UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia “Galileo Galilei”

Corso di Laurea in Fisica

Tesi di Laurea

Validation of a mathematical model for the estimate of intracellular calcium ion fluctuations

Relatore
Prof. Mario Bortolozzi

Laureando
Alessandra Sabatti

Anno Accademico 2018/2019

Abstract

Calcium ion (Ca^{2+}) acts as a vital second messenger within living cells and the experimental measurement of its concentration is critical in Biology. During muscular contraction, nerve impulse transmission and many other physiological events, Ca^{2+} forms micro- and nano-domains within the cell cytoplasm which can be described solving reaction-diffusion equations between Ca2+ and its buffers interacting with cell boundaries. Innovative surface and volume subdivision algorithms based on unstructured three-dimensional mesh generation can be applied to locally increase or decrease the accuracy of the differential equation solution. This thesis work has developed and validated a simple and computationally inexpensive diffusion algorithm to simulate diffusion in an arbitrary volume subdivided into tetrahedral voxels. The algorithm was implemented into the user-friendly software SimulCell from Bortolozzi lab and utilized to simulate the real calcium concentration in cell micro- and nano-domains.

Lo ione calcio (Ca^{2+}) svolge una funzione di secondo messaggero nelle cellule, e la misura sperimentale della sua concentrazione è estremamente importante in biologia. Durante la contrazione muscolare, la trasmissione degli impulsi nervosi e molti altri eventi fisiologici, il Ca^{2+} forma all'interno del citoplasma cellulare dei micro- e nano-domini, che possono essere descritti tramite la risoluzione di equazioni di reazione-diffusione tra il Ca^{2+} e i suoi buffer, mentre interagiscono con le superfici della cellula. Innovativi algoritmi di suddivisione delle superfici e dei volumi basati sulla generazione di mesh non strutturate tridimensionali possono essere applicati per aumentare o diminuire localmente l'accuratezza della soluzione delle equazioni differenziali. Questo lavoro di tesi ha sviluppato e validato un semplice e computazionalmente economico algoritmo di diffusione che permette di simulare la diffusione all'interno di un volume arbitrario suddiviso in voxel tetraedrici. L'algoritmo è stato inserito in SimulCell, un software user-friendly del laboratorio del professor Bortolozzi, ed utilizzato per simulare la concentrazione di calcio reale all'interno di micro- e nano-domini cellulari.

Index

1 Introduction

1.1 Calcium in biological systems

Calcium ions (Ca^{2+}) regulate multiple processes in cells, such as intracellular signal transduction mechanisms, ranging from excitation-contraction coupling, to synaptic transmission and genetic transcription.

It works as a second messenger inside the cell. The canonical way of transmitting messages to cells involves the interaction of "first messengers" with plasma membrane receptors. The interaction activates the production of diffusible "second messengers" that convey the information to cellular targets.

It carries messages to virtually all-important functions of cells. Its binding by complex molecules is particularly easy. A large group of proteins has evolved to bind or transport calcium. They contribute to buffer it within cells, but several also decode its message for the benefit of the target.

Although essential to the correct functioning of cell processes, if not carefully controlled spatially and temporally within cells, it generates variously severe cell dysfunctions, and even cell death. (Carafoli & Krebs, 2016).

1.2 Calcium imaging

Optical measurement of the intracellular concentration of Ca^{2+} ($[Ca^{2+}]$) is paramount to understanding cell physiology and function.

Several molecular probes, namely fluorescent dyes, capable of sensing the local ion concentration with high selectivity, have been developed over the last twenty years. Chelation is the binding or complexation of a bi- or multidentate ligand with a single metal ion. The mechanism of Ca^{2+} chelation by a ligand called BAPTA is shown in Figure 1.1.

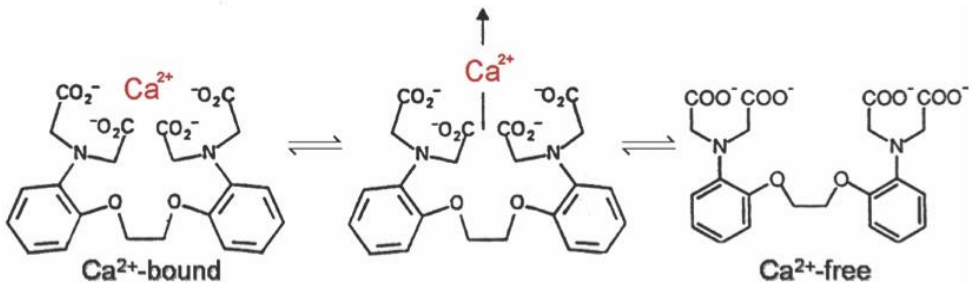


Figure 1.1 Mechanism of Ca^{2+} chelation by BAPTA. The presence of four carboxylic acid (usually written as –COOH) functional groups makes possible the binding of Ca^{2+} ions

Chelation of Ca^{2+} by a buffer B, to form a complex CaB, is described by the reaction

$$[Ca^{2+}]+[B]\underset{k_{off}^{B}}{\overset{k_{on}^{B}}{\rightleftharpoons}}[CaB] \qquad \textbf{(1.1)}$$

and the corresponding kinetic equation is

$$\frac{d[CaB]}{dt} = k_{on}^{B}[Ca^{2+}][B] - k_{off}^{B}[CB]$$

where square brackets are used to indicate concentration, k_{on}^{B} is the rate constant for Ca^{2+} binding to B and k_{off}^{B} is the rate constant for Ca^{2+} dissociation. At chemical equilibrium

$$\frac{d[CaB]}{dt} = 0$$

therefore

$$\frac{[Ca^{2+}][B]}{[CaB]} = \frac{k_{off}^{B}}{k_{on}^{B}} \equiv k_{d}^{B}$$

In the above equation, which represents an instance of the law of mass action under equilibrium conditions, k_{d}^{B} is the equilibrium or dissociation constant.

