DEVELOPMENT OF AN ITERATIVE METHOD TO SOLVE THE DIFFUSION PROBLEM IN THE PREDESIGN STEP

Author
Mr. Said ABOUALI SANCHEZ

Supervisor
Mr. Vasilis VLACHOUDIS

CERN
EN-STI-EET
Project ID: 14674
August 29th, 2014
## Contents

1 Introduction to the Diffusion Equation 3  
  1.1 Theoretical Concepts .................................................. 3  
  1.2 Diffusion Problem Statement ......................................... 4  

2 Iterative Method 5  
  2.1 Initial information from FLUKA ...................................... 5  
  2.2 Approach to the problem ................................................. 6  
  2.3 Issues to consider ......................................................... 7  
  2.4 Structure of the Main and Waiting Matrices ...................... 8  

3 Description of the code 10  
  3.1 Main program and functions .......................................... 10  
  3.2 Other files ..................................................................... 10  
  3.3 Inputs ............................................................................ 11  
  3.4 Process .......................................................................... 11  
  3.5 Outputs ......................................................................... 13  
  3.6 Plotting the solution ....................................................... 13  

4 Results 15  
  4.1 Definition of the problem no. 1 ...................................... 15  
  4.2 Program solution to the problem no. 1 ............................ 15  
  4.3 FLUKA solution to the problem no. 1 ............................... 16  
  4.4 Definition of the problem no. 2 ...................................... 16  
  4.5 Program solution to the problem no. 2 ............................ 17  
  4.6 FLUKA solution to the problem no. 2 ............................... 17  

5 Conclusions 19  
  5.1 Program performance analysis ......................................... 19  
  5.2 Advantages of the program .............................................. 19  
  5.3 Limitations of the program .............................................. 19  

6 Future Projects 20
Acknowledgments

On June 30th this project was proposed to me. The first thought that came to my mind was that it was going to be really hard to solve it. I didn’t know how to manage it. Little by little I started overcoming obstacles and obtaining results.

Today, exactly 9 weeks later, I’m glad to present the results in this report. I’ve enjoyed a lot my time working at CERN (even the difficult moments with this project) and that’s the reason why I’m writing this section: To thank to all the people who made possible for me to live this experience. There are two main groups: the ones who helped me to come here and the ones who helped me while I was at CERN. Without the help of all of you, this would not have been possible.

As regards the first group, I would like to thank all my relatives, but especially my father, Rachid Abouali, my mother, Maria Jose Sanchez and my sister, Maria Abouali (and her husband, Marcelo Fuentes, who joined this group some years later). They have been supporting and helping me since I was a child. Obviously, without them I would not be the person I am, and all my past, present and future achievements have been, are and will be possible thanks to them.

There are also other people who has encouraged my decisions. Among them I should mention my professors Jose Luis Perez Aparicio, Sergio Hoyas and Enrique Ballester, who also helped me to make my dream of being at CERN true. I don’t forget any professor I had because all of them taught me important things.

As regards the second group, I have to congratulate and say thank you to the Human Resources Group, which organized superbly my stay at CERN and they were always willing to help. Special thanks to my supervisor, Vasilis Vlachoudis, who trusted me from the very beginning and from whom I’ve learned lot of things. Moreover, it’s impossible to forget to thank Leonel Morejon, my coworker, who has been always there to help me with any issue, both from the academical and from the personal point of view.

Finally, before starting "diffusing" the information of these 9 weeks and after thanking all these great people, I’d like to ask for forgiveness to all the people I have not mentioned but were also important in my life (relatives, professors, friends and colleagues).
1 Introduction to the Diffusion Equation

Nowadays, in most of the physics and engineering design problems it’s essential to know both the particle or heat diffusion as well as the heat deposition in the area of interest. Therefore, understanding the way the particles or heat diffuse is an important step in the design.

Note that the solution and code development for the diffusion we are going to study in this report has been done in a general way. Thanks to this fact, the program will be valid for particle, heat diffusion and any kind of diffusion problem in which we are able to understand the physics behind the problem before applying this code.

![Energy deposition distribution in a piece](image.png)

**Figure 1:** Energy deposition distribution in a piece

In the Figure 1 the heat deposition in a piece is represented.

