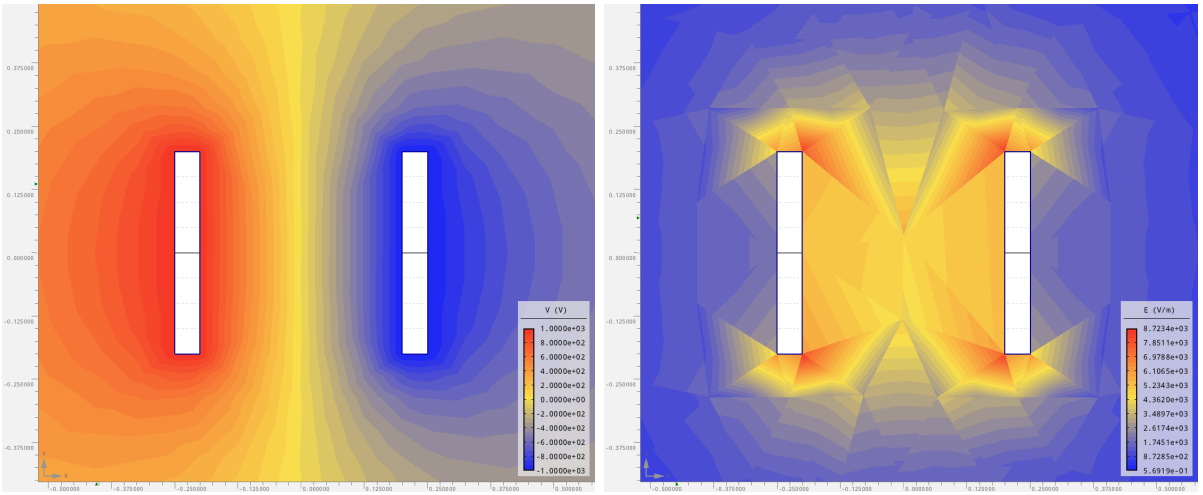
system. It uses many librarys and applications like dealii or Hermes. It supports graphical usage but also python scripting with it's own library. But there are also some limits in this application. The most important limit is the ability to only calculate the electric field in two dimensions. The other limit which is to take in account is, that the numerical calculation isn't exact.
For the thesis, this application was used to calculate the electric field of the system. Because of the complexity of this system, it is easier to write an python script than using the graphical surface. The python script also has the advantage of using python functions like numpy in this script to use the ability to save the results of the electric field in an ascii file.
We will now treat two examples to get a better understanding in Agros2D

3.5.1 Planar capacitor

For a better understanding Agros2D calculating the electric field of a planar capacitor in Agros2D is a good excercise. Therefore we have to do the folowing:
First creating a world volume. In the end it is the place where Agros2D finally calculates the electric field. So it should be big enough that the two plates matches inside the volume and also that the influence of the plates on the border of the Volume are not to be considered. We begin by placing four nodes at the edges of the world volume. Then we use the edges to connect the edges that it looks like a quadrangle. Then we create two plates inside this square. Also we begin with the four nodes for each plate and connect them with the edges. Then we use the labels and create a label in every square. Now we have the structure of the system.
What we need to do now is creating boundary conditions and materials. By right clicking on the surface of the preprocessor you can chose new boundary conditions. In a new window that appears you can choose between fixed voltage and surface charge density and give the boundary condition a name. By choosing fixed Voltage you are also able to set this voltage to a specific value. For the plates we add 2 new boundary conditions with fixed voltage. One Voltage for the one plate and a different voltage for the other plate. For the border of the world volume we create a third boundary condition where we set the first option to surface charge density. This is the nearest option to treat the border like it doesn't exist. The other possibility to set the boundary condition to a fixed surface and set the voltage equal zero. But therefore you have to choose a bigger world volume that this border doesn't influence the electric field near this border. Next comes the materials. First we add labels inside every square. In our example these are the square of the world volume and the sqare which demonstrates each plate. But we only need to link the label inside the worldvolume with a material because we aren't interessted in the field inside one of the plates. You can add a new material by rightclicking on the surface and choose new material. In the widow that appears you can give the material a name define the dielectricum and define a charge density. Because the world volume is filled with air, we leave the dielectricum at 1 and the charge density at 0 (default values). Then the system is defined. By clicking on solve Agros2D will do the rest. The result with the electric field and the potential can be seen in figure 3

For defining a system in Agros2D you also have to take attention to the folowing things:

- Every edge must have a boundary condition if not Agros2D will give an error.
- It is not possible to connect the nodes with only one edge. Every edge must have a closed surface if not there will be an error
- Not every label must have a material but in every field must be a label.
- Agros2D will not calculate the electric field inside a field where isn't defined a material.



(a) Potential field of a planar capacitor (b) Elektric field of a planar capacitor

Figure 3: Potential and electric field of a planar capacitor

- You are able to generate curved edges by right clicking on the edge and chosing object properties. There you can change the value of angle to get a curved edge. By clicking on swap directions (above object properties) you can change the direction of the curve.
- Agros2D reserves only a limited amound of memory for the calculations. So there are limits of the complexity of the system you define. But by decreasing the number of refinements and the polynomial order in the properties tab there is a change to calculate more complex systems. But by decreasing this two parameters, the calculations are less exact.

The advantage of the graphical method to generate a system are the following:

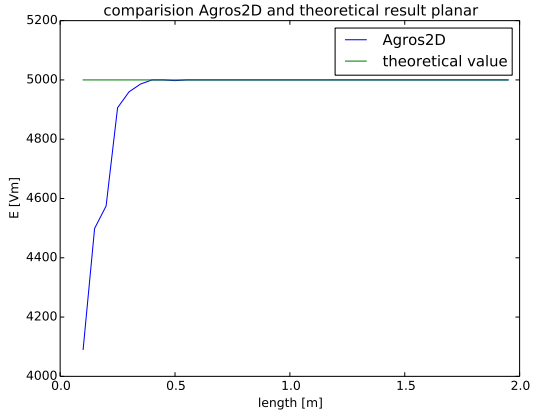
- It is easy to use and it doesn't need much time to learn. It don't requires programming knowledge.
- You can see direcly how you have defined your system. And therefore you can see very fast if there is error.

An other method to use Agros2D is by python scripting. Therefore Agros2D offers his own developpment environement. This environement offers the usage of converting a graphical created system into a python script. There you have the possibility to also use the python functions. The adventage of python scripting are the folowing:

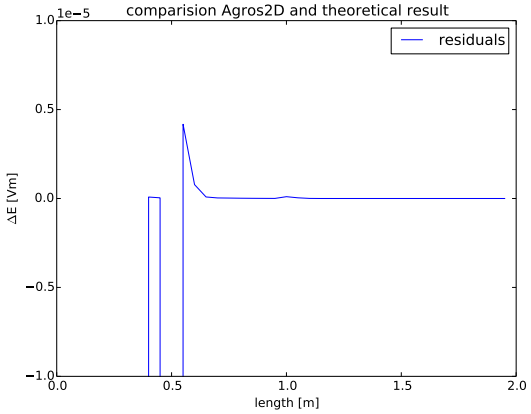
- Imoport of other package like numpy or matplotlib
- Usage of variables, loops and functions from Python
- Automatisation of the script
- Faster to generate complex calculations
- Possibility to generate more than one system in one script
- Possibility to extract values and export them in an ascii file with the usage of numpy
- Usable for comparision of the theoritical value with the calculations of Agros2D

With the formula of the infinite planar capacitor

$$E = \frac{U_1 - U_2}{d} \tag{5}$$



(a) Calculation from Agros2D vs the theoretical calculation



(b) Theoretical value minus Agros2D

Figure 4: Electric Field of a planar capacitor between the plates

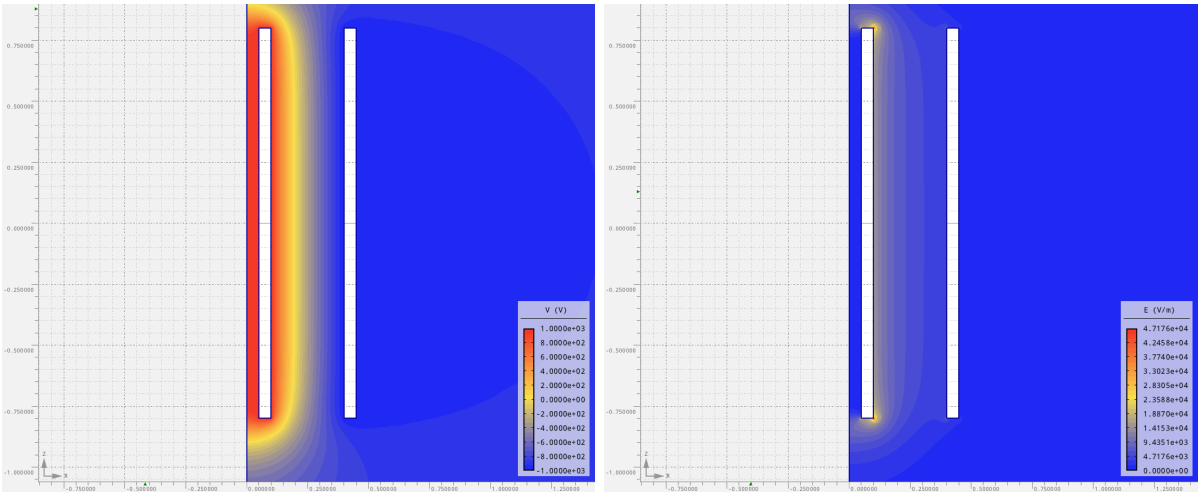
we can calculate the theoretical value and compare this with the result of Agros2D. We have an electric fiel strength of 5000Vm. By increasing the length of the plates the Agros2D field converges to this value. So we can see that we have defined our system the right way. With the help of the matplotlib library we are also able to display the result graphically. In figure 4 you can see the electrical field strength in the middle of the capacitor dependant from the length of the plates. By electric field strength increases and converts to the theoretical result.

3.5.2 Cylindrical capacitor

The cylindrical capacitor is similar to write. Because of the capability of Agros2D to only draw in 2 dimensions, you are able to draw the capacitor on 2 different ways. One way is to draw in the x-y-plane the other way in the r-z-plane. Because of the radialsymetrie of a cylindrcal capacitor, we choose the r-z-plane to also see the field at the edges of the capacitor. Also like the planar capacitor we beginn by drawing the system and the extend this system by converting it into a python script. The biggest difference between the planar capacitor and the cylindrical capacitor in Agros2D is the use of axisymmetric coordinates. The coordinates can be changed in the properties tab. Then you can see in the preprocessor tab that you now have the r and z axis and you aren't able to place nodes, edges or labels in the negative r-area. Then you have to create the cylindrical capacitor in the same way like the planar capacitor. You can see the result with the electric field and the potential in figure 5. By looking at the solved system you can see that the field decreases with the distance from the inner plate, which also says the theory of a cylindrical capacitor.

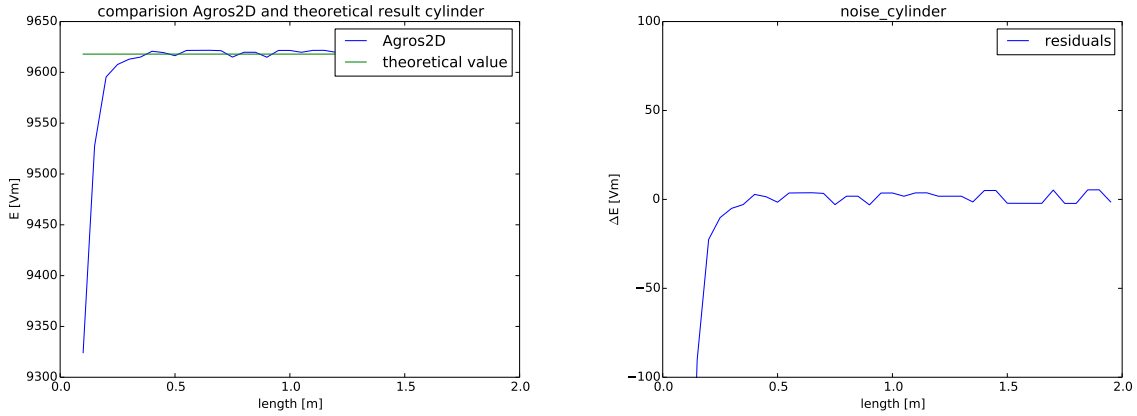
$$E(r) = \frac{U_1 - U_2}{r \cdot ln(\frac{r_2}{r_1})} \tag{6}$$

Also there we compare the result from Agros2D with the theoretical result in figure 6. But compared to the planar capacitor the result differs much more from the theoretical result also by increasing the length of the plates. The reason of the difference of the result comes from the error of the numerical calculation. By increasing the number of refinements and the polonical order you can see that the final result converges to the theoretical calculation. But now it takes more time to calculate. The difference from the theoretical result on nearly infinite plates gives us information about the accuracy in Agros2D. For this thesis the information about the accuracy is espessialy important for the final system at which we will look later.



(a) Potential field of a cylindrical capacitor (b) Elektric field of a cylindrical capacitor

Figure 5: Potential and electric field of a planar capacitor



(a) Calculation from Agros2D vs the theoretical calculation (b) Theoretical value minus Agros2D

Figure 6: Electric Field of a planar capacitor between the plates

4 The storage ring in Agros2D and examining the results

In this section, we will create a model of the storage ring in Agros2D, calculate it's electric field and put the results from a previously defined box into an ascii file. With C++ and ROOT we can plot this data and fit a linear function to it. The reason of a linear function is that with the Quadrupole part the function has to increase linear and the dipole part should give an electric field with a constant value. This is also the optimum which we want to reach. The plot in ROOT should match as best as possible a linear function. We can reach this by changing some parameters in the Agros2D script.
With the fit, we can calculate the difference for each value and the value of the fit in this position. The standard deviation for all these values is the number which gives us information about the proximity to the optimum. So we need to minimize this one value. Another advantage of having only one value of how good the values matches is that we can display many of them graphically dependant from some system parameters. This will help us finding the minimum. Therefore we will also write a short python script which reads all values and plots them.
In the following sections the steps are described more detailled.