Ca^{2+}-selective fluorescent probes share a modular design consisting of a metal-binding site (or sensor) A, covalently coupled to a fluorophore B, therefore

$$[A] = [B]$$

Fluorescent probes are molecules whose spectral properties can altered in a suitable manner by the parameter to be measured ($[Ca^{2+}]$), like a change in fluorescence yield or a shift in the excitation or emission spectrum. (Mammano & Bortolozzi, 2010). In Figure 1.2 it is possible to see the typical sigmoid dependence of the fluorescence emission on $[Ca^{2+}]$.

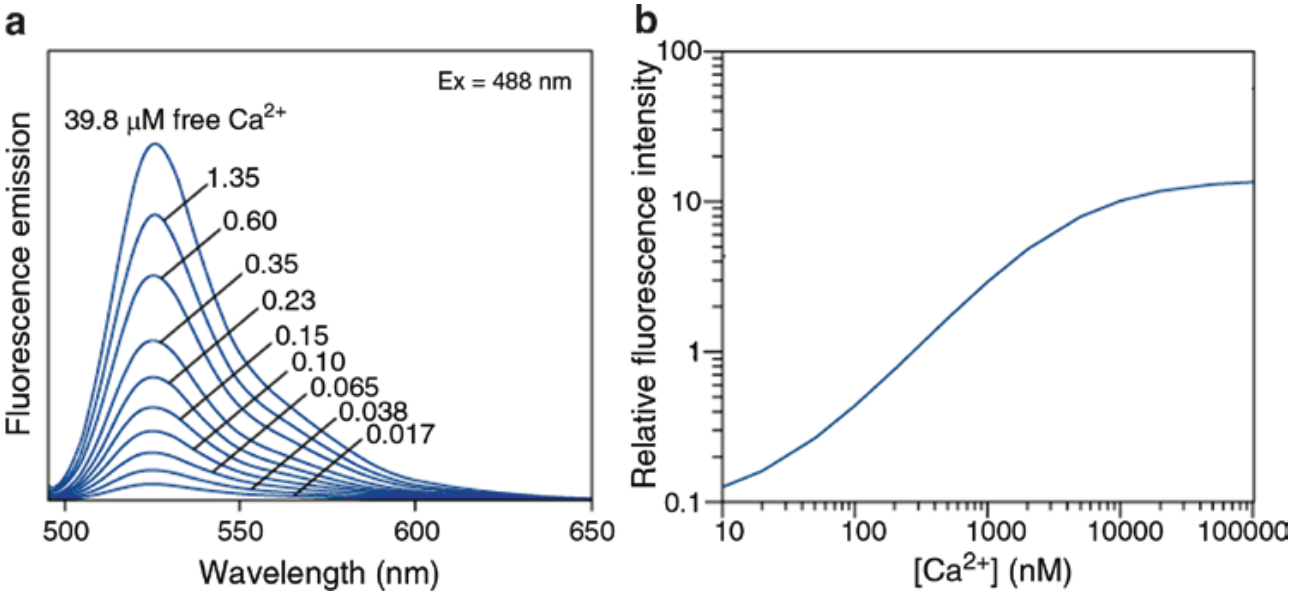


Figure 1.2 (a) Fluo-3 (a single wavelength fluorescence indicator) spectra, excited by the 488-nm line of the Argon laser, are shown for different values of the free Ca^{2+} concentration. (b) Relative fluorescence emission intensity, measured at the peak of each spectrum in (a), plotted against the corresponding $[Ca^{2+}]$.

1.2.1 Fluorescence

Suppose now that we have a system of fluorophores A at a total concentration c_T, which we excite with light of a given intensity and wavelength λ, we can summarize the excitation process as

$$A + photon \xrightarrow{k_A} A^*$$

Where k_A is the excitation rate constant (in units of s^{-1}) and A^* represents fluorophores in the excited state. The system relaxes either non-radiatively (nr), with a rate k_{nr}, or radiatively (r), emitting a photon of longer wavelength (i.e. reduced energy $h\nu'$) with a rate constant k_r (Figure 1.3).

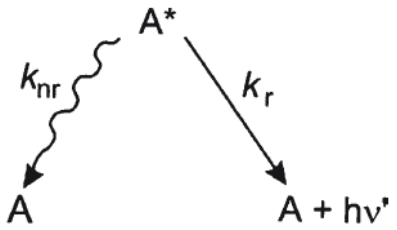


Figure 1.3: Radiative and nonradiative decay from the excited state

The overall relaxation rate constant k_M is given by

$$k_M = k_r + k_{nr} = \frac{1}{\tau_{ex}}$$

(in units of s^{-1}) where τ_{ex} is the excited state lifetime (typically a few ns). Under constant illumination, a steady state is rapidly reached such that

$$k_A \alpha \left(c_T - [A^*] \right) = k_M [A^*]$$

where the dimensionless parameter α represents the fraction of absorbed photons. Therefore, the equilibrium (steady state) concentration of excited state fluorophores $[A^*]_{eq}$ is given by

$$[A^*]_{eq} = \frac{\alpha c_T}{\alpha + k_M / k_A}$$

where, in general, $k_M \ll k_A$. The fluorescence emission intensity $f(t)$ is proportional to the number of photons emitted in the process of relaxation from the excited state

$$[A^*] \xrightarrow{k_r} A + photon$$

therefore, its steady state value is

$$f = k_r[A^*]_{eq} = \frac{\alpha k_r c_T}{\alpha + k_M / k_A}$$

where the result is expressed in mols of photons emitted per unit time and unit volume of dye solution.
For a given c_T, α is proportional to the product $\varepsilon(\lambda)\cdot l$, where $\varepsilon(\lambda)$ is the molar absorption coefficient (in units of $mol^{-1}\,m^{-1}$) and l is the length of the path traversed by the illuminating beam through the absorbing medium. The fluorescence quantum yield (sometimes termed quantum efficiency) is a gauge for measuring the efficiency of fluorescence emission relative to all of the possible pathways for relaxation and is generally expressed as