1.1 Theoretical Concepts

In order to be able to understand basically the development of the code it’s basic to have a previous idea of the main theoretical concepts and equations used to solve the heat diffusion problem.

These equations describe the behavior of a material (or a magnitude) undergoing diffusion or diffusive-like processes. In a general way, the diffusion equation can be written as follows:

\[
\frac{\partial \phi(r,t)}{\partial t} = \nabla [D(\phi, r) \nabla \phi(r,t)]
\]  \hspace{1cm} (1)

where:

- \( \phi(r,t) \) is the variable under study and depends upon the position \( r \) and the time \( t \)
- \( D(\phi, r) \) is the diffusion coefficient and depends upon the value of the variable and the position
Solving this equation will allow us to know the distribution of the variable under study for any time at any position we are interested in.

For example, in the particular case of heat diffusion \( \phi(r, t) \) is the Temperature \( T \) and \( D(\phi, r) \) is the thermal diffusivity coefficient \( \alpha \). Then, the equation for this case can be expressed as:

\[
\frac{\partial T}{\partial t} = \nabla \left[ \alpha (\nabla T) \right]
\]  

(2)

1.2 Diffusion Problem Statement

One of the most important and difficult steps in the design process is the predesign. In this step, the main goal is to obtain the geometry characteristics and some other features to solve the problem we want to solve. Thus, reducing the duration of this phase is one of the challenging objectives in the actual engineering design process.

As regards our study, following the energy deposition calculations with the means of the FLUKA code, a heat diffusion code is usually needed to calculate the evolution in time of the temperature distributions as well the associated stresses on the object under study. As a preliminary optimization step in the object design process, in this project we are proposing the development of a new method implemented in a code which allows us to obtain the solution for this equation in a complex geometry by using the information given by the FLUKA code for a simple geometry. By this way, we will be able to reduce the predesign phase duration.

Highlight again that from now on we are going to focus on the particle diffusion since it is "easier" to understand. However, the code is equally applicable for heat diffusion or any kind of diffusion.
2 Iterative Method

The main goal is to obtain the solution of the problem in an object by knowing the behavior of a small part of this object. This initial information is going to be given by FLUKA. Then, the developed code will use this information to achieve the energy deposition distribution in the object under study.

Throughout this report, some of the notation will be introduced and in some explanations the notation of the code will be used in order to make it easier the subsequent understanding of the code.

2.1 Initial information from FLUKA

By means of the FLUKA code we obtain the way the particles diffuse in a small cube. The choice of the size and the material will depend on the problem we want to solve and the accuracy we want to achieve.

Once we have decided these conditions, three matrices are obtained from FLUKA code. These matrices contain all the needed parameters to know how the particles diffuse in the cube depending on the energy of the particles. The size of the matrices is \( m \times m \), where \( m \) is the number of different energy domains in which we divide the energy axis. Remark that this division of the energy domains and the FLUKA execution is the part of the process in which the physics behind the problem has to be taken into account. Then, all the physics will be behind the matrices and by understanding the matrix structure we are able to build the code that solves the whole problem.

After colliding a beam of particles with different energies with the cubic target, the three obtained diffusion matrices are:

- \( T \rightarrow \) Contains information about how the particles go through the cube and leave it by the opposite face.
- \( R \rightarrow \) Contains information about how the particles are reflected when they hit the cube.
- \( S \rightarrow \) Contains information about how the particles go out of the cube by one side. Assuming isotropic behavior of the material, the \( S \) Matrix will be the same for the four sides. In case no assumption can be done, there would be a need to obtain a different \( S \) Matrix for every side.

For every diffusion matrix the internal structure follows the same rules: \([X] = X_{i,j}\), where:

- \( X \) can be \( T \), \( R \) or \( S \) depending on the face of the cube we are interested in
- \( i \) is related to the type of energy of the input particles in the cube
- \( j \) is related to the type of energy of the output particles going out of the cube by the corresponding face.
- \( i,j \in 0,1,2...,m-1 \)
• $X_{ij}$ is the fraction of the particles with energy $j$ that goes out the cube in comparison with the amount of particles going in with energy type $i$.