4.1 The physical system

The physical system is a model from the storage ring written in Python with Agros2D (see also in the annotations). The model in Agros2D mainly looks like this:

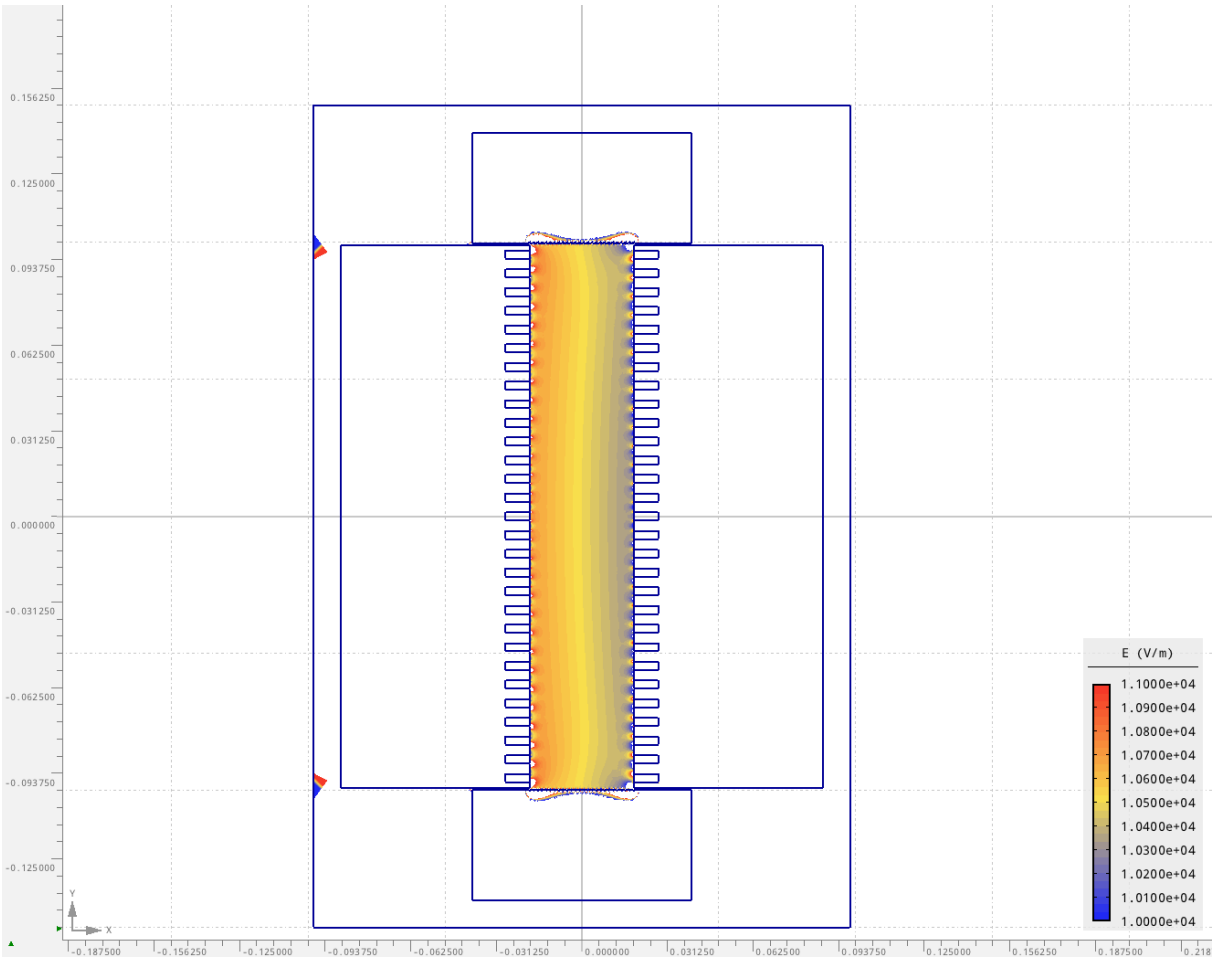


Figure 7: The storage ring model in Agros2D

It contains the vertical stripes and the horizontal wires. This stripes and wires all have a voltage, so the field insde the stripes looks like a dipole field or a Quadrupole field. That

depends on the values you set in the script. We use a dipole part and a quadrupole part, so we can take care of both in one run. You can also see the blocks outside, but they do not have much influence on the final field. The first problem with this script was the complexity of the designed structure. By trying to excecute this script the first time, there a memory error occured. Some parameters were changed like the number of refinements and the polonomical order, so that the script runs without problems. The disadvantage is that Agros2D doesn't calculate it very excacly. We will later see something about the uncertainty of the electric field of this parameters. After the system is solved, the python script grabs the data of the x-and y-component of the electric field in a defined surface inside the stripes and wires and saves them into an ascii file with the coresponding x-and y-values.

4.2 Chosing the parameters

By looking at the system in the annotations you can see many parameters. But with some parameters you already know that they do not have a big influence on the electric field and some are dependant on each other. So we don't need to examine them all. We chose to examine the parameters of the stripes and wires (Deflector geometry in the annotations). For knowing exactly which they are, we need to look at the python script.
The python script is written, so that you can simply change some values and excecute the script. By looking at the system you can also find what parameters can change the electric field, and what parameters don't do a lot of changing or they scale only the electric field.
So the following parameters were chose as variables that will be tried to optimize (See figure 8).

- ratioVS (ratio of vertical stripes) it is the ratio between the length of one stripe divided by the length of the stripe and the gap. So with ratioVS=1.0 there is no gap and with ratioVS=0.0 there are no stripes
- dVS (thicknesl of vertical stripes)
- hVS (height of vertical stripes)
- NVS (Number of vertical stripes)
- NHW (Number of horizontal wires)
- rHW (radius of horizontal wires)

These parameters all have some limits. One is the technical limit the other one is the limit from Agros2D. For example Agros2D canot create a mesh when some edges are overlapping. The technical limits are for example limits for the thickness of the material. They need to withstand the current and other physical factors.
Now we have to run the script and changing the values, so that we finally find the best parameters possible. There we have to take attention to the following things:

- how we can save time by automatisating the script
- the fineness of changing the parameters (finding the parameters more excacly or saving time)
- trying if possible as many combinations as possible

With this points, I've chosen to only change two parameters for one series of measurements and let the other ones stay constant. Later I will do the same with two other parameters in an other series of calculations. By using only two parameters you have a compromise between time consuming and trying as many parameters as possible. By naming the ascii file where the results are saved with the parameter values, you can later see which file has which parameters.

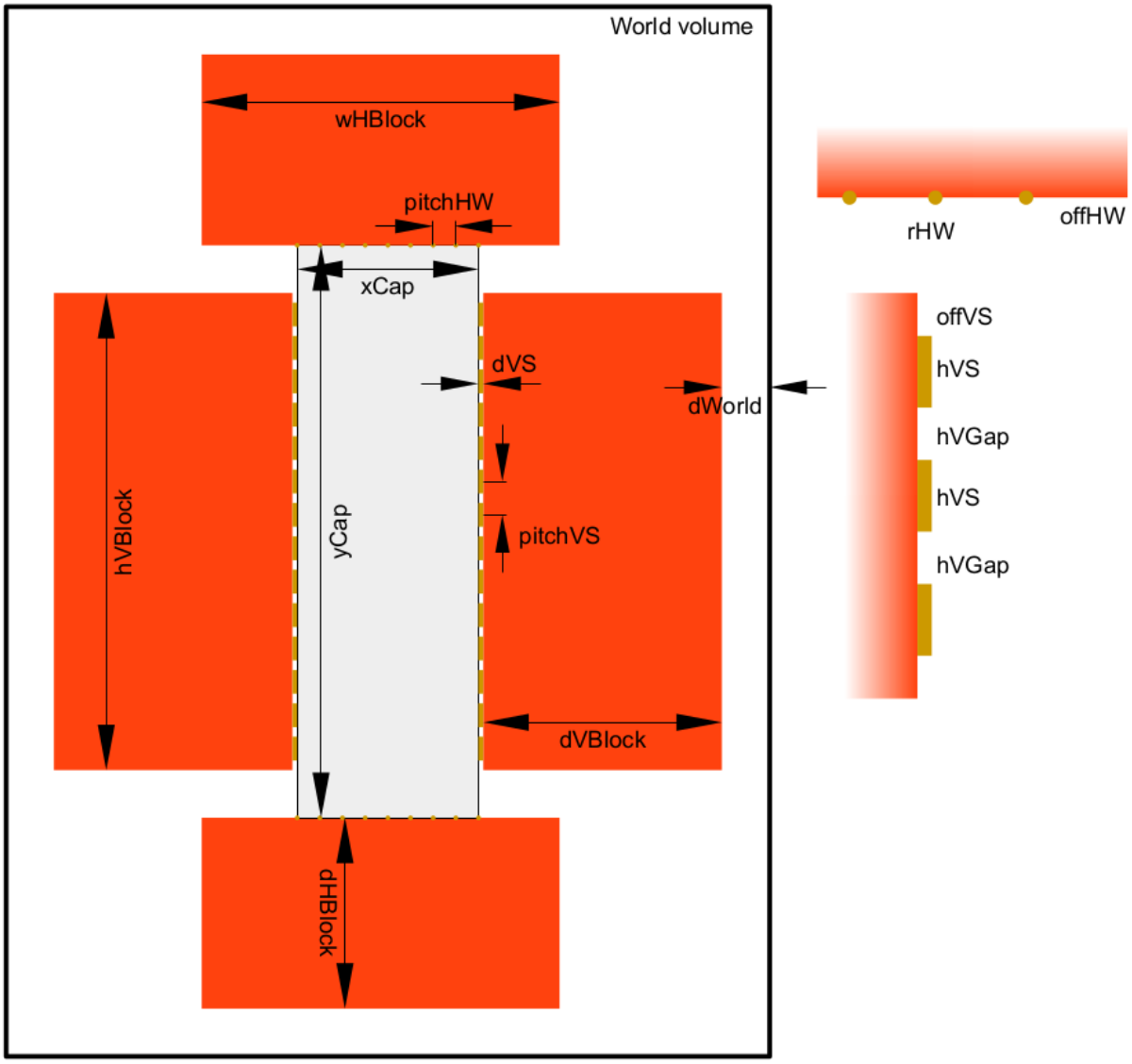


Figure 8: geometry in the deflector (see also in the annotations)

4.3 Fitting, Residuals and deviation of the Residuals

Now that we have the ascii file which contains all important information about this measurement we can write a program which reads this file and calculates the deviation from an optimal E-field. For writing this program a language like C++ linked with ROOT is a good choise because of the efficenty of C++ and the ability of ROOT to create a fit and calculate the Residuals.
In this program we begin by reserving space for four arrays x, y, Ex, Ey which are in the ascii file. We also reseve space for saving the number of lines in this file and the number of different x and y elements. So by reading the file which we place the filename in the arguments of the program we can fill the arrays with the corespanding value. We also write the number of lines and the number of different x and y values in the appropriate reserved space. So we have stored all information from the file, so we can finally close it.
Next comes the part where we are using the root library. With the number of x and y values and the minimal and maximal value, we create two empty histograms. The neseccary arguments of the Th2f function which defines an empty histogram are the name of the histogram the name of the axis, the borders of the axis and the number of bins for each axis. One for the Ex values and one for the Ey values. By defining the histogram it is important to shift the borders from x and y a half bin to the outside of the histogram. If we don't do it some Ex and Ey values will be in the same bin. The reason of this is that by defining a 2D histogram the function sees the borders as the borders of the bin which it is not in the ascii file.

```
52        //define borders
53        float xmin=*min_element(x,x+bins[0]);
54        float xmax=*max_element(x,x+bins[0]);
55        int xbins=bins[1];
56        float ymin=*min_element(y,y+bins[0]);
57        float ymax=*max_element(y,y+bins[0]);
58        int ybins=bins[2];
59        float dx2=(xmax-xmin)/(2*(xbins-1));
60        float dy2=(ymax-ymin)/(2*(ybins-1));
61
62        //alignement:
63        xmin=xmin-dx2;
64        xmax=xmax+dx2;
65        ymin=ymin-dy2;
66        ymax=ymax+dy2;
67
```

Figure 9: part of the program thats aligns the histogram so, that each value is in the middle of a bin

With the functions findbin and setbincontent, we can add the z values Ex and Ey to the histograms. By saving the histgramm into an pdf-file we are able to see the resulting plot. Then it is a good time to compile and excecute the program and looking at the resulting plot. This is a necessary step to compare the result with the result in Agros2D. When there are some errors, it is better to look for the right binning or to see if there are other errors. A good hint for a wrong binning are holes in the plot where the value there is zero. That often means that there was never a assigned value to this bin. By replacing setbincontent with addbincontent, you are easily able to see if a bin was found twice. Another possible source for an error are the numbers of different x and y values counted by reading the file. Also you can print the two numbers and see if they are the same as the defined number in Agros2D. When the histogram was defined right, it should look a bit like a dipole field combined with a quadrupolefield. Some sinus-like deviations at the border of the histogram can also be seen in Agros2D and are normal. Now that we know that everything is defined right, we can continue by fitting a linear 2D function to the histogram. Root has some predefined functions for linear fitting, but mostly they are defined in one dimension. I have chosen to defining the function myself with the expected parameters. Then you can call the fit-function. Now it is a good time to also write the plots with the fits in