$$\eta_F = \frac{k_r}{k_M}$$

Therefore, we conclude that f depends on factors such as illumination intensity, molar concentration of fluorescent probes, fluorescence quantum yield, molar absorption coefficient, and path length. Let us then assume that the concentration of Ca^{2+}-selective fluorescent probes is kept low enough that the relationship between fluorescence emission intensity and concentration is indeed linear. In general, the concentration $[F]$ and $[CaF]$ of the Ca^{2+}-free (b) and Ca^{2+}-bound (b) forms differ with respect to quantum yield and absorption. Therefore, we write f as a linear combination

$$f = S_f[F] + S_b[CaF] \qquad \textbf{(1.2)}$$

where the proportionality constants S_b and S_f lump all (system-dependent) factors such as illumination intensity, η_F, $\varepsilon(\lambda)$ and l. We are interested in measuring $[Ca^{2+}]$ in a closed system (e.g. the cell cytoplasm).
Hence, we must also include the mass balance equation

$$c_T = [F] + [CaF] \qquad \textbf{(1.3)}$$

We then define

$$f_{\max} = S_b c_T \qquad \textbf{(1.4)}$$

as the fluorescence emission under Ca^{2+} saturation conditions and

$$f_{\min} = S_f c_T \qquad \textbf{(1.5)}$$

as the corresponding emission under Ca^{2+}-free conditions. Combining equations (1.5), (1.4), (1.3) and (1.2), we can write

$$\frac{f - f_{\min}}{f_{\max} - f} = \frac{[CaF]}{[F]} \qquad \textbf{(1.6)}$$

At chemical equilibrium $\frac{[Ca^{2+}][F]}{[CaF]} = \frac{k_{off,B}}{k_{on,B}} = k_{d,B}$, so

$$[Ca^{2+}] = k_{d,B}\frac{[CaF]}{[F]}$$

Therefore, we conclude that

$$[Ca^{2+}] = k_{d,B}\frac{f - f_{\min}}{f_{\max} - f} \qquad \textbf{(1.7)}$$

Equation (1.7) expresses a quantitative relationship between the physiologically relevant equilibrium $[Ca^{2+}]$ the dissociation constant $k_{d,B}$ and optically measurable quantities $f_{\min}$, $f_{\max}$ and f for single wavelength Ca^{2+}-selective probes. However, there are a number of caveats and problems with the practical use of (1.7). First, we note that the denominator vanishes as $f \to f_{\max}$. Consequently, even small fluctuations in the estimate of f (e.g. due to instrumental noise) may cause unacceptably large fluctuations in the estimation of $[Ca^{2+}]$. Furthermore, (1.7) is difficult to apply to imaging experiments where $f_{\max}$, $f_{\min}$ and f change rapidly over time due to photo-bleaching.

2 Simulation

To understand complex biological systems such as cells, tissues, or even the human body, it is not enough to identify and characterize the individual molecules in the system. It also is necessary to obtain a thorough understanding of the interaction between molecules and pathways.
Modeling approaches are essential for biologists, enabling them to analyze complex physiological processes. Computational models help investigators to systematically analyze systems perturbations, develop hypotheses to guide the design of new experimental tests, and ultimately assess the suitability of specific molecules as novel therapeutic targets.
Mathematical models allow researchers to investigate how complex regulatory processes are connected and how disruptions of these processes may contribute to the development of disease. Numerous mathematical methods have been developed to address different categories of biological processes, such as metabolic processes or signaling and regulatory pathways. (Fischer, 2008)
To study the dynamics of Ca^{2+}, imaging techniques are used to observe temporal and spatial dynamics of Ca^{2+} ions. Typically, these techniques combine Ca^{2+}-sensitive fluorescent dyes, patch clamp and optical microscopy together with various stimulation protocols. (Rispoli, 2001; Lelli, et al., 2003; Issa, 1994; Tucker, 1995)
Simulation and mathematical methods are necessary, both to calculate the expected time course of calcium concentration increase and their spatial extent in the presence of multiple buffers (Nowycky, 1993) and to extrapolate the cytosolic calcium concentration from fluorescent measurements.

For example, simulations in Figure 2.1 indicate that free Ca^{2+} concentration estimate, based on equilibrium of the reactants (1.7) is seriously in error: the $\Delta F/F0$ signal is a compressed and low pass filtered version of the real $[Ca2+]i$. Because of the speed of Ca^{2+}influxes, non-equilibrium conditions are fundamental.