As we will see later, thanks to this way of arranging the matrices, the amount of output particles of each energy type will be computed with a matrix product.

2.2 Approach to the problem

The use of an iterative method is proposed to obtain the final solution from the diffusion matrices given by FLUKA.

First of all, it’s important to understand the way of building the mesh. The cube is built by combining the small cubes (also known as "Elements" in our context). So, the variables of the programs $nx$, $ny$ and $nz$ control the mesh size in every axis, making it possible to build a prism of the desired size just by the combination of the cubic elements. The numbering of each cube (from 1 to the total number of cells increasing 1 unit as we increase the $x$ coordinate) allows us to be able to control the output direction as a function of the input direction.

After numbering the cells, there is the possibility of assigning one different material for each cell. The only thing we have to take into account is that to be able to assign a material to a cell we must have the three diffusion matrices for this material.

Notice that all the mesh and material information is stored in the main matrix, whose structure is going to be explained in the next sections.

Once the meshing has been done and the materials have been assigned the diffusion problem can be solved as follows. Particles are introduced in one cell with one direction. Then, the diffusion starts in this cell, which transmit particles to the six adjacent cells. In order to make this diffusion possible, the three diffusion matrices are used. Then, in the next iteration there will be six cells with an input that has to be processed as it was done with the first one. Again, by multiplying the transfer matrices by the amount of particles we will obtain the outputs in the six faces of the small cubes, which will be the inputs for the next six adjacent cells. This process is repeated in an iterative way till the program reaches the Stop condition. The setting of this stop condition is an open problem, that can be solved in different ways and that will be explained later.

In every iteration, the input of a cell is processed, obtaining the outputs. Then, it’s possible to calculate either the particle flux and the energy deposition at each cell just by making the difference between what went into and what goes out the cell.

Finally, once the Stop condition has been reached, it’s considered that the diffusion problem has finished and the solution is the deposition in every cell, thus having the energy deposition distribution along the whole object.

In the following subsections and sections more details about the way of solving the problem and coding will be explained.
2.3 Issues to consider

The main function of the program has already been explained. However, there are many details that must be taken into account to understand properly how the program was built and how it works. The basic ones are:

- The way of numbering we have explained makes it easier the computations and requires less memory space. However, it’s a bit difficult to understand for the user. For this reason, an external function was created ("cart2i") which allows the user introduce the Cartesian coordinates of the cube $x, y, z$ and obtain automatically the corresponding number $i$ which will be used by the internal program.

- Due to this way of numbering, errors may appear when trying to access one of the outermost cells. This errors can be accessing a wrong cell or exceeding the matrix size (which will be introduced later). In order to solve this problem, the program generates automatically an external wall, adding cells in the outer layer. This outer cells are obtained by an algorithm and they are then identified in the main matrix by setting the third column to a defined value. Then, we are able to know when the particles are entering this cells and thus knowing when the particles have entered this external virtual wall. Therefore, all the particles and energy that enters the external wall are considered as particles that went out the original object, so they are stored in a variable that counts the particles or energy that left the prism. That means that thanks to this external wall we are not only able to avoid program errors but also store the amount of particles that are leaving the object. As a predesign step, it is interesting to have this kind of variables since they are going to give us information about how many particles are leaving the object and then we will be able to have an idea of how we can modify the mesh in order to fit the whole diffusion problem (or maybe reduce the mesh if there is no need of so many cells).

- The stop condition is an open question. it can be solved by several ways. In case there was not stop condition the program would run and diffuse without stopping. Then, this condition is given by the variable "waytol" and it’s controlled by the variable "tol", which acts as a threshold. Highlight that this condition applies to a cell. It means, they stop the diffusion in the cell that fits the requirement of the stop condition. However, the diffusion can continue in other cells. Then, the whole program will end when all the cells have achieved the stop condition and there are not more inputs to process. Up to now, there are three ways of stopping the program:

  - waytol = 1 $\rightarrow$ When the amount of input particles in a cell is lower than the threshold it’s considered that there is not input in this cell.
  - waytol = 2 $\rightarrow$ When the energy deposition in a cell is lower than the threshold it’s considered that there is not more outputs in this cell and the diffusion ends here.
  - waytol = 3 $\rightarrow$ When the relative energy deposition in a cell with respect the energy deposition that was before in this cell is lower than the threshold it’s considered that there is not more outputs in this cell and the diffusion ends here. It has been found after several simulations that this is the best stop conditions since it gets more accurate results in less time. This is because with this condition we ensure that the deposition is stabilized at every cell.
For this reason, the current version of the program is working with the third way of
defining the stop condition. However, the user can modify it easily as it has been
created as an input of the program.