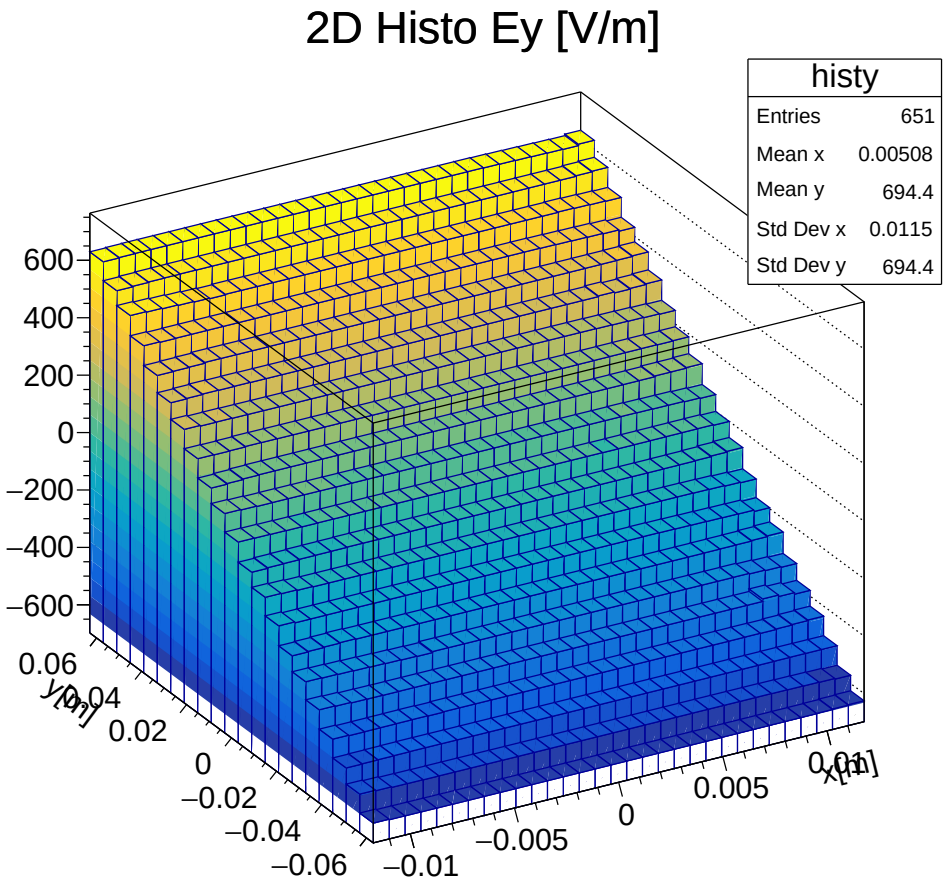


Figure 10: Y-component of the electric field from one sample

a pdf-file, compile the program and excecute it. The fit-function gives us information how the fit runs and if there was an error or the fiting finished without errors. We also get information about how the fitting finished. By looking at the plot we can also see if there was a gross mistake.
By calculating the residuals and plotting them you can also see if the fit runs without erros. The residuals aren't difficult to calculate. You make a loop where you search the corresponding bin for each x and y value. Then you can use the getbinvalue function to get this value. With the eval function you can get the corresponding value from the fit-function. Now when you have calculated the residuals you can do many things with them. The first thing is to make a residualplot and see how the fitting runs. Another thing is to store the residuals and create a 1D histogram with it. And the most important thing is to calculate the standard deviation of the residuals. This is the value which should later be minimized. In the residualplot you can

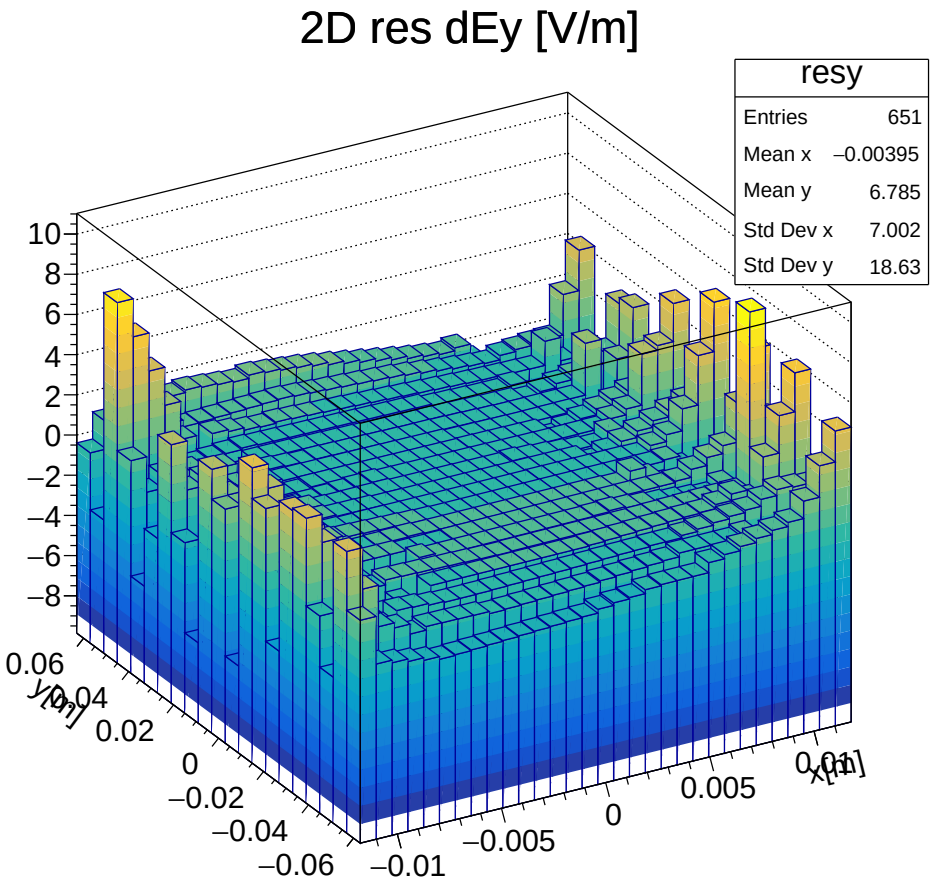


Figure 11: Residuals of the Y-component of the electric field from one sample

see that the electric field isn't optimal and you can see some structures in them. The sinus like behavior at the border is one of them and when you compare it with the Agros2D result, you can see the same behavior.
By looking at the 1D histogram you can also control if there was an error. The mean value of the histogram should be around zero. If not we have to control our code another time.
The last thing we need to do in this program is to calculate the standard deviation and print out the standard deviation. I've chosen to print out the filename which contains all the parameters and in the same line the standard deviation of each electric field. So when we excecute the program we get an output of the two fittings from the root function and in another line we get the filename and the standard deviation.
This program should be designed to read only one ascii file and calculates the standard devi-

```
 1 FCN=1.31751e-07 FROM MIGRAD    STATUS=CONVERGED      41 CALLS          42 TOTAL
 2                     EDM=1.27612e-29    STRATEGY= 1      ERROR MATRIX ACCURATE
 3  EXT PARAMETER                                   STEP         FIRST
 4  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
 5   1  p0          -2.48015e-03   3.91931e+02   1.90936e-01  -1.01894e-19
 6   2  p1          -3.04026e-01   5.76568e+04   3.00000e-01   8.75709e-20
 7   3  p2           1.05011e+04   1.07875e+04   2.49190e+00   1.54821e-20
 8 Calc_ratio0.435000_dVS0.008900_hVS0.009000_NVS29.0_NHW24.0_rHW0.000202_EDipole10500.0_kx10500.0.txt Ex= 0.107565 Ey= 0.142263
 9 FCN=7.56369e-08 FROM MIGRAD    STATUS=CONVERGED      41 CALLS          42 TOTAL
10                     EDM=3.26955e-29    STRATEGY= 1      ERROR MATRIX ACCURATE
11  EXT PARAMETER                                   STEP         FIRST
12  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
13   1  p0           1.04992e+04   3.91931e+02   1.90936e-01  -1.48994e-18
14   2  p1          -1.05040e+04   5.76568e+04   2.81426e+00  -2.80757e-20
15   3  p2           7.12477e-02   1.07875e+04   3.00000e-01   7.32441e-19
16 FCN=7.47339e-08 FROM MIGRAD    STATUS=CONVERGED      41 CALLS          42 TOTAL
17                     EDM=1.38872e-29    STRATEGY= 1      ERROR MATRIX ACCURATE
18  EXT PARAMETER                                   STEP         FIRST
19  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
20   1  p0           1.04992e+04   3.91931e+02   1.90936e-01   3.23965e-18
21   2  p1          -1.05046e+04   5.76568e+04   2.81426e+00  -4.10554e-20
22   3  p2           7.27478e-02   1.07875e+04   3.00000e-01   4.20318e-19
23 FCN=1.30458e-07 FROM MIGRAD    STATUS=CONVERGED      39 CALLS          40 TOTAL
24                     EDM=8.1202e-29    STRATEGY= 1      ERROR MATRIX ACCURATE
25  EXT PARAMETER                                   STEP         FIRST
26  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
27   1  p0          -1.86781e-03   3.91931e+02   1.90936e-01  -8.83777e-20
28   2  p1          -3.57736e-01   5.76568e+04   3.00000e-01   2.21023e-19
29   3  p2           1.05013e+04   1.07875e+04   2.49190e+00   7.59498e-21
30 FCN=7.46252e-08 FROM MIGRAD    STATUS=CONVERGED      41 CALLS          42 TOTAL
31                     EDM=4.95553e-29    STRATEGY= 1      ERROR MATRIX ACCURATE
32  EXT PARAMETER                                   STEP         FIRST
33  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
34   1  p0           1.04992e+04   3.91931e+02   1.90936e-01   5.67454e-18
35   2  p1          -1.05043e+04   5.76568e+04   2.81426e+00  -2.97099e-20
36   3  p2           7.18341e-02   1.07875e+04   3.00000e-01   8.85414e-19
37 FCN=1.31761e-07 FROM MIGRAD    STATUS=CONVERGED      39 CALLS          40 TOTAL
38                     EDM=3.693e-30    STRATEGY= 1      ERROR MATRIX ACCURATE
39  EXT PARAMETER                                   STEP         FIRST
40  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
41   1  p0          -1.59505e-03   3.91931e+02   1.90936e-01  -3.95100e-20
42   2  p1          -2.96044e-01   5.76568e+04   3.00000e-01   4.69839e-20
43   3  p2           1.05012e+04   1.07875e+04   2.49190e+00   2.01825e-20
44 Calc_ratio0.435000_dVS0.008900_hVS0.009000_NVS29.0_NHW24.0_rHW0.000203_EDipole10500.0_kx10500.0.txt Ex= 0.107066 Ey= 0.142268
```

Figure 12: Example for one output.log file. You can see the output from the fit and the final result with the corresponding filename

ation. Now that we know the program is written wright, we can uncomment everything that prints out the plots. So when we excecute the program now. It gives us only output in the terminal.
By automatisating with an sh script, we can run this program for each file in a series of measurements and store the output in a file called output.log. How excacly this happens will be explained later. For now it is important that we have stored the output from the program for all files in one file.

4.4 Plotting the results of a series of mesurements

The next step is to plot the result of the series of mesurements. For this we write a python-script. The reason for using python is the simplicity of reading a file and the matplotlib library. We can read from a complex ascii file and create a graph quickly. We begin by importing all packages we need. These are import sys for reading the program arguments, numpy for using the numpy array and other functions numpy supports, re for finding the values of the parameters, different things from matplotlib like pyplot and colors. From collections we import OrderedDict for specific anlaysis of an array like sorting or removing double values.
We begin writing a function dependant of the filename which reads the file, extracs all parameters, looks which parameters stay constant and which change. They store all values in arrays. We begin by reading the file line by line and defining empty lists for each parameter. The system dependant parameters from Agros2D are ratioVs, dVS, hVS, NVS, rHW EDipole and k. We also create two empty lists zx and zy which will later be filled with the deviation values. We also have two empty lists xy and xypar which are later filled with the variables.
With re we create a pattern which recognizes if the actual line is a line from the fitting or a line which contains the parameters and the deviation. The lines for the fitting are ignored, but the other lines were read. Re finds the values of each parameter and stores all values to the specific lists. When the file is finished reading we need to find the lists which contains constant values and the lists where the values changes. For zx and zy we do nothing because we already know it is the standarddeviation which changes. The lists where it is important are the ones defined from Agros2D (ratioVS, dVs, hVS, NVS, NHW, rHW). For finding the other values which change we use the count function for lists. It counts the number of elements which are equal to the one defined in the argument of the system. So when all elements of a list are equal to the first element it is constant. If not they are changing parameters.
The lists of the changing parameters are added to the xy lists. Parameters names are added to xypar list as a string.
Then we need to look at the xy and the xypar list. If the number of lists in xy are equal too, we have nothing else to do with this list, because our goal is to plot the zx and zy lists in a 2D hist dependant from the two lists in xy. If the number of changing parameters is greater the two we have to examine the lists. How it is said in a prior section, I made some measurements where more than two parameters are changing. But the third parameter is correlated to one of the other two. So we need to find this one by looking at a dependance of of two or more lists. When we found this one, we can delete them from the xy list and the xypar list. Also in the xypar list I've added the string of the dependant parameter the dependancy of the third parameter, so in the final plot we know of this dependancy. The third case is that xy contains only one list. This can happen if Agros2D could only calculate the results of one parameter as a variable and couldn't change the other without giving an error. Then we add a list which only contains zeros to xy and the string '0' to xypar. For the case that xy doesn't contain a list it raises an error. If this happens it is possible that python wasn't able to find the structure in the ascii file.
When this is done a few other thing are done in this function like defining a plotname and calculating the number of different x and y values. At the end of this function x, y, zx, zy, the number of x-values, the number of y-values, xypar and the plotname are returned.
The next step is to define the plot function dependant from x, y, z, the numbers of x and y

elements and the name of the axis, which creates a 2dHistogram like a heatmap. Like in the C++/ROOT program we begin by aligning the bins. Then the only thing we need to do is using the hist2d function from matplotlib.pyplot, define names for the axis, a colorbar and saving the result as an image or pdf file.
Now that we have defined all functions, we can excecute the functions. Like said before we take the filename from the arguments of the program and excecute the read-function dependant from this argument. Now it returns us every necessary value and also the names of the axis and the plotnames. The function was excecuted with the value for z beeing the sum of zx and zy. Then we can find the saved plots and see where the minimum is located.

4.5 Automatisation of the processes and containing clarity

The reason for autimatisation of all the processes is the amount of data and the amount of results from all calculations. So in this chapter i will explain the things i have used and all things that i had to take care of for automatisation.
By automatisation, many things are to take care of to contain clarity. I will list them in the following:

- contain clarity in the mass of files
- therefore create folders
- filenames and foldernames should contain all essential informations
- changing parameters and parameters range should be as easy as possible
- the programs for examining the data should stay how they are and should not need to be changed
- the programs should take attention to all exceptions which can happen
- there should be as few as possible moments necessary for the user to step in.