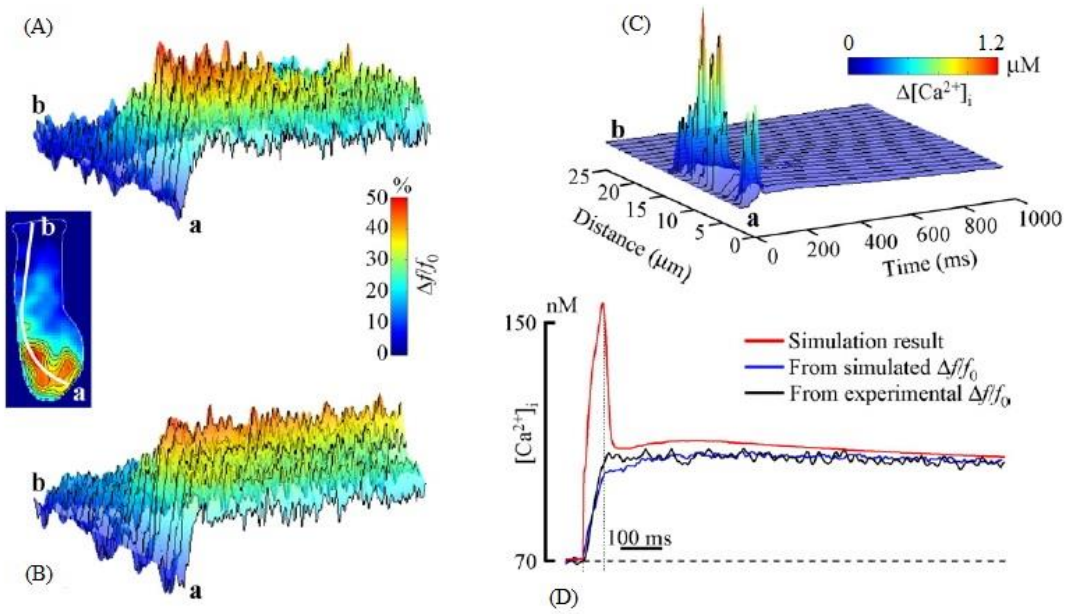


Figure 2.1 Fluorescence signals vs. cytosolic free Ca2+ concentration. (A) Pseudo-line-scan representation of $\Delta f / f_0$ ***signals obtained from the experiment; (B) Simulated*** $\Delta f / f_0$ ***signals for a model. (C)*** $\Delta[Ca^{2+}]_i$ ***changes corresponding to the simulation in (B). (D) Time course of the simulated*** $\Delta[Ca^{2+}]_i$***, integrated over the entire cell (red trace) and*** $\Delta[Ca^{2+}]_i$ ***values derived either from the simulated (blue trace) or from the experimental (black trace)*** $\Delta f / f_0$ ***whole cell signals, based on the law of mass action at equilibrium.***

2.1 SimulCell

SimulCell is a user-friendly simulation software from Bortolozzi lab. It allows the user to solve arbitrary systems of differential equations, typically reaction-diffusion equations, in a 1-dimensional and 3-dimensional geometry. Furthermore, it is possible to simulate the fluorescence signals of the dye, provided it is inserted as a variable in the equations.
In this thesis work, we added a new feature to SimulCell, and used it to perform simulations.

2.1.1 Define Equations

In SimulCell "Define equations" panel, the user has to define the equations, parameters and eventual input statements or input functions.
Equations are written in a user-friendly language, and later turned into code by a parser.

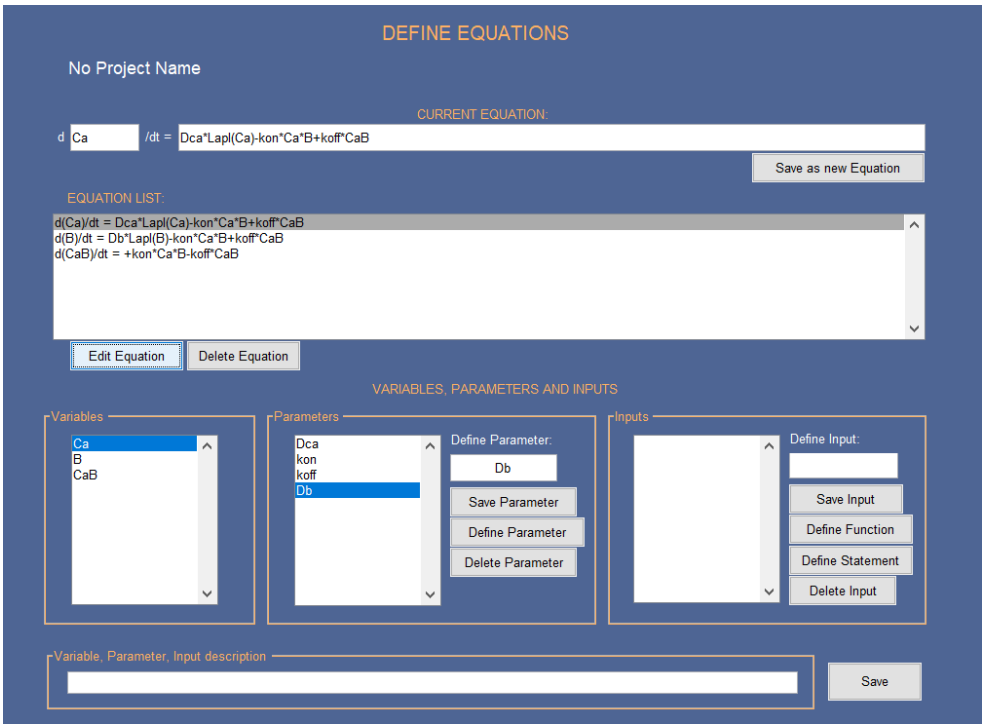


Figure 2.2 System of reaction-diffusion equations involving calcium (Ca), a buffer molecule B, and the buffer bounded with calcium (CaB).

2.1.2 Geometry

"Geometry 1D" panel allows the creation of a one-dimensional object. The user can choose the cell length and the number of voxel to discretize it. One or more voxels can be selected by specifying their indexes, and be set as a pattern. Pattern can be chosen according to simulation purposes.
For instance, it is possible to create a 100µm long cell and divide it into one hundred voxels. Useful patterns may be central voxel, border voxels, internal voxels, all the voxels…

In "Geometry 3D" panel, the user can choose a three-dimensional shape between box, ellipsoid, cylinder and cone and set objects physical dimensions. He can set the number of points used to shape the object. Given two or more objects, it is possible to make set operations, like union, intersection and difference.
The software allows to discretize the object using a regular grid of cubic voxels. Cubic voxels are the domains in which equations are solved.