- Finally, the last important detail is the need of an auxiliary matrix. As the processing
of a cell means the generation of inputs in six adjacent cells, there is some point in
which one cell may have several inputs in different directions. To solve this issue, a
Waiting Matrix (WM) has been created. When there is an input in a cell that has not
been processed yet and appears to be a new input in the same cell, this new input is
stored in the WM. Then, when the input that was already in the Main Matrix (MM)
is processed, the code retakes again the inputs that are waiting to be processed in the
WM.

These issues are the main issues we have to take into account when studying or trying to
modify the code. However, there are much more small details that must be taken into account
but at the end all of them are related by one way or another to these ones we have already
explained

2.4 Structure of the Main and Waiting Matrices

Up to now, it has been explained the main bases of the approach to the problem and
some of the issues we have to take into account. Now, before introducing the way it has
been coded, a brief explanation of the structure of the two matrices that have been already
introduced is going to be done.

Basically, in the Main Matrix all the relevant information for the diffusion is stored. It’s
then the Matrix in which we are going to store the inputs and the solutions. On the other
hand, the Waiting Matrix stores information that has to be processed by the Main Matrix
but that we were not able to store because there were some inputs in the same cell that had
not been processed yet.

As regards the Main Matrix:

- The size of this matrix is fixed and it does not vary
- Each row of the MM corresponds to a cell. Then, the row $i$ is related to the cell $i$
- Each column is related to one different variable of the program. The number of columns
can vary depending on how many energy types there are. In the program, the number
of different energy types is stored in the variable $m$. The columns are:
  - Column 1 $\rightarrow$ Active column. If the value is 0 it means that there are not inputs
to be processed in the corresponding cell. If the value is 1 it means that in the
other columns there is information about some input that has not been diffused
yet.
  - Column 2 $\rightarrow$ Waiting inputs. This column stores how many waiting inputs in
a cell are in the WM. Every time there is an input in a cell that already has an
input without processing, one unit is added to this column. After that, every time
a waiting input is retaken, one unit is subtracted from this column
– Column 3 → This column stores the number of the cell corresponding to the row.
  Take into account that this is not needed. However, it is useful to have this column
  when studying the performance of the code and when checking some things. This
  means that once the code is completely finished this column will be removed to
  save memory space.

– Column 4 → This column stores the direction of the input in this cell (in case
  there is an input). The possible directions are:
  * +1 : to the positive x axis
  * −1 : to the negative x axis
  * +nx : to the positive y axis
  * −nx : to the negative y axis
  * +nx · ny : to the positive z axis
  * −nx · ny : to the negative z axis
  * 1.5 : this is a input direction which never will be possible. Then, this is the
    way of identifying the cells that belong to the external wall. Note that in this
    cells we don’t need to know the input direction because we are not going to
    diffuse the particles as they are already considered as an output. Then, it’s not
    needed to add an additional column for the wall and we can use this column
    to identify the wall

– Column 5 to 4+m → this columns store the amount of particles at each energy
  level that are the input of the corresponding cell. Remember that m stands for
  the number of different divisions in the energy axis.

– Column 5+m → In this column the particle flux or energy deposition is stored.
  Take into account that the value of the deposition in one iteration is added to the
  value of the deposition that was before. This means, it’s a cumulative column.

– Column 6+m → In this column we can find the material ID for the cell. At the
  beginning of the program, there is a dictionary (or database) in which an ID is
  assigned to every material.

• When the stop condition is given and it’s considered that the computation has finished,
  the solution is the column in which the energy deposition is found

As regards the Waiting Matrix:

• The size of the WM is not fixed, and it can vary as a function of how many waiting
  inputs are.