So let's begin by looking at some part of the Agros2D script in figure 14 We have defined the function calc which takes all parameters as arguments. They normally calculate the electric field and save them into an ascii file. The name of the file is important for clarity. It was chosen to bring all values of the parameters and the parameters name into this file. Before we excecute the function we create a folder with the name of the variables, the date and time when the function first excecutes. In this folder all the calculations for one series of measurements are saved. For caling the function, we use two for loops, because we always takes two ranges of parameters and the other ones as constant. So when the loop is finished, we can see many ascii files. From the names of the files you can see that two parameters changes the other ones stay constant. There also is the possibility that Agros2D can't calculate the system with some parameters for example if the wires overlap. The solution therefore is to use the try function from python. When one function can't be calculated yet it continues. If there was an exception an other code runs instead. This part stores in a file called error.log for which parameters an error occurs. By making it easier to change the parameters i've created some dictionarys you can see in the annotations.
The next step is to use the C++ program which calculates the standard deviation of all of these files. The good thing at this files are that they always have the same format. So we can read them always the same way. For not always changing and recompiling the code i've chosen to take the filename from the first program argument. So we only need to call the program in the terminal and write the filename. Than it prints the filename and the standard deviation. Now that we can't do this with every file we need to write a small sh script. It will look like this: The script searches for every ascii file from the Agros2D script and excecutes the C++ programm

```
###Function arguments

### Number of vertical strips (VS) and horizontal wires (HW)
#NVS
#NHW

########### vertical strips (VS) ######################
### ratio of width of strips wVS to the pitch [m]
#ratioVS

############ horizontal wires (HW) ####################
### radius [m]
#rHW

### thickness of vertical strips [m]
#dVS

### depth of the cutout of the strips
### should be greater than dVS
#hVS

############ field strength ####################
### nominal field of dipole [V/m]
#EDipole

### Quadrupole strength [V/m^2]
#kx
#ky

def calc(ratioVS=0.4, dVS=0.001, hVS = 0.01, NVS=19, NHW=9, rHW = 0.0005, EDipole=10500, kx=10500, ky=-10500, dep=dep):
    # problem
    problem = a2d.problem(clear = True)
    problem.coordinate_type = "planar"
    problem.mesh_type = "triangle"

```

Figure 13: defining the calc function with the changable parameters as arguments

for each file. To save some time this script creates many subprocesses in parallel. The output from the program will be saved in an output.log file. For excecuting this script you need to be in the folder which contains the ascii files with the terminal. The writeoutput script has to be placed in the parant directory. With the command ../writeoutput the script runs. Then this output file can be read with a python script which displays the results in a 2d histogram.
The next step of automatisation became necessary because of the reason of having more than only one folder of series of calculations. For this I have written an other sh script, which looks for every output.log file in every folder and excecutes the python script with the output.log file as an argument. The script was written like the writeoutput script and it works the same way.

```
#!/bin/sh
for argument in */output.log; do #search in every folder a file called output.log
  ./findmin3.py "$argument"& #excecutes the pythonscript with the output.log file as argument
done
wait #wait till all scripts are finished
```

Figure 16: script which excecutes the findmin3.py program for each output.log file

The last script i've written combines the two sh scripts. By excecuting this, it looks for every folder from Agros2D, goes into this folder end excecutes the writeoutput script. When this is finished it goes to the next folder and so on. When all output.log files are written it excecutes the other bash script which writes all plots.

```
#!/bin/sh
for directory in *_*/; do #the directorys from agros2D contains the _ caracter so this script will find this folders
  cd "$directory"
  ../writeoutput #execites the script writeoutput for the actual directory
  cd ..
  echo $directory "output written" #only for confirmation that this script was in this folder
done
./writeplot #finally excecute the python script
```

Figure 17: script for linking the two sh scripts

```
295     np.savetxt(os.path.join(dep, 'Calc_ratio{:f}_dVS{:f}_hVS{:f}_NVS{:.1f}_NHW{:.1f}_rHW{:f}_EDipole{:.1f}
    _kx{:.1f}.txt'.format(ratioVS, dVS, hVS, NVS, NHW, rHW, EDipole, kx)), np.array([x,y,efieldx,efieldy]).T,
    header="x y Ex Ey", comments='')
296
297
298 #Variables ranges
299 ratioVS=np.arange(0.436, 0.440, 0.00025)
300 dVS=np.arange(0.016, 0.018, 0.0001)
301 #hVS=np.arange(0.0005, 0.01, 0.0005)
302 NVS=np.arange(29, 30, 1)
303 NHW=np.arange(24, 27, 1)
304 rHW=np.arange(0.00020, 0.00022, 0.000001)
305
306 def doev(Variables):
307     arguments={'ratioVS':0.435,'dVS':0.0089,'hVS':0.0090,'NVS':29,'NHW':25,'rHW':0.000215} #all constants
    tring to take them as near as possible to the minimum
308     keys=list(Variables.keys())
309     #for naming the folder:
310     date=time.strftime("%d %b %H.%M", time.gmtime())
311     dep='_'+keys[0]+'_'+keys[1]
312     dep=date+dep
313     os.makedirs(dep)
314     errorlog=open(os.path.join(dep,"error.log"), "w")
315
316     for i in Variables[keys[0]]:
317         for j in Variables[keys[1]]:
318             try:
319                 arguments[keys[0]]=i
320                 arguments[keys[1]]=j
321                 calc(ratioVS=arguments['ratioVS'], dVS=arguments['dVS'], hVS=arguments['dVS']+0.0001,
    NVS=arguments['NVS'], NHW=arguments['NHW'] ,rHW=arguments['rHW'], dep=dep)
322             except Exception:
323                errorlog.write("exception for i="+str(i)+" j="+str(j)+"\n")
324                continue
325     errorlog.close()
326
327 #calculate the electric field for all combinatins of ranges and constants
328 doev({'ratioVS':ratioVS, 'dVS':dVS})
329 doev({'ratioVS':ratioVS, 'NVS':NVS})
330 doev({'ratioVS':ratioVS, 'NHW':NHW})
331 doev({'ratioVS':ratioVS, 'rHW':rHW})
332 doev({'dVS':dVS, 'NVS':NVS})
333 doev({'dVS':dVS, 'NHW':NHW})
334 doev({'dVS':dVS, 'rHW':rHW})
335 doev({'NVS':NVS, 'NHW':NHW})
336 doev({'NVS':NVS, 'rHW':rHW})
337 doev({'NHW':NHW, 'rHW':rHW})
```

Figure 14: Part of the Agros2D script where the calc function is excecuted with all possible values and how the calc function saves the data to an ascii file

5 Evaluation of the results

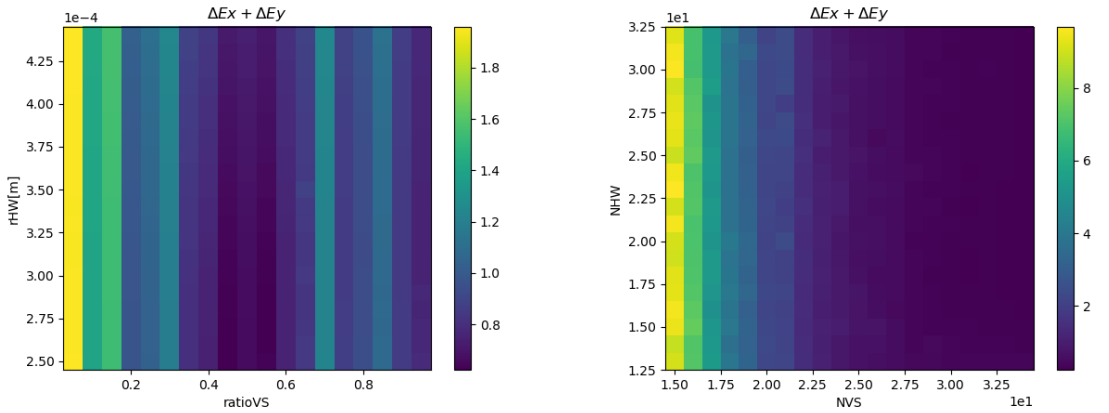
Basically the evaluation of the results is done by looking at the plots and finding a structure in this heatmap. A very important thing is by looking at the colormap. By comparing the colormaps you can see which parameter has a big influence and which not. By seeing stripes you also know if one of the two parameters which are compared in the plot dominates. Later you have to adapt the parameters to find a finer result. For evaluating the results, we need to have plots dependant from all combinations of the six parameters.

5.1 Relevance of the Parameters

In the plots you can see very well which parameter has a big influence and which not (colormap stripes). The most influence of the deviation have the parameters dVS and hVS like you can see in this plot. This is also the plot where you can regognize a good structure and see that hVS must be a bit greater than dVS. If the difference between dVS and hVS is big, the deviations also become very big. The white area is the area where the calculation of the field wasn't possible, because $hVS < dVS$ or the deviation is too high ($\Delta E > (max(z) - min(z)) * 0.1 + min(z)$). The other parameter which also have a big influence are ratioVS and NVS. The other two parameters NHW and rHW have the least influence at the deviation.

```sh
#!/bin/sh
thread=0 #counter for the number of actual processes running
rm -f output.log # if existing delete the actual output.log file
for filename in Calc*.txt; do #search every file from Agros2D
  ../Readdata4 $filename >> output.log & #excecutes the program Readdata. The results are printed to the output.log
     file
  if [ "$thread" -gt "256" ] #when 256 processes are running the script waits till all are finished then it continues
  then
    wait
    thread=0
  fi
  thread=$(expr $thread + 1) #counter
done
wait
```

Figure 15: sh script for handling the Readdata program for each file



(a) Standarddeviation of the residuals dependant from ratioVS and rHW

(b) Standarddeviation of the residuals dependant from NVS and NHW

Figure 19: Standarddeviation from Ex and Ey dependant from some parameters

5.2 Method of chosing the parameters

So at the beginning we try out every combination of the parameters with the whole range possible. I've chosen a fineness of 20 values per range. So calculatiing all combinations doesn't take too much time. Like said in the previous chapter, we begin by adapting the range of the parameters which have the greatest influence. By looking at all plots the dependance of dVS and hVS is very clear, so that we change the Agros2D script so that hVS is always a bit bigger than dVS like $hVS = dVS + 0.0001m$. Now we have to find the other 5 values. Therefore it is a good idea to do the same things we know now with this one condition. Also now that we have an idea which values are the best, we can also change the some constants. By looking at the plots now it is a good idea to compare the results of the parameters which have a big influence at the deviation. There you can adapt the range of the parameters. Sometimes you can also find two interessting ranges of values for one parameter for example dVs has an interessting range about 0.009 and 0.018. The finess should be left by around 20 values for one range for not taking too much time. After the calculations the constants should also be adapted to the value where you previously found a good value. You need to take attention of the colormap to confirm that the deviation becomes smaller and so you have to try out a bit. When you found the best values for the parameters with big influence you can now try to adapt the other parameters a bit more and look at the developpment. When you continue doing this you begin to see more and more irregularities, that means that you are near the point where the influence of the parameter doesn't dominate the deviation, but the uncertanty from Agros2D. That is the point where Agros2D is limiting us and we aren't able to make more exact measurements.

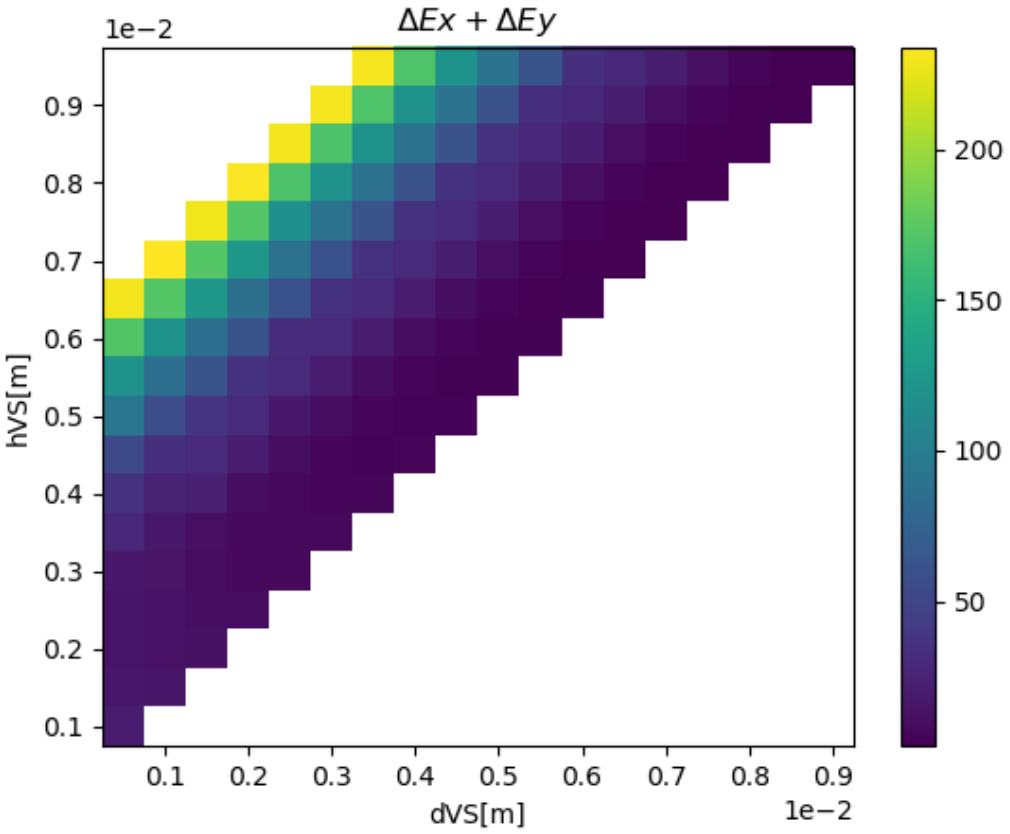
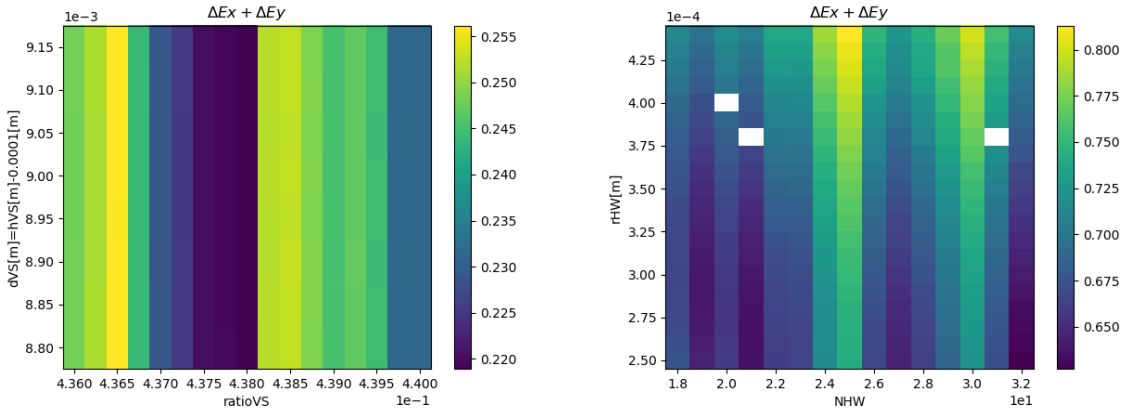


Figure 18: Standarddeviation of the residuals dependant from dVS and hVS

5.3 Finding the minimum

In figure 19 you can see that ratioVS and NVS has a lot more influence to the deviation than rHW and NHW. So we need to optimize them first. For ratioVS it's interessting to see examine the range between 0.35 and 0.6 to examine further. For NVS we can see that more stripes makes the deviation smaller. There we can see what happens by the higher range of NVS.