Part of the work of this thesis was the implementation of the discretization with a tetrahedral mesh. (See paragraph 2.2)
In three-dimensional case, it is possible to choose between a number of given patterns: volume, external surface, center, planes parallel to coordinate-axis, lines parallel to axis and their extremes.

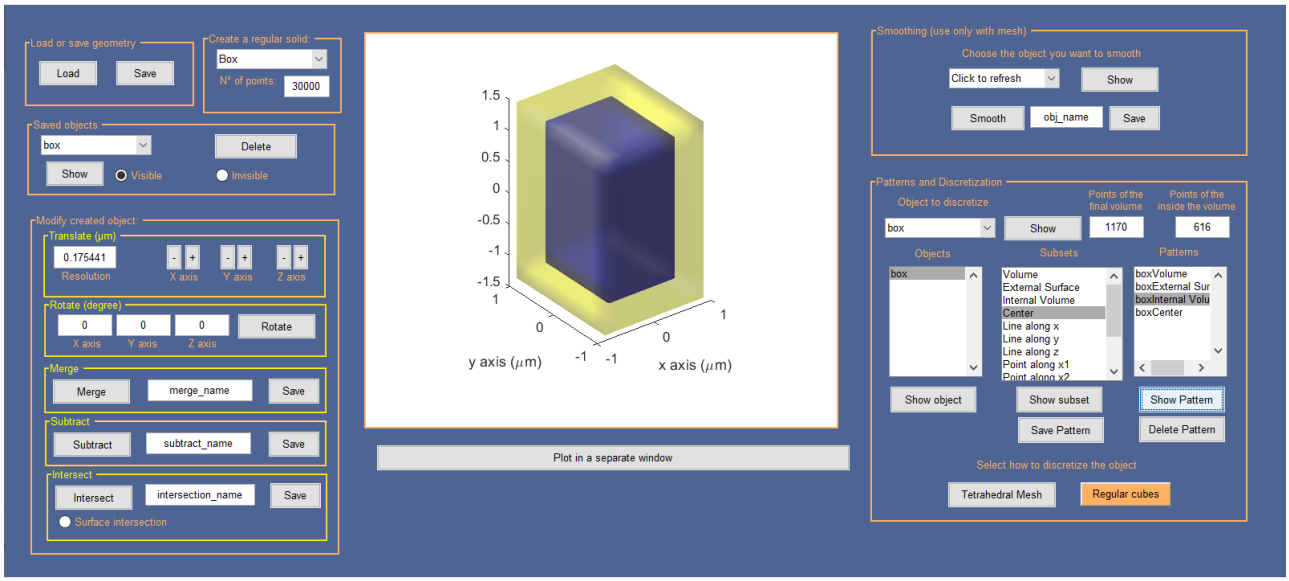


Figure 2.3 Box created with Geometry3D. Volume, external surface, internal volume and center are the selected patterns. In the picture, we can see the whole object (yellow) and the selected pattern 'box Internal Volume' (blue).

2.1.3 Initial Conditions

"Initial condition" panel allows to set the value of every variable at t = 0. A variable can have a different initial value in each pattern defined in Geometry section. Patterns must not intersect, to avoid multiple definition of a variable in a single voxel. All the voxels must have a starting value for the variables. To make it possible with the available patterns, "Rest of voxels" pattern is automatically defined in the list of the patterns.
To simulate the diffusion terms of the equations, we have to define voxels permeability. For each pattern, the permeability with other patterns selected in "Geometry" must be specified, and also the permeability with the external space.
The initial value of every parameter must be set using patterns as well, and function or statements must be referred to voxels belonging to a pattern.

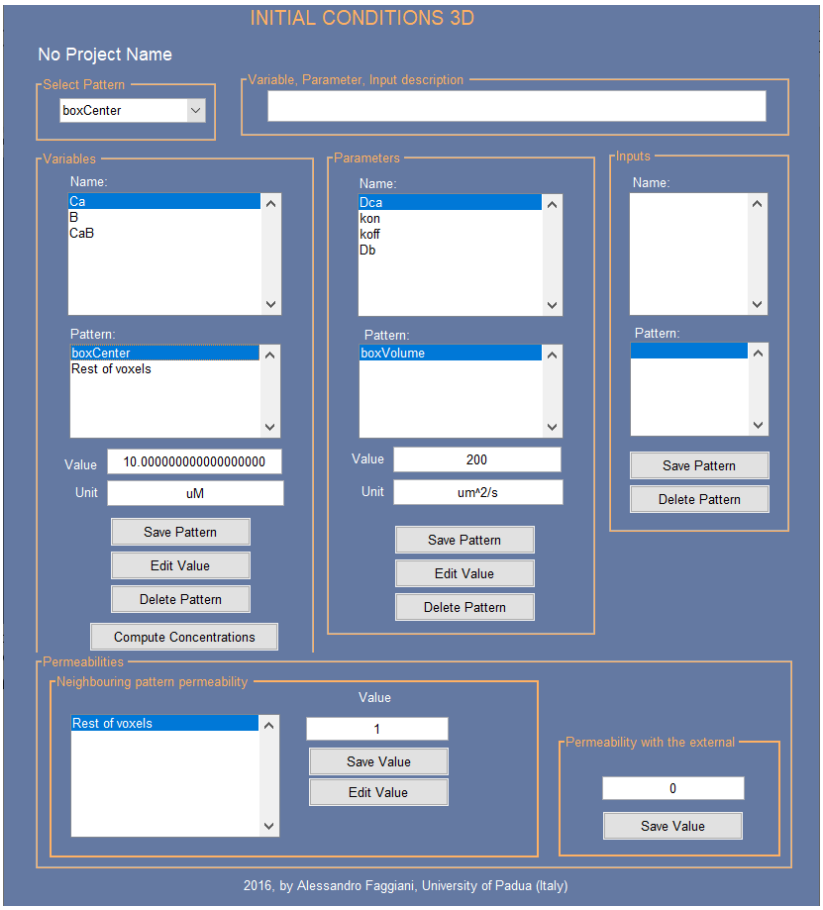


Figure 2.4 Initial Condition panel: starting value of 'Ca' variable is defined in the pattern "boxCenter" and in the "Rest of voxels". Value for Dca coefficient is defined in the whole box volume.