• The waiting matrix has as many rows as waiting inputs are. Every row contains the
  information of an input that could not be processed by the MM when it was generated

• It has 2+m columns with all the important information:
  – Column 1 → It contains the number of the cell in which there is the waiting input
  – Column 2 → It contains the direction of the input. The direction possibilities are
    the same than we studied for the previous matrix
  – Column 3 to 2+m → These columns store the amount of particles at each energy
    level for the waiting input in the corresponding cell
3 Description of the code

Once it has been explained basically the way of solving the diffusion problem, in this section we are going to study the basic structure of the code

3.1 Main program and functions

The structure of the program is as follows:

- **main_program**: is the file in which all the inputs are introduced, calls the functions that solve the diffusion problem and gives the desired results. The important outputs are automatically stored in a text file

  - **program.diffusion**: is the function that contains all the needed code and also calls some small functions. Its operation will be explained in following subsections

  * **definitions.dif_cell**: is the function that computes the matrix product to obtain the amount of particles going out of each face of the cube by knowing as inputs the particles going into the cube and all the diffusion matrices. Note that the outputs can be in terms of particles or in terms of energies. In the case of energies we also need the three transfer matrices but with the corresponding coefficients for this case.

  * **definitions.cart2i**: is the function that obtains the numbering of the cell as a function of the Cartesian coordinates of this cell

- **plot_solution**: is the script that reads the text file generated by the main_program and plots the deposition at every horizontal or vertical plane

3.2 Other files

There are also some other text files that are required for the proper running.

- The 3 diffusion matrices in terms of particle output for every material we are going to assign:
  
  - MATERIAL_T.txt
  - MATERIAL_R.txt
  - MATERIAL_S.txt

- The 3 diffusion matrices in terms of energy output for every material we are going to assign:

  - MATERIAL_T_Energy.txt
  - MATERIAL_R_Energy.txt
  - MATERIAL_S_Energy.txt

- Evector.txt: the vector which contains the energy of a particle in every energy level in which the energy level was divided
Note that for the first iteration we give as an input the amount of particles, so we need the "Evector.txt" to obtain the energy going in a cell. After this iteration, with the particle matrices we obtain the particles going out the cell (which will be the inputs for the adjacent cells) and with the energy matrices we obtain the energy outputs (which will be the energy input for the adjacent cells), thus being able to compute the particle flux or energy deposition at every cell.

3.3 Inputs

The inputs that the user must introduce in the main program are:

- \( nx \): number of cubes in the "x" axis
- \( ny \): number of cubes in the "y" axis
- \( nz \): number of cubes in the "z" axis
- \( \text{energies} \): vector with the different energy types in which the energy domain was splitted
- \( \text{initial\_inputs} \): matrix in which we can introduce the inputs that are in several cells. Each row stands for an input. For every row, the first column is the number of the cell in which we introduce the particles, the second one is the direction of the input (following same rules as before) and the rest of the columns contain the amount of particles for each energy level
- \( \text{normalize} \): this variable must be set to 1 if we want the deposition normalized with respect to the input energy or to 0 if we want the absolute value of the deposition
- \( \text{way\_tol} \): number from 1 to 3 that selects the way of stopping the diffusion in one cell that were explained in previous sections.
- \( \text{tol} \): if defines the threshold for the different stop conditions. It must be taken into account the chosen variables to adapt this one and fulfill the accuracy requirements
- Finally, the user also has to provide with all the matrix files for every material

3.4 Process

Once the inputs have been introduced the program is ready to run. The basic procedure of the program is the following:

First, the Main Matrix is created and filled with zeros. The external virtual wall is also built (adding cells in the outer layer) and this cells are identified as it was previously explained in the corresponding column of the MM. The next step is to fill the MM with the input information that the user has introduced. Then, the inputs are transferred to the MM and the rows that have an input set their active column to 1. Moreover, in this point of the program the user is still able to assign to each cell a different material, just by coding the required conditions of this assignment.

At this point, the main loop is prepared to start. This main loop is a while loop, and the variable that controls it is the "active\_sum" variable. This variable computes the sum of the
active column at the end of every iteration, and the while loop finishes when this variable is not greater than 0. This means that the iterations are going to continue while there are still inputs that have to be processed.