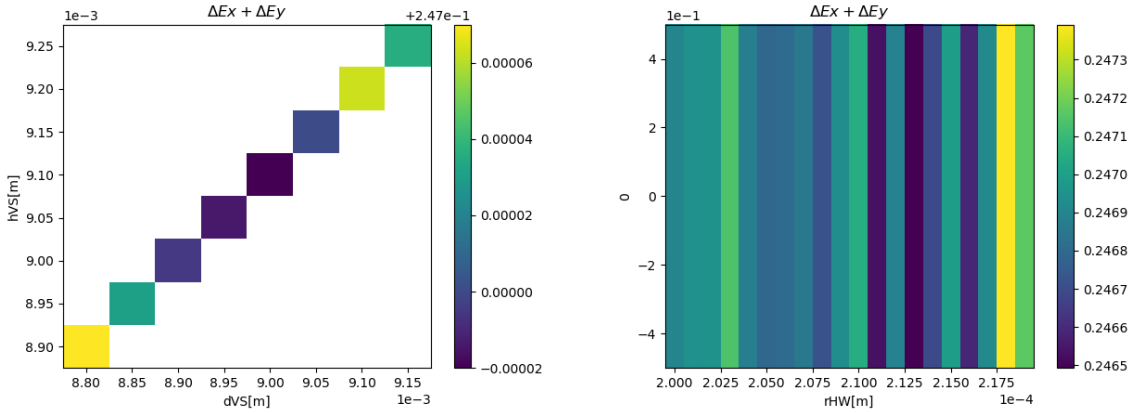


(a) Standarddeviation of the residuals dependant from ratioVS and dVS

(b) Standarddeviation of the residuals dependant from NHW and rHW

Figure 20: Standarddeviation from Ex and Ey dependant from some parameters

You can see in figure 20 plots of a finer range. Especialy in figure 20b) you can see the irregularities of NHW. A problem that needs to be examined further. Therefore we can look at the output.log file if there was an error by fitting the parameters which didn't happened. The most probable source for this irregularity is the uncertainty from Agros2D. Other parameters also take on this form of irregularity when the range becomes finer. So we can't do much then giving the optimum a wide error.



(a) Standarddeviation of the residuals dependant from dVS

(b) Standarddeviation of the residuals dependant from rHW

Figure 21: Standarddeviation from Ex and Ey dependant from some parameters. Here you can see only the dependancy of one parameter for one heatmap

In figure 21 you can see how you can chose the parameters when you are near the minimum. We only use one dimensional plot to see the minimum easier. This can be done with every single parameter. After looking at all these plots following results were found:

- $ratioVS = 0.438 \pm 0.005$
- $dVS = 0.009m \pm 0.0001m$
- $hVS = 0.0091m \pm 0.0001m$
- $NVS = 29 \pm 1$
- $NHW = 27 \pm 1$
- $rHW = 0.00021m \pm 0.00001m$

Additional information about the results:

- The difference of hVS and dVS should be as small as possible but greater than zero. This has the greatest influence.
- dVS and hVS also have a second minimum at 0.016 and 0.0161. This has nearly the same standard deviation, perhaps a bit bigger. Both minimas are sourounded by some high values, but it can also be an uncertanty of Agros2D
- ratioVS also has a wide range of local minimas. The widest minimum was chosen
- For NVS you can say that the greater the value is the smaller the deviation, but at some point the deviation only gets better a small amount
- For NHW the number doesn't have a very big influence so if there are other limits, it isn't nesessary to force the system to this value

- rHW was one of the easier determinable values because by rHW you can always see a good structure and for all series the minimus stayed at the same place
- The values are only determined with Agros2D by a numerical method without a very high finesse. So perhaps other algorithms will find completely different values. For finding good values there a more powerful program than Agros2D should be used.
- The number of refinements was set to zero for not having a memory error by running the script. By comparing systems, like the planar or cylindrical capacitor with theoretical values, we know they aren't always very exact.
- Some parameters like ratioVs and dVS show also some dependacy
- When we incrise the number of horizontal wires (NHW), we need to decrease the radius of horizontal wires to stay at a minimum.

5.4 Problems and uncertanty

The method I used for finding the minimum is a bit like a brute-force method. I calculated the electric field and saw how the standard deviation looks like. This was tried for a lot of parameters. The problem is only that it takes a lot of time to calculate the standard deviation of each possibility. So we have to limit the parameters we choose. And our strategy was to find the minimum of the plots we see and analyse the places excacly. But that means it is possible that we only found a local minimum and not a global. There are also some other things which have some influence of the uncertainty. One is the numeric kind of calculation in Agros2D which also gives us an uncertainty. The second thing is that we only calculate the deviation in specific places.
So the question is now which uncertainties have to be taken into account and which are so small that they do not have to be considered. How I already said it is possible that we only found a local minimum and not the global. To know how the probability is we need to make more measurements and look at other minimums where we need much time. To look at other minimums makes the propability near zero not to have a local but a global minimum but it is never zero. So it is possible to optimize it with more aviable time.
The second source of uncertanty is the numerical calculation from Agros2D. There it is easier to estimate the error. One reason for this is that we know the theoretical values of some electric systems which we can also defined in Agros2D and compare then. One other help for estimationg the error is by looking at some calculations from our system in Agros2D. By looking at the noise of the standard deviation from the last calculations we also can estimate an error.
The third error which comes from our fineness of the parameters is not to take into account. The fineness is good enough to see the noise of the Agros2D calculation what means it is much more smaller than the other errors.

6 Conclusion

This thesis gives us one method how to find the minimum of a physical system. We also know of some other possible minimums which we can examine if we want to do it. The result also gives us a bit more information at how we can improve the system. For furher optimization it is perhaps possible to use one dimensional plots, there you can find the local minimums better for each parameter. It is also easier to find minimums of not so influenced parameters. But I've chosen to use 2 dimensinal histograms for also seeing the dependance of the parameters to each other. You can see more information on two dimensional histograms. Perhaps there should also be more use of the changing of the dipole field and quadropole field. They were left constant.
The programs written for analysing the electric field runs very reliable and they also consider many variations of the variables. But they always can be optimized and shortened more.

The results have to be handled with care. It could be possible that the minimum found isn't a global minimum. To be sure that it is a global minimum, we need to try every single combination of the six parameters. If we try 20 values for each parameter, we have to calculate 20^6 electric fields when we want every combination. But we tried at least every combination of the most influencable parameters, so if we have only found a local minimum, the global minimum should be near our determined values.
We need to compare this result with other results for this system. The second problem is the uncertanty of Agros2D. For not having a memory error we need to take the number of refinements to zero and the polimomical order to two. This means Agros2D can't calculate very excactly. The third disadvantage of Agros2D is the use of only two dimensions. To improve the result it is a good idea to write an own algorithm for the calculation of complex systems which we can optimize for our own needs. It should need a bit of time writing a Program which is able to numerically calculate the electric field more excacly but it is possible. Perhaps the are also other programs like Agros2D which can do it more exacly. In conclusion the optimization of the electric field of the storage ring is not finished yet and needs more detailed calculations for the complete understanding of the problem.

7 Annotations

```
import agros2d as a2d
import numpy as np
import matplotlib.pyplot as plt

U1, U2=1000, -1000 #voltage of the plates U1 is the left plate U2 the right plate
d_2=0.2 #half of the distance between the plates
l=0.8 #length of the plates
h=0.05 #thickness of the plates
wl=5 #half length of on side of the world volume
def calc(l=l, d_2=d_2, h=h, wl=wl):
    # problem
    problem = a2d.problem(clear = True)
    problem.coordinate_type = "planar"
    problem.mesh_type = "triangle"

    # fields
    # electrostatic
    electrostatic = a2d.field("electrostatic")
    electrostatic.analysis_type = "steadystate"
    electrostatic.matrix_solver = "mumps"
    electrostatic.number_of_refinements = 0
    electrostatic.polynomial_order = 2
    electrostatic.adaptivity_type = "disabled"
    electrostatic.solver = "linear"


    # boundaries
    electrostatic.add_boundary("U1", "electrostatic_potential", {"electrostatic_potential" : U1})
    electrostatic.add_boundary("U2", "electrostatic_potential", {"electrostatic_potential" : U2})
    electrostatic.add_boundary("border", "electrostatic_surface_charge_density", {"electrostatic_surface_charge_density" : 0})


    # materials
    electrostatic.add_material("worldvolume", {"electrostatic_permittivity" : 1, "electrostatic_charge_density" : 0}) #for the
     outside of the condensator


    ############
    # geometry #
    ############

    #left plate
    geometry = a2d.geometry
    geometry.add_edge(-d_2-h, l, -d_2, l, boundaries = {"electrostatic" : "U1"})
    geometry.add_edge(-d_2, -l, -d_2, l, boundaries = {"electrostatic" : "U1"})
    geometry.add_edge(-d_2-h, l, -d_2-h, -l, boundaries = {"electrostatic" : "U1"})
    geometry.add_edge(-d_2-h, -l, -d_2, -l, boundaries = {"electrostatic" : "U1"})

    #right plate
    geometry.add_edge(d_2, -l, d_2, l, boundaries = {"electrostatic" : "U2"})
    geometry.add_edge(d_2, l, d_2+h, l, boundaries = {"electrostatic" : "U2"})
    geometry.add_edge(d_2+h, l, d_2+h, -l, boundaries = {"electrostatic" : "U2"})
    geometry.add_edge(d_2+h, -l, 0.2, -l, boundaries = {"electrostatic" : "U2"})

    #world volume
    geometry.add_edge(-wl, wl, wl, wl, boundaries = {"electrostatic" : "border"})
    geometry.add_edge(wl, wl, wl, -wl, boundaries = {"electrostatic" : "border"})
    geometry.add_edge(wl, -wl, -wl, -wl, boundaries = {"electrostatic" : "border"})
    geometry.add_edge(-wl, -wl, -wl, wl, boundaries = {"electrostatic" : "border"})

    #labels of the materials this only calculates the field outside the plates and inside the worldvolume
    geometry.add_label(0, 0, materials = {"electrostatic" : "worldvolume"})
    geometry.add_label(-d_2-h/2, 0, materials = {"electrostatic" : "none"})
    geometry.add_label(d_2+h/2, 0, materials = {"electrostatic" : "none"})
    a2d.view.zoom_best_fit()

    #create mesh and calculate the field
    problem.solve()

    return electrostatic.local_values(0, 0)["E"]
l=np.arange(0.1, 2.0, 0.05)
E=[]
for i in l:
    E+=[calc(l=i)]

plt.figure()
plt.plot(l, E, label='Agros2D')
plt.plot(l, np.ones(len(l))*(U1-U2)/(2*d_2), label='theoretical value')
plt.xlabel('length [m]')
plt.ylabel('E [Vm]')
plt.legend()
plt.title('comparision Agros2D and theoretical result planar')
plt.savefig('comparision_planar.pdf')

plt.figure()
```

```
plt.plot(l, np.array(E)-(U1-U2)/(2*d_2), label='residuals')
plt.xlabel('length [m]')
plt.ylabel(r'$\Delta$E [Vm]')
plt.ylim(-1e-5,1e-5)
plt.ticklabel_format(axis='y',style='sci',scilimits=(1,4))
plt.legend()
plt.title('comparision Agros2D and theoretical result')
plt.savefig('noise_planar.pdf')
```

Agros2D and Python script for the planar capacitor (named plattenkondensator.py)

```python
import agros2d as a2d
import numpy as np
import matplotlib.pyplot as plt

r1=0.2
r2=0.4
h=0.05
l=0.8
wl=4
U1=1000
U2=-1000

def calc(r1=r1, r2=r2, h=h, l=l, wl=wl):
    # problem
    problem = a2d.problem(clear = True)
    problem.coordinate_type = "axisymmetric"
    problem.mesh_type = "triangle"

    # fields
    # electrostatic
    electrostatic = a2d.field("electrostatic")
    electrostatic.analysis_type = "steadystate"
    electrostatic.matrix_solver = "mumps"
    electrostatic.number_of_refinements = 2
    electrostatic.polynomial_order = 3
    electrostatic.adaptivity_type = "disabled"
    electrostatic.solver = "linear"


    # boundaries
    electrostatic.add_boundary("U1", "electrostatic_potential", {"electrostatic_potential" : U1})
    electrostatic.add_boundary("U2", "electrostatic_potential", {"electrostatic_potential" : U2})
    electrostatic.add_boundary("border", "electrostatic_surface_charge_density", {"electrostatic_surface_charge_density" : 0})


    # materials
    electrostatic.add_material("worldvolume", {"electrostatic_permittivity" : 1, "electrostatic_charge_density" : 0})

    # geometry
    geometry = a2d.geometry
    #world volume:
    geometry.add_edge(0, wl, wl, wl, boundaries = {"electrostatic" : "border"})
    geometry.add_edge(wl, wl, wl, -wl, boundaries = {"electrostatic" : "border"})
    geometry.add_edge(wl, -wl, 0, -wl, boundaries = {"electrostatic" : "border"})
    geometry.add_edge(0, -wl, 0, wl, boundaries = {"electrostatic" : "border"})

    #left plate r1
    geometry.add_edge(r1, -l, r1, l, boundaries = {"electrostatic" : "U1"})
    geometry.add_edge(r1-h, -l, r1-h, l, boundaries = {"electrostatic" : "U1"})
    geometry.add_edge(r1, l, r1-h, l, boundaries = {"electrostatic" : "U1"})
    geometry.add_edge(r1, -l, r1-h, -l, boundaries = {"electrostatic" : "U1"})

    #right plate r2
    geometry.add_edge(r2, -l, r2, l, boundaries = {"electrostatic" : "U2"})
    geometry.add_edge(r2+h, -l, r2+h, l, boundaries = {"electrostatic" : "U2"})
    geometry.add_edge(r2, l, r2+h, l, boundaries = {"electrostatic" : "U2"})
    geometry.add_edge(r2, -l, r2+h, -l, boundaries = {"electrostatic" : "U2"})

    geometry.add_label((r1+r2)/2, 0, materials = {"electrostatic" : "worldvolume"})
    geometry.add_label(r1-h/2, 0, materials = {"electrostatic" : "none"})
    geometry.add_label(r2+h/2, 0, materials = {"electrostatic" : "none"})
    a2d.view.zoom_best_fit()

    problem.solve()

    return electrostatic.local_values((r1+r2)/2, 0)["E"]
l=np.arange(0.1, 2, 0.05)
Ea=[]
for i in l:
    Ea+=[calc(l=i)]
def E(r):
    return (U1-U2)/(r*np.log(r2/r1))

plt.figure()
plt.plot(l, Ea, label='Agros2D')
plt.plot(l, np.ones(len(l))*E((r2+r1)/2), label='theoretical value')
plt.xlabel('length [m]')
plt.ylabel('E [Vm]')
plt.legend()
plt.title('comparision Agros2D and theoretical result cylinder')
plt.savefig('comparision_cylinder.pdf')

plt.figure()
plt.plot(l, np.array(Ea)-E((r2+r1)/2.), label='residuals')
plt.ticklabel_format(axis='y',style='sci',scilimits=(1,4))
plt.xlabel('length [m]')
plt.ylabel(r'$\Delta$E [Vm]')
plt.ylim(-100.,100.)
```

```
89 plt.legend()
90 plt.title('noise_cylinder')
91 plt.savefig('noise_cylinder.pdf')
```