2.1.4 Compile and run the model

After defining the model, the user can choose between several ordinary differential equation solvers and set some parameters like error tolerance, size of time step, maximum number of time steps. Compilation consists in writing expressly all the equations in every voxel of the mesh. A parser recognises equation terms written by the user in "Equation" section, and the equations are interpreted for the solver. It is possible to set the final time (in seconds) for the simulation, and the intermediate timesteps to be saved.

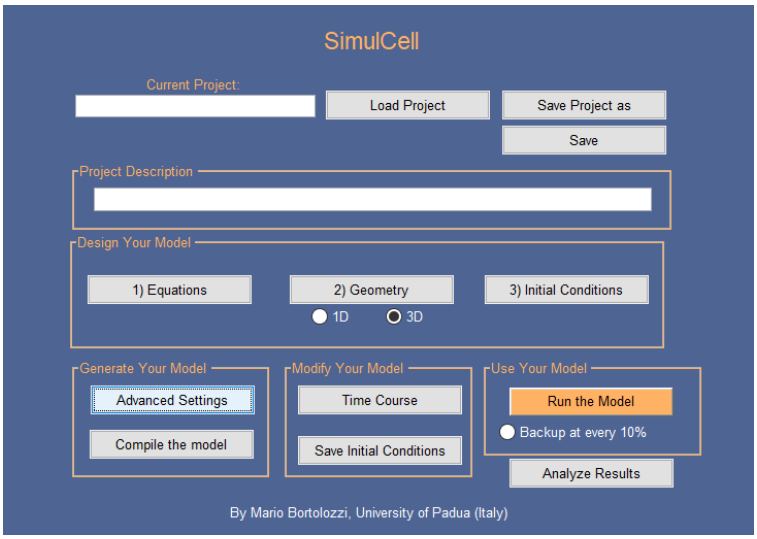


Figure 2.5 Main interface of SimulCell. After designing the model, the user can compile and run it, having set several solver parameters.

2.1.5 Analyse Results

"Analyze Results" panel is provided in order to visualize the solution of the equation system. The user can select a pattern and visualize the time course of a variable in the voxels belonging to the pattern. It is possible to make time and spatial averages of variables. There are also some features that allow to calculate the fluorescence emitted by a dye buffer, starting from the solution of reaction-equations.

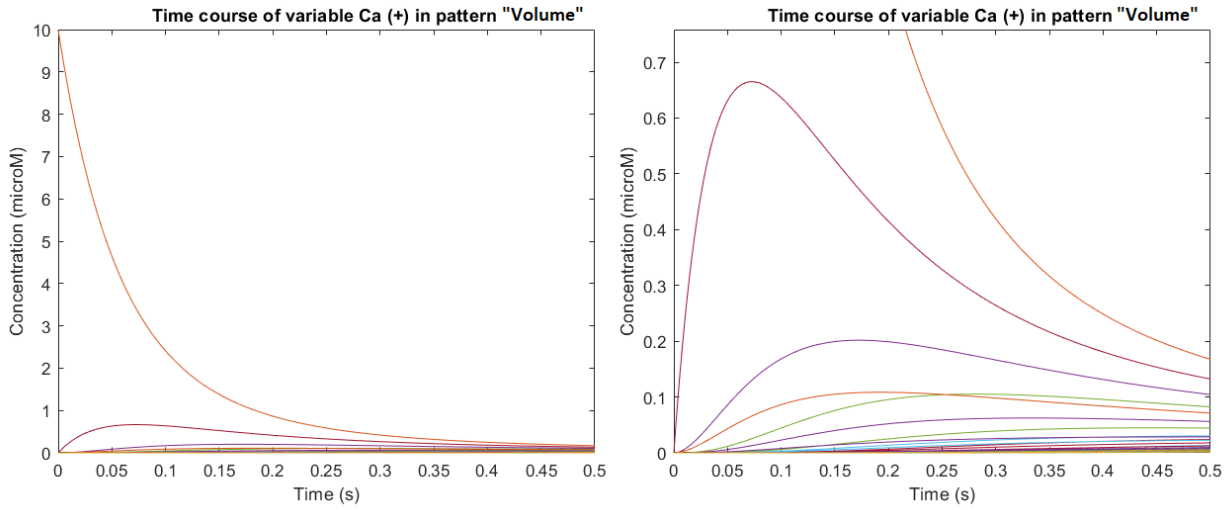


Figure 2.6 Evolution of concentration of a diffusive substance from a central point of source in a sphere discretized with regular cubes. Every line represents one voxel. The voxel with the higher concentration is the central one. Concentration in the adjacent voxels increases from zero and then diffuses to peripherical voxels. The picture on the right is a zoom of the same plot.

In this thesis work, we added to SimulCell the possibility of discretizing any object using a mesh composed by tetrahedral voxels instead of cubic voxels. All the software sections were modified in order to support the new feature.

2.2 Mesh

A mesh is a network formed of cells and points. It can have almost any shape and any size.
Meshes can be used to discretize domains for solving Partial Differential Equations. Each cell of the mesh represents an individual solution of the equation. These solutions, combined for the whole network, result in a solution for the entire mesh.
Solving the entire object without dividing it into smaller elements can be impossible because of its geometrical complexity. Holes, corners and angles can make it extremely difficult for solvers to obtain a solution. Small cells are comparably easy to solve and therefore the preferred strategy.