At every iteration, the code looks for the rows of the matrix that have the active column set to 1 and once it finds it starts the diffusion code. The code checks the input direction column and in case the value of this element is 1.5, it is known that this cell belongs to the wall. Then, all the amount of particles there are stored as an output of the cube and this cell does not diffuse anything else.

In case this cell does not belong to the wall, the code continues running. All the values of the row are analyzed. Depending on the material ID of this cell, the code loads the corresponding diffusion matrices for this material. Then, it also reads the input direction and the amount of particles for each energy level and stores this information in a vector. Once we have this information in the correct format, the function "definitions.dif_cell" is called. The input particles are known, and by doing the product of this vector with the particle matrices and the energy matrices we are able to calculate the amount of particles and energy going out the cube. In the case of the amount of particles, the information of each face is stored in vectors. Moreover, as we have already calculated the energy going out by each face of the cube, we sum this energy to the deposition column of the cell in which it is entering. Moreover, as it has been calculated all the energy that has gone out of the current cell, we subtract this amount from the deposition column of this row (take into account that the input energy was added when the input was generated by some of its adjacent cells and the output energy is subtracted when the diffusion at this cell is computed. The overall balance is the energy deposition at this cell).

Notice at this point that before doing that, the stop conditions for the cell are checked depending on the way the user chooses when building the inputs. In case the stop condition for each case is fulfilled, the diffusion is stopped, no more cells are activated and the active column of this cell turns into 0.

Once we have arrived to this point, the diffusion in the cell has been performed and it’s time to fill the MM with the new information. There are two possibilities:

- In case the row for the new cell we are trying to access has the active column set to 0 means that there is not any input. So, the input particle information is stored depending on the input direction. There is an algorithm with fills the row with the correct vector for each case (T,R or S) depending on the input direction that was in the previous cell. After doing that, as there is already an input, the active column is set to 1.

- In case the row for the new cell we are trying to access has the active column set to 1 means that there is some input at this cell that has not been processed yet. Then, the new information is stored in the WM by means of waiting values and waiting vectors that allow us not to lose any important information. Additionally, the code also checks if there is already an input with the same direction in the WM, and in case there is, the values are added to avoid matrix growing too much. In case there is not, a new row is added.

After doing that, the new input directions for every cell are calculated in order to be able to continue the diffusion in the next iterations. Moreover, as this cell has already been
processed its active column turns into 0 and the other elements of the row are cleaned except the material ID which is maintained constant and the deposition which is a cumulative value.

Once the active value has been set to 0, the code checks the value of the second column, which contains information about how many waiting input are in the WM. If this value is greater than 0, we look for a waiting input in this cell in the WM and we retake it and fill again its corresponding row in the MM and the active column is set 1 again. After that, one unit is subtracted to the value of the number of waiting inputs and this row is deleted in the WM.

To finish with the iteration, the code obtains the sum of all the values in the active column, and in case there are still cells to be processed, the sum will be greater than 0 and more iterations will be performed.

After some time and depending on the threshold, there won’t be more inputs to process, so the sum of the active column will be equal to 0 and the program will have finished. Now, we are able to obtain the solution to our problem.

### 3.5 Outputs

Once the program has finished the computation, all the important information is stored in the MM and in some other variables (we can even add more variables to record other interesting points). Particularly, the outputs that the main program gives to us are:

- **solution**: is the vector that stores the values corresponding to the energy deposition at every cell. In other words, it gives us the information about the heat deposition distribution in the object.

- **A**: is the whole MM. This is not really needed, but the program keeps it as an output in case the user wants to check something in this matrix.

- **amount_out_walls**: it’s the value of the variable that stored the amount of energy that went out of the object (when the input entered a cell belonging to the external virtual wall).

- **iterations**: records the number of iterations that were needed to obtain the final solution. It’s then a variable that is related to the computational time needed for the solution to converge.

As it has already been mentioned, some other interesting outputs can be stored easily just by adding some lines in the code.

Finally, a text file is created with all the information concerning the energy deposition distribution in order to be able to study and represent it later.

### 3.6 Plotting the solution

The script that plots the solution simply reads the solution text file generated and stores all the information in a new matrix. Then, the user is able to select the horizontal or vertical plane in which he wants to display the energy deposition. Then, the program applies an
algorithm to select only the cells at this planes, sort the information and plot it to have a visual estimation.