Agros2D and Python script for the cylindrical capacitor (named Cylindercondensator.py)

```
#############################################################################################################
#                                                S E T U P                                                  #
#############################################################################################################
from __future__ import division
import numpy as np
import agros2d as a2d
import time
import os
dep=''

###Function arguments

### Number of vertical strips (VS) and horizontal wires (HW)
#NVS
#NHW

############ vertical strips (VS) ########################
### ratio of width of strips wVS to the pitch [m]
#ratioVS

############ horizontal wires (HW) ########################
### radius [m]
#rHW

### thickness of vertical strips [m]
#dVS

### depth of the cutout of the strips
### should be greater than dVS
#hVS

############ field strength ########################
### nominal field of dipole [V/m]
#EDipole

### Quadrupole strength [V/m^2]
#kx
#ky

def calc(ratioVS=0.4, dVS=0.001, hVS = 0.01, NVS=19, NHW=9, rHW = 0.0005, EDipole=10500, kx=10500, ky=-10500, dep=dep):
    # problem
    problem = a2d.problem(clear = True)
    problem.coordinate_type = "planar"
    problem.mesh_type = "triangle"

    # fields: electrostatic
    multipole = a2d.field("electrostatic")
    multipole.analysis_type = "steadystate"
    multipole.matrix_solver = "mumps"
    multipole.number_of_refinements = 0
    multipole.polynomial_order = 2
    multipole.adaptivity_type = "disabled"
    multipole.solver = "linear"


    #############################################################################################################
    #                                          User: Input Quantities                                           #
    #############################################################################################################

    ### Size of the beam volume [m]
    xCap = 0.038
    yCap = 0.200

    ### usable area [m]
    ### this is not a physical area. Field quality will only be recorded in this area

    xUse=0.6*xCap #record 60% of the physical length and width
    yUse=0.6*yCap

    ### the blocks [m]
    hVBlock = 0.198
    dVBlock = 0.06
    wHBlock = 0.08
    dHBlock = 0.04

    ### empty gap at the end of the block [m]
    offVS = 0.002

    ### pitch
    pitcwVS = ( hVBlock - 2.*offVS ) / (NVS - 1. + ratioVS)

    ### width of the strips (wVS)
    wVS = ratioVS * pitcwVS

    ### empty gap at the end of the block [m]
    offHW = 0.021
    ### depth of the wires in the block (from the surface of the block to the center of the wire [m]
    hHW = rHW
```

```
    ### pitch between centers of the wires
    pitchHW = ( wHBlock - 2.*offHW ) / ( NHW - 1. )

    ### distance of world volume to the outer face of the blocks [m]
    dWorld = 0.01
    ### horizontal width and vertical height of the world volume
    xlWorld = xCap + 2. * ( dVS + dVBlock + dWorld )
    ylWorld = yCap + 2. * (       dHBlock + dWorld )
    ### Coordinates of the world label
    xWLabel = xlWorld/2. - dWorld/2.
    yWLabel = ylWorld/2. - dWorld/2.

    ################################################################################
    #                  Now we calculate the position of the strips                 #
    ################################################################################
    ### coordinates of the strip specify the center of the front face of the strip
    def xVS(N):
        xpos = xCap/2.
        return xpos

    def yVS(N):
        ypos = -hVBlock/2. + offVS + wVS/2. + N * pitcwVS
        return ypos

    ### coordinates of the wires specify the center of the wire.
    def xHW(N):
        xpos = -wHBlock/2. + offHW + N * pitchHW
        return xpos

    def yHW(N):
        ypos = yCap/2.
        return ypos

    ################################################################################
    #                     Define the potentials on the electrods                   #
    ################################################################################
    ### These function calcualate the nominal potential
    ### Dipole
    def UD(x,y):
        U = -EDipole * x
        return U

    ### Quadrupole
    def UQ(x,y):
        U = 0.5 * kx * (x**2 - 0.25*xCap**2) + 0.5 * ky * (y**2 - 0.25*yCap**2)
        return U

    ################################ vertical electrods ####################################
    for i in range(0 , NVS):
        U = UD(-xVS(i),yVS(i)) + UQ(-xVS(i),yVS(i))
        multipole.add_boundary("UVL%d"%(i), "electrostatic_potential", { "electrostatic_potential" : U })
        U = UD( xVS(i),yVS(i)) + UQ( xVS(i),yVS(i))
        multipole.add_boundary("UVR%d"%(i), "electrostatic_potential", { "electrostatic_potential" : U })

    ################################ horizontal electrods ##################################
    for j in range(0 , NHW):
        U = UD(xHW(j), yHW(j)) + UQ( xHW(j), yHW(j))
        multipole.add_boundary("UHU%d"%(j), "electrostatic_potential", { "electrostatic_potential" : U })
        U = UD(xHW(j),-yHW(j)) + UQ( xHW(j),-yHW(j))
        multipole.add_boundary("UHL%d"%(j), "electrostatic_potential", { "electrostatic_potential" : U })

    ################################### other potentials ###################################
    multipole.add_boundary("ground", "electrostatic_potential", {"electrostatic_potential" : 0.})
    multipole.add_boundary("block",  "electrostatic_surface_charge_density", {"electrostatic_surface_charge_density" : 0.})

    ################################################################################
    #                                Materials                                     #
    ################################################################################
    multipole.add_material("vacuum", {"electrostatic_permittivity" : 1, "electrostatic_charge_density" : 0})
    multipole.add_material("ebaboard", {"electrostatic_permittivity" : 10, "electrostatic_charge_density" : 0})

    ################################################################################
    #                     Build the geometry of the electrostatic                  #
    ################################################################################
    deflector = a2d.geometry

    ################################ world volume ##########################################
    deflector.add_edge(-xlWorld/2., -ylWorld/2.,  xlWorld/2., -ylWorld/2., boundaries = {"electrostatic" : "ground"})
    deflector.add_edge( xlWorld/2., -ylWorld/2.,  xlWorld/2.,  ylWorld/2., boundaries = {"electrostatic" : "ground"})
    deflector.add_edge( xlWorld/2.,  ylWorld/2., -xlWorld/2.,  ylWorld/2., boundaries = {"electrostatic" : "ground"})
    deflector.add_edge(-xlWorld/2.,  ylWorld/2., -xlWorld/2., -ylWorld/2., boundaries = {"electrostatic" : "ground"})
    deflector.add_label(xWLabel, yWLabel, materials = {"electrostatic" : "vacuum"})

    ############################### vertical strips ########################################
    for i in range(0 , NVS):
         ### left strips first
        deflector.add_edge(-xVS(i)-dVS, yVS(i)-wVS/2., -xVS(i)    , yVS(i)-wVS/2., boundaries = {"electrostatic" : "UVL%d"%(i)})
        deflector.add_edge(-xVS(i)    , yVS(i)-wVS/2., -xVS(i)    , yVS(i)+wVS/2., boundaries = {"electrostatic" : "UVL%d"%(i)})
```

```
        deflector.add_edge(-xVS(i)    , yVS(i)+wVS/2., -xVS(i)-dVS, yVS(i)+wVS/2., boundaries = {"electrostatic" : "UVL%d"%(i)})
        deflector.add_edge(-xVS(i)-dVS, yVS(i)+wVS/2., -xVS(i)-dVS, yVS(i)-wVS/2., boundaries = {"electrostatic" : "UVL%d"%(i)})
        deflector.add_label(-xVS(i)-dVS/2., yVS(i), materials = {"electrostatic" : "none"})

         ### now right strips
        deflector.add_edge( xVS(i)+dVS, yVS(i)-wVS/2.,  xVS(i)    , yVS(i)-wVS/2., boundaries = {"electrostatic" : "UVR%d"%(i)})
        deflector.add_edge( xVS(i)    , yVS(i)-wVS/2.,  xVS(i)    , yVS(i)+wVS/2., boundaries = {"electrostatic" : "UVR%d"%(i)})
        deflector.add_edge( xVS(i)    , yVS(i)+wVS/2.,  xVS(i)+dVS, yVS(i)+wVS/2., boundaries = {"electrostatic" : "UVR%d"%(i)})
        deflector.add_edge( xVS(i)+dVS, yVS(i)+wVS/2.,  xVS(i)+dVS, yVS(i)-wVS/2., boundaries = {"electrostatic" : "UVR%d"%(i)})
        deflector.add_label( xVS(i)+dVS/2., yVS(i), materials = {"electrostatic" : "none"})

    ############################## horizontal wires #######################################
    for i in range(0 , NHW):
         ### upper wires first
        deflector.add_edge(xHW(i)-rHW, yHW(i)    , xHW(i)    , yHW(i)-rHW, angle=90, boundaries = {"electrostatic" : "UHU%d"%(i)})
        deflector.add_edge(xHW(i)    , yHW(i)-rHW, xHW(i)+rHW, yHW(i)    , angle=90, boundaries = {"electrostatic" : "UHU%d"%(i)})
        deflector.add_edge(xHW(i)+rHW, yHW(i)    , xHW(i)    , yHW(i)+rHW, angle=90, boundaries = {"electrostatic" : "UHU%d"%(i)})
        deflector.add_edge(xHW(i)    , yHW(i)+rHW, xHW(i)-rHW, yHW(i)    , angle=90, boundaries = {"electrostatic" : "UHU%d"%(i)})
        deflector.add_label(xHW(i), yHW(i), materials = {"electrostatic" : "none"})

         ### now lower wires
        deflector.add_edge(xHW(i)    , -yHW(i)+rHW, xHW(i)-rHW, -yHW(i)    , angle=90, boundaries = {"electrostatic" : "UHL%d"%(i)})
        deflector.add_edge(xHW(i)+rHW, -yHW(i)    , xHW(i)    , -yHW(i)+rHW, angle=90, boundaries = {"electrostatic" : "UHL%d"%(i)})
        deflector.add_edge(xHW(i)    , -yHW(i)-rHW, xHW(i)+rHW, -yHW(i)    , angle=90, boundaries = {"electrostatic" : "UHL%d"%(i)})
        deflector.add_edge(xHW(i)-rHW, -yHW(i)    , xHW(i)    , -yHW(i)-rHW, angle=90, boundaries = {"electrostatic" : "UHL%d"%(i)})
        deflector.add_label(xHW(i),-yHW(i), materials = {"electrostatic" : "none"})

    ################################## vertical blocks #####################################
    ### left block
    deflector.add_edge(-xCap/2.-dVS+hVS,-hVBlock/2.,-xCap/2.-dVS+hVS,yVS(0)-wVS/2., boundaries = {"electrostatic" : "block"} )
    deflector.add_edge(-xCap/2.-dVS+hVS, hVBlock/2.,-xCap/2.-dVS+hVS,yVS(NVS-1)+wVS/2., boundaries = {"electrostatic" : "block"} )

    for i in range(0, NVS):
        if i < NVS-1:
            deflector.add_edge(-xCap/2.-dVS+hVS, yVS(i)+wVS/2.,-xCap/2.-dVS+hVS, yVS(i+1)-wVS/2., boundaries = {"electrostatic" : "
     block"} )
        if hVS != 0.:
            deflector.add_edge(-xCap/2., yVS(i)+wVS/2,-xCap/2.-dVS+hVS, yVS(i)+wVS/2, boundaries = {"electrostatic" : "block"} )
            deflector.add_edge(-xCap/2., yVS(i)-wVS/2,-xCap/2.-dVS+hVS, yVS(i)-wVS/2, boundaries = {"electrostatic" : "block"} )

    deflector.add_edge(-xCap/2.-dVS-dVBlock,-hVBlock/2.,-xCap/2.-dVS-dVBlock, hVBlock/2., boundaries = {"electrostatic" : "block"} )
    deflector.add_edge(-xCap/2.-dVS-dVBlock,-hVBlock/2.,-xCap/2.-dVS+hVS    ,-hVBlock/2., boundaries = {"electrostatic" : "block"} )
    deflector.add_edge(-xCap/2.-dVS-dVBlock, hVBlock/2.,-xCap/2.-dVS+hVS    , hVBlock/2., boundaries = {"electrostatic" : "block"} )
    deflector.add_label(-xCap/2.-dVS-dVBlock/2., 0. , materials = {"electrostatic" : "ebaboard" } )

    ### right block
    deflector.add_edge( xCap/2.+dVS-hVS,-hVBlock/2., xCap/2.+dVS-hVS,yVS(0)-wVS/2., boundaries = {"electrostatic" : "block"} )
    deflector.add_edge( xCap/2.+dVS-hVS, hVBlock/2., xCap/2.+dVS-hVS,yVS(NVS-1)+wVS/2., boundaries = {"electrostatic" : "block"} )

    for i in range(0, NVS):
        if i < NVS-1:
            deflector.add_edge( xCap/2.+dVS-hVS, yVS(i)+wVS/2., xCap/2.+dVS-hVS, yVS(i+1)-wVS/2., boundaries = {"electrostatic" : "
     block"} )
        if hVS != 0.:
            deflector.add_edge( xCap/2., yVS(i)+wVS/2, xCap/2.+dVS-hVS, yVS(i)+wVS/2, boundaries = {"electrostatic" : "block"} )
            deflector.add_edge( xCap/2., yVS(i)-wVS/2, xCap/2.+dVS-hVS, yVS(i)-wVS/2, boundaries = {"electrostatic" : "block"} )

    deflector.add_edge( xCap/2.+dVS+dVBlock,-hVBlock/2., xCap/2.+dVS+dVBlock, hVBlock/2., boundaries = {"electrostatic" : "block"} )
    deflector.add_edge( xCap/2.+dVS+dVBlock,-hVBlock/2., xCap/2.+dVS-hVS    ,-hVBlock/2., boundaries = {"electrostatic" : "block"} )
    deflector.add_edge( xCap/2.+dVS+dVBlock, hVBlock/2., xCap/2.+dVS-hVS    , hVBlock/2., boundaries = {"electrostatic" : "block"} )
    deflector.add_label( xCap/2.+dVS+dVBlock/2., 0. , materials = {"electrostatic" : "ebaboard" } )


    ################################## horizontal blocks ###################################
    ### upper block
    deflector.add_edge(-wHBlock/2., yCap/2.-hHW, xHW(0)-rHW, yCap/2.-hHW, boundaries = {"electrostatic" : "block"} )
    deflector.add_edge( wHBlock/2., yCap/2.-hHW, xHW(NHW-1)+rHW, yCap/2.-hHW, boundaries = {"electrostatic" : "block"} )

    for j in range(0, NHW):
        if j < NHW-1:
            deflector.add_edge(xHW(j)+rHW, yCap/2.-hHW, xHW(j+1)-rHW, yCap/2.-hHW, boundaries = {"electrostatic" : "block"} )
        if hHW != 0:
            deflector.add_edge(xHW(j)-rHW, yCap/2.-hHW, xHW(j)-rHW, yCap/2., boundaries = {"electrostatic" : "block"} )
            deflector.add_edge(xHW(j)+rHW, yCap/2.-hHW, xHW(j)+rHW, yCap/2., boundaries = {"electrostatic" : "block"} )

    deflector.add_edge(-wHBlock/2., yCap/2.+dHBlock, wHBlock/2., yCap/2.+dHBlock, boundaries = {"electrostatic" : "block"} )
    deflector.add_edge(-wHBlock/2., yCap/2.-hHW, -wHBlock/2., yCap/2.+dHBlock, boundaries = {"electrostatic" : "block"} )
    deflector.add_edge( wHBlock/2., yCap/2.-hHW,  wHBlock/2., yCap/2.+dHBlock, boundaries = {"electrostatic" : "block"} )
    deflector.add_label(0., yCap/2.+dHBlock/2., materials = {"electrostatic" : "ebaboard" } )

    ### lower block
    deflector.add_edge(-wHBlock/2.,-yCap/2.+hHW, xHW(0)-rHW,-yCap/2.+hHW, boundaries = {"electrostatic" : "block"} )
    deflector.add_edge( wHBlock/2.,-yCap/2.+hHW, xHW(NHW-1)+rHW,-yCap/2.+hHW, boundaries = {"electrostatic" : "block"} )

    for j in range(0, NHW):
        if j < NHW-1:
            deflector.add_edge(xHW(j)+rHW,-yCap/2.+hHW, xHW(j+1)-rHW,-yCap/2.+hHW, boundaries = {"electrostatic" : "block"} )
        if hHW != 0:
```

```
            deflector.add_edge(xHW(j)-rHW,-yCap/2.+hHW, xHW(j)-rHW,-yCap/2., boundaries = {"electrostatic" : "block"} )
            deflector.add_edge(xHW(j)+rHW,-yCap/2.+hHW, xHW(j)+rHW,-yCap/2., boundaries = {"electrostatic" : "block"} )

    deflector.add_edge(-wHBlock/2.,-yCap/2.-dHBlock, wHBlock/2.,-yCap/2.-dHBlock, boundaries = {"electrostatic" : "block"} )
    deflector.add_edge(-wHBlock/2.,-yCap/2.+hHW, -wHBlock/2.,-yCap/2.-dHBlock, boundaries = {"electrostatic" : "block"} )
    deflector.add_edge( wHBlock/2.,-yCap/2.+hHW,  wHBlock/2.,-yCap/2.-dHBlock, boundaries = {"electrostatic" : "block"} )
    deflector.add_label(0.,-yCap/2.-dHBlock/2., materials = {"electrostatic" : "ebaboard" } )


    # Activate preprocessor
    deflector.activate()
    # Solve problem
    problem.solve()

    # Extract data from usable volume
    x, y, efieldx,efieldy=[], [], [], []
    xmin, xmax = -xUse/2., xUse/2.
    ymin, ymax = -yUse/2., yUse/2.
    nstepx = 30
    nstepy = 20
    dx = (xmax-xmin)/nstepx
    dy = (ymax-ymin)/nstepy

    for i in xrange(nstepx+1):
        for j in xrange(nstepy+1):
            efieldx.append(multipole.local_values(xmin + i * dx, ymin + j * dy)["Ex"])
            efieldy.append(multipole.local_values(xmin + i * dx, ymin + j * dy)["Ey"])
            x.append(xmin + i * dx)
            y.append(ymin + j * dy)

    # Save data

    np.savetxt(os.path.join(dep, 'Calc_ratio{:f}_dVS{:f}_hVS{:f}_NVS{:.1f}_NHW{:.1f}_rHW{:f}_EDipole{:.1f}_kx{:.1f}.txt'.format(
     ratioVS, dVS, hVS, NVS, NHW, rHW, EDipole, kx)), np.array([x,y,efieldx,efieldy]).T, header="x y Ex Ey", comments='')


#Variables ranges
ratioVS=np.arange(0.436, 0.440, 0.00025)
dVS=np.arange(0.016, 0.018, 0.0001)
#hVS=np.arange(0.0005, 0.01, 0.0005)
NVS=np.arange(29, 30, 1)
NHW=np.arange(24, 27, 1)
rHW=np.arange(0.00020, 0.00022, 0.000001)

def doev(Variables):
    arguments={'ratioVS':0.435,'dVS':0.0089,'hVS':0.0090,'NVS':29,'NHW':25,'rHW':0.000215} #all constants tring to take them as near
     as possible to the minimum
    keys=list(Variables.keys())
    #for naming the folder:
    date=time.strftime("%d %b %H.%M", time.gmtime())
    dep='_'+keys[0]+'_'+keys[1]
    dep=date+dep
    os.makedirs(dep)
    errorlog=open(os.path.join(dep,"error.log"), "w")

    for i in Variables[keys[0]]:
        for j in Variables[keys[1]]:
            try:
                arguments[keys[0]]=i
                arguments[keys[1]]=j
                calc(ratioVS=arguments['ratioVS'], dVS=arguments['dVS'], hVS=arguments['dVS']+0.0001, NVS=arguments['NVS'], NHW=
     arguments['NHW'] ,rHW=arguments['rHW'], dep=dep)
            except Exception:
               errorlog.write("exception for i="+str(i)+" j="+str(j)+"\n")
               continue
    errorlog.close()

#calculate the electric field for all combinatins of ranges and constants
doev({'ratioVS':ratioVS, 'dVS':dVS})
doev({'ratioVS':ratioVS, 'NVS':NVS})
doev({'ratioVS':ratioVS, 'NHW':NHW})
doev({'ratioVS':ratioVS, 'rHW':rHW})
doev({'dVS':dVS, 'NVS':NVS})
doev({'dVS':dVS, 'NHW':NHW})
doev({'dVS':dVS, 'rHW':rHW})
doev({'NVS':NVS, 'NHW':NHW})
doev({'NVS':NVS, 'rHW':rHW})
doev({'NHW':NHW, 'rHW':rHW})
```