Mesh types can be classified as structured or unstructured.
Structured meshes, commonly called grids, have a structure allowing for easy identification of neighbouring cells. In fact, structured meshes are applied over analytical coordinates systems (rectangular, elliptical, spherical, etc.) and form a regular grid. Each mesh element can be directly mapped into a three-dimensional array, making computations easier. This is the case of objects cubic discretization in SimulCell.
Unstructured meshes are more general and can arbitrarily approximate any geometry shape. They require special data structures, such as an adjacency matrix and the node coordinates list. The unstructured mesh node/cell numbering can be arbitrary and sparse since it does not require any analytical form of adjacency query.

Complex geometries that would be impractical to generate a structured mesh within, can be discretized using unstructured meshing techniques. Mesh can be thickened in border regions whose boundary has a high curvature (Figure 2.7) (SimWiki)
The tetrahedral meshing implemented in SimulCell is an example of unstructured mesh generation.
Because of the extreme variety of elements shape, it may be necessary to determine elements quality. Higher-quality (better-shaped) elements have better numerical properties. A mesh is considered to have higher quality if a more accurate solution is calculated more quickly. Increasing the mesh elements number always increases the accuracy but also increases computational cost.
One of these parameters is smoothness. It is the maximum volume ratio between two neighbouring elements. Sudden jumps in the size of the cell may cause erroneous results.
Another parameter is aspect ratio. It is the ratio of longest to the shortest side in a cell. Adjacent cell sizes should not vary by more than 20%. (Wikipedia, 2019)

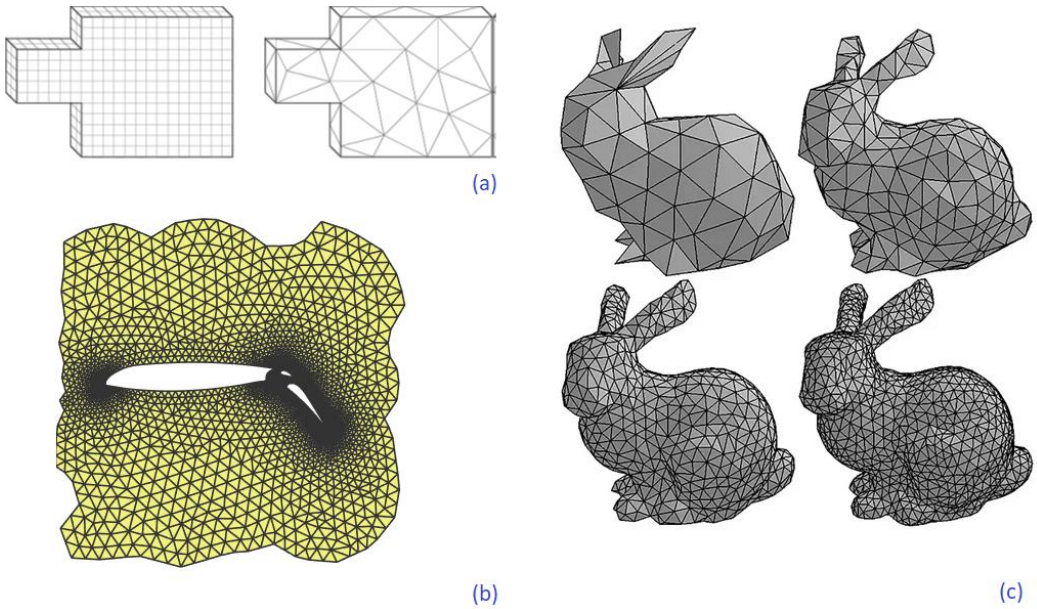


Figure 2.7 Structured and unstructured meshing of a solid (a). Two-dimensional mesh local refinement (b). Tetrahedral meshing with various elements size (c).

Another mesh feature (very important for PDE solving), is the possibility of thickening mesh size in regions requiring a more accurate solution. For example, in cells like sensory cells, Ca^{2+} concentration increases abruptly following opening of voltage-dependent channels (Lewis, 1983). These channels show a cooperative behaviour and cluster, creating "hotspots", microdomains of elevated $[Ca^{2+}]_i$ (Figure 2.8) (Bortolozzi, Lelli, & Mammano, 2008). A more precise meshing of the space near the hotspot would allow to give a more precise description of concentration gradient.

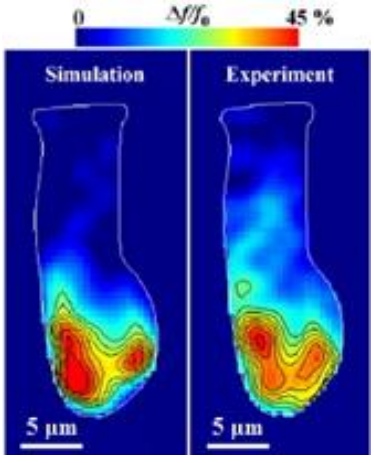


Figure 2.8 Simulated and experimental fluorescence for Calcium hotspots in a hair cell

3 Diffusion

Diffusion is a phenomenon that can be described by Fick's laws, partial differential equations which describe the temporal and spatial change in concentration of the diffusing solute. Fick's laws can be used to study diffusion of molecules inside the cell cytoplasm and their transport through biological membranes.
Fick's first law relates to the diffusive flux of a substance inside a fluid, to its concentration. It postulates that the flux goes from regions of high concentration to regions of low concentration, with a magnitude that is proportional to the concentration gradient.

$$\bar{j} = -D \cdot \bar{\nabla} C \tag{3.1}$$

Inserting the ((3.1) in the mass conservation law ((3.2)

$$\frac{dC}{dt} = -\bar{\nabla} \cdot \bar{j} \tag{3.2}$$

we obtain Fick's second law (3.3). It is a non-linear parabolical partial differential equation.

$$\frac{dC}{dt} = D \cdot \bar{\nabla}^2 \bar{j} \tag{3.3}$$

If we consider diffusion of N identical particles from a point of source in an infinite isotropic and homogeneous medium, the initial condition for Fick's equation is equivalent to:
$C(t=0,x) = M\delta(x)$, where M is the number of moles of the diffusing substance.
The solution of Fick's second law is

$$C(x,y,z,t) = \frac{M}{(4\pi Dt)^{3/2}} e^{-\frac{x^2+y^2+z^2}{4Dt}} \tag{3.4}$$

Imposing M=1 in ((3.4), we obtain a probability density function, whose mean square displacement from the initial position is $\overline{x^2} = 2Dt$, and variance: $\sigma^2 = 2Dt$.