Remark that for more accurate comparisons, it is not enough by comparing the plots. In this cases, all the accurate results and outputs can be read directly from the solution text file.
4 Results

The main functioning of the program has been explained in the previous sections. This code is able to solve any problem (dealing with the computer resources restrictions) for any mesh size and any materials while its diffusion matrices are available. In this section, a small particular problem is proposed, its solution is obtained and compared with the FLUKA solution.

4.1 Definition of the problem no. 1

The main features of this problem are:

- A beam of neutrons colliding with a cube in the midpoint with direction "y"
- The material of the cube is water
- The size of the cube is 6cm x 10cm x 6cm
- The cube is composed by 1cm small cubes. This means that \( n_x = 6, n_y = 10, n_z = 6 \)
- The desired solution is the particle flux

4.2 Program solution to the problem no. 1

After running the program under this conditions, the solution is attached below.

---

Figure 2: Particle flux obtained with the program
4.3 **FLUKA solution to the problem no. 1**

The solution obtained with FLUKA code is attached below.

![Particle flux obtained with FLUKA](image)

**Figure 3:** Particle flux obtained with FLUKA

Comparing the Figure 2 with the Figure 3, it can be stated that the shape of the solution fits really well and the values coincide in the range of the order of magnitude.

Just note that we are representing the deposition in the plane corresponding to the cell which received the initial beam. However, the program gives the solution in the whole cube and it’s possible to represent the deposition at any plane (both horizontal and vertical) just by changing the inputs of the Plot script.

4.4 **Definition of the problem no. 2**

The main features of this problem are:

- A beam of neutrons colliding with a cube in the midpoint with direction "y"
- The material of the cube is water
- The size of the cube is 10cm x 10cm x 10cm
- The cube is composed by 1cm small cubes. This means that \( nx = 10, ny = 10, nz = 10 \)
- The desired solution in this case is the energy deposition
4.5 Program solution to the problem no. 2

After running the program under this conditions, the energy deposition in the horizontal plane (beam direction) is represented in Figure 4 and in the vertical plane perpendicular to the beam axis is attached in the Figure 5.

![Figure 4: Energy deposition in the horizontal axis obtained with the program](image)

![Figure 5: Energy deposition in the vertical axis obtained with the program](image)

4.6 FLUKA solution to the problem no. 2

After running FLUKA under this conditions, the energy deposition in the horizontal plane (beam direction) is represented in Figure 6 and in the vertical plane perpendicular to the beam axis is attached in the Figure 7.

The solutions obtained with the program coincide again in shape with the FLUKA solution. The order of magnitude is again similar too. However, far from the center of the
cube, the solutions diverge a bit. This makes it necessary to run with a higher accuracy, thus spending much more computational time. The solution to this problem will be found in next sections, where we will propose some possible ideas to make the program faster.

Anyway, there is a need of doing more comparisons between the program and FLUKA in order to figure out better the performance of the code.
5 Conclusions

Throughout this report it has been studied the code of the program, what it does, how it does and its performance. In this section, we will try to summarize all these information and also state the advantages and limitations of it.

5.1 Program performance analysis

The program is able to give a converged solution with relatively high accuracy in approximately 2 to 5 minutes and a rough idea (but maybe with enough information for the first stages of the design process) of how the solution looks like in just 5 to 10 seconds. This means that it fulfills the initial purpose of the project, which was reducing the time of the predesign step. However, being ambitious, this is not enough and the main goal now would be to obtain the high accuracy solution that is obtained today in 2 to 5 minutes in less than 2 seconds.

Note that despite it’s a really ambitious goal, it’s more than possible. Just remember that the first versions of the program spent around 2 minutes to give the solution for a specific problem. In this initial versions the waiting matrix grew every time there was a waiting input. In the current version, before storing the waiting input the code checks if there is already a waiting input with the same input direction in the WM to avoid uncontrolled growing. Only this change allowed us solving the same specific problem in 3 seconds instead of 2 minutes and the high accuracy problem in 2 to 5 minutes instead of one hour.