Agros2D and Python script for the storage ring (named Multipole-V03.py)

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include "TH2F.h"
#include "TF2.h"
#include "TFitResult.h"
#include "TCanvas.h"

using namespace std;

void Read(const char *filename, float *x, float *y, float *Ex, float *Ey, int *bins);
void adaptdata(float *x, float *y, float *Ex, float *Ey, int *bins, float *dzx, float *dzy);
void stddev(float *dzx, float *dzy, int *bins, const char *name);

int main(int argc, char** argv)
{
  const int N=651;

  int *bins=new int[3];
  float *x=new float[N];
  float *y=new float[N];
  float *Ex=new float[N];
  float *Ey=new float[N];
  float *dzx=new float[N];
  float *dzy=new float[N];
  Read(argv[1], x, y, Ex, Ey, bins);
  adaptdata(x,y,Ex,Ey,bins,dzx,dzy);
  stddev(dzx,dzy,bins, argv[1]);
  return 0;
}

void stddev(float *dzx, float *dzy, int *bins, const char *name)
{
  float meanx=0;
  float stdx=0;
  float meany=0;
  float stdy=0;
  for(int i=0; i<bins[0]; i++)
  {
    meanx+=dzx[i]/float(bins[0]);
    stdx+=dzx[i]*dzx[i]/float(bins[0]);
    meany+=dzy[i]/float(bins[0]);
    stdy+=dzy[i]*dzy[i]/float(bins[0]);
  }
  stdx=sqrt(stdx-meanx*meanx);
  stdy=sqrt(stdy-meany*meany);
  cout<<name<<" Ex= "<<stdx<<" Ey= "<<stdy<<endl;
}

void adaptdata(float *x, float *y, float *Ex, float *Ey, int *bins, float *dzx, float *dzy)
{
  //define borders
  float xmin=*min_element(x,x+bins[0]);
  float xmax=*max_element(x,x+bins[0]);
  int xbins=bins[1];
  float ymin=*min_element(y,y+bins[0]);
  float ymax=*max_element(y,y+bins[0]);
  int ybins=bins[2];
  float dx2=(xmax-xmin)/(2*(xbins-1));
  float dy2=(ymax-ymin)/(2*(ybins-1));

  //alignement:
  xmin=xmin-dx2;
  xmax=xmax+dx2;
  ymin=ymin-dy2;
  ymax=ymax+dy2;

  TH2F *histx=new TH2F("histx", "2D Histo Ex [V/m];x[m];y[m]", xbins, xmin, xmax, ybins, ymin, ymax);
  TF2  *fx = new TF2("fx","[0]+[1]*x+[2]*y",xmin,xmax,ymin,ymax); //linear function for fitting the histogram

  TH2F *histy=new TH2F("histy", "2D Histo Ey [V/m];x[m];y[m]", xbins, xmin, xmax, ybins, ymin, ymax);
  TF2  *fy = new TF2("fy","[0]+[1]*x+[2]*y",xmin,xmax,ymin,ymax); //linear function for fitting the histogram

  int bin; //Fill Histogramm with the electric Field
  for(int i=0;i<bins[0];i++)
  {
    bin=histx->FindBin(x[i],y[i]);
    histx->SetBinContent(bin,Ex[i]);
    histy->SetBinContent(bin,Ey[i]);
    histx->SetBinError(bin, 1.e4); //error unknown at the moment, fit more important in the middle
    histy->SetBinError(bin, 1.e4);
  }//GetBinNumber for SetBinError

  //Fit histogramm
  fx->SetParameters(10500, -10500, 0);
  fy->SetParameters(0,0, 10500);

  //TMinuit *gMinuit=new TMinuit();
```

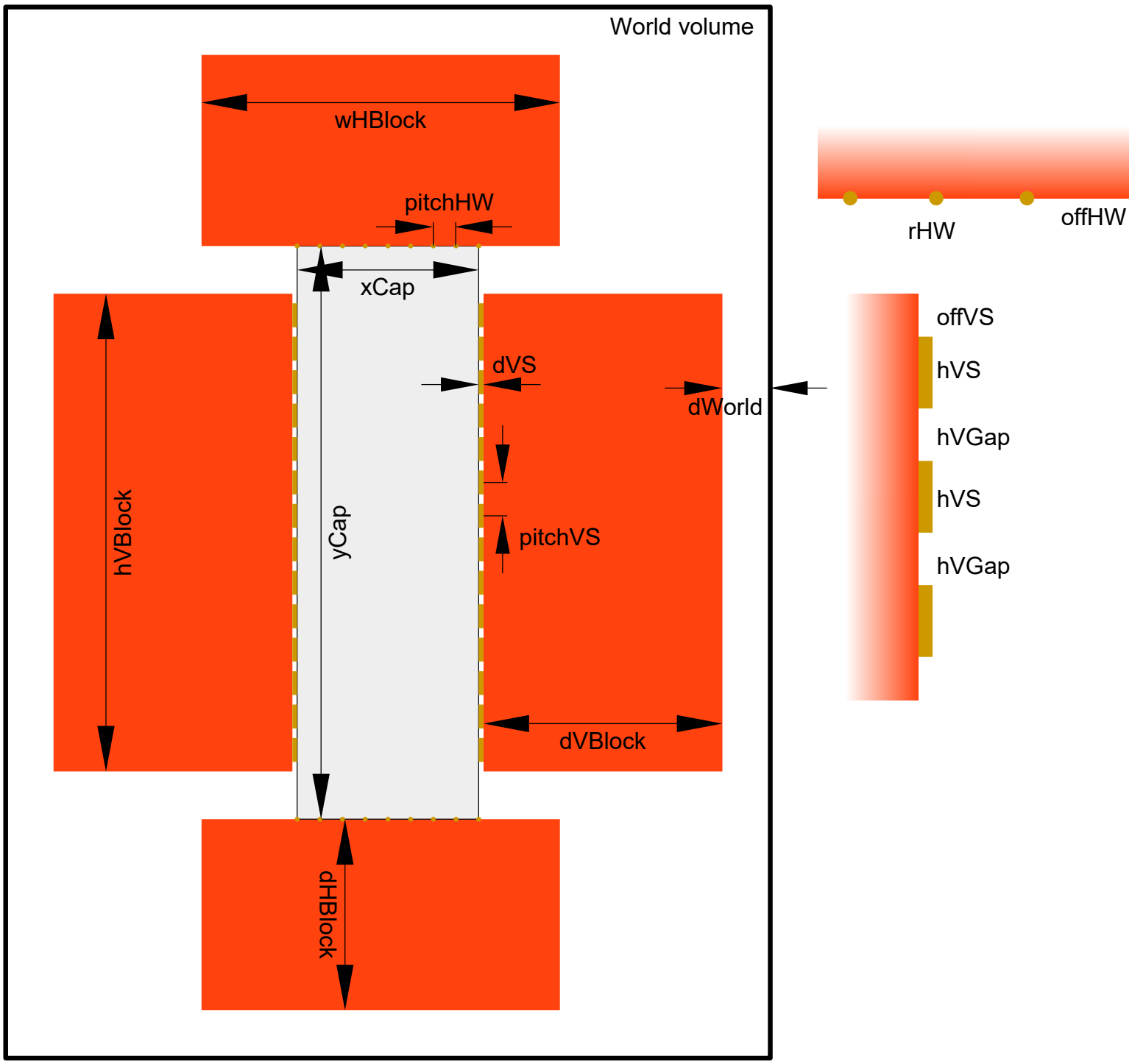
```
  TFitResultPtr rx=histx->Fit("fx","SN");//WL stands for Weighted loglikelihoodmethod, S return Status, N do not store the fit in the plot, q quiet
  TFitResultPtr ry=histy->Fit("fy","SN");