As a conclusion, this means that with some of the proposed improvements in next sections we will be able to reduce even more the computational time.

Nevertheless, there are some other points that we have to take into account before applying this code no matter how fast it is: the advantages and limitations of the method.

5.2 Advantages of the program

The main advantage has already been explained. The code allows us to obtain an approximate solution for the predesign step, one of the most difficult steps in the design process. The most important fact is that this solution is achieved faster than it would be with FLUKA code.

5.3 Limitations of the program

One of the main limitations of the program is the accuracy of the solution. However, this is not a problem, because the purpose of this code is to give a fast solution, not an exact one. For this reason, we must take into account that it’s only valid for the predesign step and will give us an idea of how we have to focus the problem in the next stages of the design process (in which this code won’t be applied).

Another limitation is that the code does not contain the physics behavior. The physics behavior is contained in the diffusion matrices and this is the reason we have to be careful. If the diffusion matrices don’t fit with the physics of the real problem, the code will give wrong results even it’s working properly. Then, before applying this program the user must ensure that the matrices fulfill the physics requirements for its specific problem.
6 Future Projects

In physics and engineering, after solving a problem you find that during the path you followed lots of new problems and ideas came out. This project has not been an exception. In the following list we summarize some new ideas we propose for future projects:

- Study another ways to set the Stop Condition in order to make the program more efficient and to obtain accurate results in the whole object.

- Divide the geometry into smaller cubes and send each cube to a different processor. Then, each cube shares the information of the amount of particles going out by each face with the corresponding adjacent cube, which takes it as a new input and vice versa.

- Parallelize the code, so as the main matrix of the program can be accessed simultaneously by different cores only by selecting the rows of the matrix in which the active column is set to 1.

- Identify simmetries and take advantage of them. We would be able to reduce the computational time by a factor of 4. It would be important to take into account the new boundary conditions we have to impose.

- Combine all the previous ideas to make a faster program. It is roughly estimated that we would be able to diminish the computational time by a factor of N, where N is defined as:

\[
N = 4 \times n \times k
\]

where the factor of 4 is due to the simmetries, n is the number of processors and k is the number of small cubes in which we can divide the whole geometry.

- After all these improvements, the idea would be to introduce this code into Flair. This program will be able to give estimated results in real time and then the user can see approximately how it is going to be the solution. Thanks to this fact, the user has a previous idea of the mesh type and size he has to use, thus reducing the time of the predesign step (which is one of the most difficult tasks in this kind of problems).

- In order to accomplish the previous objective, a database with the diffusion matrices (T,R,S) must be built. Another possibility would be building a parallel program which takes as an input the size of the cube and runs FLUKA to obtain these matrices.

- The use of non-isotropic cubic elements, in which the diffusion is not the same by the four sides. Then, modify the code in order to be able to deal with the six diffusion matrices (T,R,S1,S2,S3,S4).

- Build an iterative method which takes the same inputs than before and automatically gives you the best mesh size (number of divisions at each axes) you can use to solve the problem. Some of the variables (for example the one that stores the amount of particles or energy leaving the prism) that were previously explained may be helpful in this point. That would be a turning point in the design field since the predesign step would be done almost automatically.
Finally, once the program is well introduced in the Flair interface, we could start trying to build more complex geometries with cubes and maybe try to use different elements. For example, switching from cubic elements to tetrahedral elements will allow us to fit better curved geometries.

All the ideas we have proposed are focused to improve the performance of the program and widen the range of use of this program. The program can be used both for particle diffusion problems but also for heat diffusion treatment. This fact opens a new door related to my main field of study:

- In the aerospace field the heat diffusion problem and the stress analysis due to it is a very important issue. Then, this program could be also used as a predesign step in the heat analysis needed in aerospace and aeronautics. The idea is to make the needed changes in order to make this program fit the aerospace requirements.

- However, it could be also applied to several topics in the aerospace engineering field. For example, for the stress analysis, vibrations analysis or any predesign analysis in which a kind of energy or matter transport is involved. Then, the study should focus on the way to obtain the matrices and the problem is to represent the physical behaviour in the matrix. Once we are able to obtain the matrices, the code can be easily applied. Again, with these improvements the predesign step duration (which is critically important this field) is significantly reduced.