  for(int i=0;i<bins[0];i++)
  {
    bin=histx->FindBin(x[i],y[i]);
    dzx[i]=histx->GetBinContent(bin)-fx->Eval(x[i],y[i]);
    dzy[i]=histy->GetBinContent(bin)-fy->Eval(x[i],y[i]);
  }

  delete gDirectory->FindObject("histx");
  delete gDirectory->FindObject("histy");
  delete gDirectory->FindObject("fx");
  delete gDirectory->FindObject("fy");
}

void Read(const char *filename, float *x, float *y, float *Ex, float *Ey, int *bins)
{
  ifstream File(filename);
  File.ignore(100, '\n'); // skip the first line (till \n)
  float a, b, c, d, alast;
  int i=0;
  int xbins=0;
  while(File >> a >> b >> c >> d)
  {
    x[i]=a;
    y[i]=b;
    Ex[i]=c;
    Ey[i]=d;
    if(a!=alast){xbins++;}
    alast=a;
    i++;
  }
  bins[0]=i;
  bins[1]=xbins;
  bins[2]=i/xbins;
  File.close();
}
```

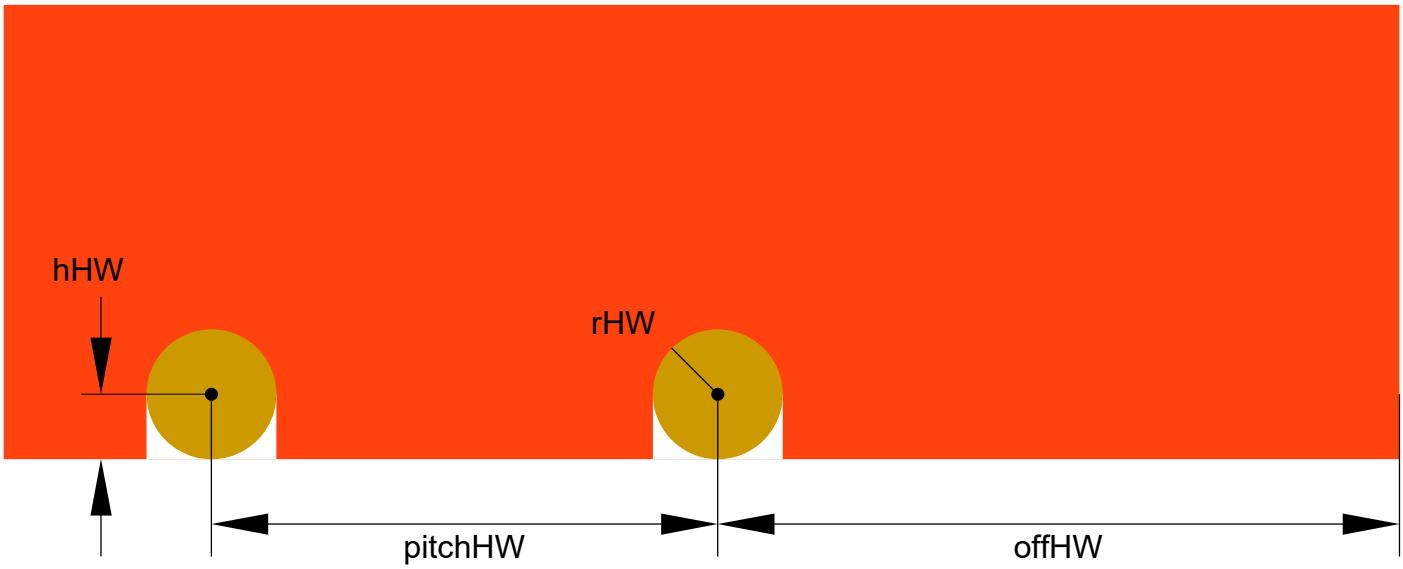
C++ script for reading an ascii file and calculation the deviation from an optimal electric field (named Readdata4.cpp)

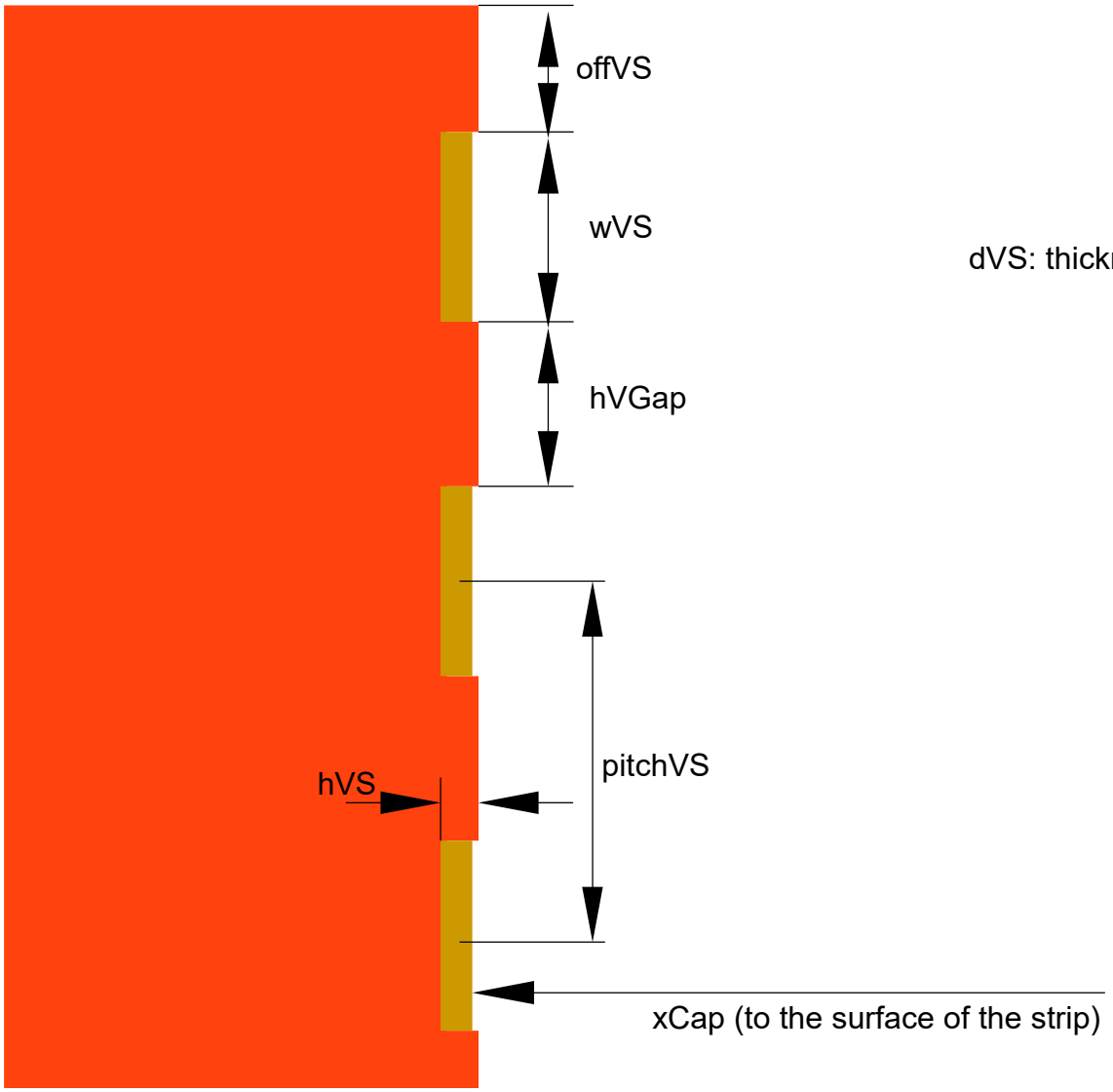
```
#!/usr/bin/python

import sys
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import re
import matplotlib.colors as colors
from collections import OrderedDict

#parameters: ratioVS: 1, dVS: 2, hVS: 3, NVS: 4, NHW: 5, rHW: 6, EDipole: 7, k: 8

def fill(filename):
  data=open(filename, "r")
  zx, zy=[], []
  xypar=[]
  xy=[]
  ratioVS, dVS, hVS, NVS, NHW, rHW, EDipole, k=[], [], [], [], [], [], [], []
  for i in data.readlines():
    line=re.search(r"Calc_ratio(.*)_dVS(.*)_hVS(.*)_NVS(.*)_NHW(.*)_rHW(.*)_EDipole(.*)_kx(.*).txt Ex= (.*) Ey= (.*)", i)
    if line:
      ratioVS+=[float(line.group(1))]
      dVS+=[float(line.group(2))]
      hVS+=[float(line.group(3))]
      NVS+=[float(line.group(4))]
      NHW+=[float(line.group(5))]
      rHW+=[float(line.group(6))]
      EDipole+=[float(line.group(7))]
      k+=[float(line.group(8))]
      zx+=[float(line.group(9))]
      zy+=[float(line.group(10))]
  if ratioVS.count(ratioVS[0])!=len(ratioVS):xy+=[ratioVS]; xypar+=["ratioVS"]
  if dVS.count(dVS[0])!=len(dVS):xy+=[dVS]; xypar+=["dVS[m]"]
  if hVS.count(hVS[0])!=len(hVS):xy+=[hVS]; xypar+=["hVS[m]"]
  if NVS.count(NVS[0])!=len(NVS):xy+=[NVS]; xypar+=["NVS"]
  if NHW.count(NHW[0])!=len(NHW):xy+=[NHW]; xypar+=["NHW"]
  if rHW.count(rHW[0])!=len(rHW):xy+=[rHW]; xypar+=["rHW[m]"]
  if EDipole.count(EDipole[0])!=len(EDipole):xy+=[EDipole]; xypar+=["EDipole[V]"]
  if k.count(k[0])!=len(k):xy+=[k]; xypar+=["k[V]"]
  consts={"ratioVS":ratioVS[0],"dVS[m]":dVS[0],"hVS[m]":hVS[0],"NVS":NVS[0],"NHW":NHW[0],"rHW[m]":rHW[0],"EDipole[V]":EDipole[0],"k[V
      ]":k[0]}
  for i in xypar: del consts[i]
  if len(xy)>2: #sort out if more than two variables
    zh1=0
    zh2=len(xy)-1
    while zh1<zh2:
      for i in range(zh1, zh2):
        diffxy=[]
        for j in range(len(xy[0])):
          diffxy+=[round(xy[zh2][j]-xy[i][j], 8)]
        if diffxy.count(diffxy[0])==len(diffxy):
          xy.pop(zh2)
          xypar[i]=xypar[i]+"="+xypar[zh2]+"-"+str(diffxy[0])+'[m]'
          xypar.pop(zh2)

      zh2-=1
  elif len(xy)==1: #case with only one variable, append array with zeroes
    xy.append(np.zeros(len(xy[0])))
    xypar.append('0')
  elif len(xy)!=2:
    raise Exception('something strange happened perhapts there is no valid data in the readfile')
  lx=list(OrderedDict.fromkeys(xy[0]))
  ly=list(OrderedDict.fromkeys(xy[1]))
  nx=len(lx)
  ny=len(ly)
  filedate=re.search(r"([0-9]{2}) ([A-Z][a-z][a-z]) ([0-9][0-9].[0-9][0-9])", filename)
  filedate=filedate.group(1)+" "+filedate.group(2)+" "+filedate.group(3) #Time
  plotname=xypar[0]+"_"+xypar[1]
  for k in consts:
    plotname+="_"+k+":"+str(consts[k])
  plotname+="_"+filedate
  plotname+='.png'
  return xy[0], xy[1], nx, ny, zx, zy, xypar, plotname

def plot(x, y, z, xn, yn, xpar, ypar, title):
  #centering
  xmin=np.min(x)-(np.max(x)-np.min(x))/((xn-1)*2.) #case xn==1 should normally not happen
  xmax=np.max(x)+(np.max(x)-np.min(x))/((xn-1)*2.)
  if yn!=1:
    ymin=np.min(y)-(np.max(y)-np.min(y))/((yn-1)*2.)
    ymax=np.max(y)+(np.max(y)-np.min(y))/((yn-1)*2.)
  else: #case for only one variable
    ymin=np.min(y)
    ymax=np.max(y)
  plt.figure()
  plt.hist2d(x, y, range=[[xmin,xmax],[ymin,ymax]], bins=[xn, yn], weights=z, cmin=np.min(z), cmax=np.max(z))
  plt.xlabel(xpar)
```

```
  plt.ylabel(ypar)
  plt.ticklabel_format(axis='y',style='sci',scilimits=(1,4))
  plt.ticklabel_format(axis='x',style='sci',scilimits=(1,4))
  plt.title(r'$\Delta Ex + \Delta Ey$')
  plt.colorbar()
  plt.savefig(r"plots/"+title)

x, y, nx, ny, zx, zy, xypar, plotname=fill(sys.argv[1])
plot(x,y,np.array(zx)+np.array(zy),nx,ny,xypar[0], xypar[1], "Ex+Ey"+plotname)
#plot(x,y,zx,nx,ny,xypar[0], xypar[1], "Ex_"+plotname)
#plot(x,y,zy,nx,ny,xypar[0], xypar[1], "Ey_"+plotname)
```

Python script for a graphical display of the standard deviation dependant from two parameters named (findmin3.py)


World volume
wHBlock
pitchHW
xCap
dVS
dWorld
hVBlock
yCap
pitchVS
dVBlock
dHBlock
rHW
offHW
offVS
hVS
hVGap
hVS
hVGap





dVS: thickness of strip