3.1 Diffusion equation discretization

In order to simulate diffusion, we need to discretize the Fick's second law:
The Laplacian operator $\bar{\nabla}^2 C(x)$ in a one-dimensional grid can be discretized in the following way.
Grid points x_i are spaced by Δx and their concentrations are indicated as $C_i = C(x_i)$.

Using the third order Taylor series of C we can write

$$C_{i+1} = C_i + \frac{\delta C}{\delta x}\Big|_i \Delta x + \frac{1}{2}\frac{\partial^2 C}{\partial x^2}\Big|_i \Delta x^2 + \frac{1}{6}\frac{\partial^3 C}{\partial x^3}\Big|_i \Delta x^3 + O(\Delta x^4)$$

$$C_{i-1} = C_i - \frac{\delta C}{\delta x}\Big|_i \Delta x + \frac{1}{2}\frac{\partial^2 C}{\partial x^2}\Big|_i \Delta x^2 - \frac{1}{6}\frac{\partial^3 C}{\partial x^3}\Big|_i \Delta x^3 + O(\Delta x^4)$$

By adding them and inverting the final equation, we obtain:

$$\frac{\partial^2 C}{\partial x^2}\Big|_i = \frac{C_{i+1} + C_{i-1} - 2C_i}{\Delta x^2} + O(\Delta x^2) \qquad \textbf{(3.5)}$$

To obtain the Laplacian discretization in a three-dimensional grid, we can repeat for y and z coordinates and sum the expressions found for the second derivatives:

$$\overline{\nabla}^2 C\,|_{ijk} = \frac{C_{i+1\,jk} + C_{i-1\,jk} - 2C_{ijk}}{\Delta x^2} + \frac{C_{i+1\,jk} + C_{ij-1k} - 2C_{ijk}}{\Delta x^2} + \frac{C_{ijk+1} + C_{ijk-1} - 2C_{ijk}}{\Delta z^2} \qquad \textbf{(3.6)}$$

The temporal at the n-th step of the simulation can be discretized by the forward derivative:

$$\frac{\delta C}{\delta t} = \frac{C_i^{n+1} - C_i^n}{\Delta t}$$

Where $C_i^n = C(x_i, t_n)$, $t_n = t_0 + n\Delta t$ and Δt is the time step.

The diffusion equation can be re-written by substituting the operators with their discretized expressions:

$$\frac{\delta C}{\delta t} = D\overline{\nabla}^2 C$$

In one dimension:

$$\frac{C_i^{n+1} - C_i^n}{\Delta t} = \frac{D}{\Delta x^2}\left[C_{i+1} + C_{i-1} - 2C_i\right] \qquad \textbf{(3.7)}$$

In three dimensions:

$$\frac{C_{ijk}^{n+1} - C_{ijk}^n}{\Delta t} = \frac{D}{\Delta x^2}\left[C_{i+1\,jk} + C_{i-1\,jk} - 2C_{ijk}\right] + \frac{D}{\Delta y^2}\left[C_{ij+1k} + C_{ij-1k} - 2C_{ijk}\right] + \frac{D}{\Delta z^2}\left[C_{ijk+1} + C_{ijk-1} - 2C_{ijk}\right] =$$

(With $\Delta x = \Delta y = \Delta z =: \Delta r$)

$$= \frac{D}{\Delta r^2}\left[C_{i+1\,jk} + C_{i-1\,jk} + C_{ij+1k} + C_{ij-1k} + C_{ijk+1} + C_{ijk-1} - 6C_{ijk}\right] \qquad \textbf{(3.8)}$$

This is called FTCS algorithm (forward time centred space derivates).

3.2 Random walk

It is possible to get to the same results starting from the concept of Brownian motion and random walk.

Brownian motion is the stochastic motion of particles induced by random collisions with water molecules. (Metcalfea, Speetjensb, Lesterc, & Clercxd, 2012)

At equilibrium, the average velocity of the particles over a long period of time is zero, which is a consequence of the particles moving in all directions with equal probability. The key mathematical concept to model Brownian motion is the random walk model. Einstein used the random walk model to relate Brownian motion to the self-diffusion coefficient in the limit of sufficiently long time.

Let's consider a one-dimensional random walk problem. A Brownian particle starts at $x = 0$ and each step is of the same size Δx and has a probability p to be on the right and a probability q to be on the left. (Rudin & Choi, 2013)

After N steps, the probability to have done *m* steps on the right and *N-m* steps on the left is the binomial distribution:

$$P(m) = \binom{N}{m} p^n q^{N-m}$$

The mean of binomial probability density function is $\overline{m} = Np$,

and its variance is $\text{var}(m) = \overline{m^2} - \overline{m}^2 = Npq$

The relation that links *x* and *m* is

$$x=(2m-N)\Delta x \qquad \textbf{(3.9)}$$

Then, considering $p = q = 0.5$, we have

$$\overline{x} = 0 \qquad \textbf{(3.10)}$$

and $$\text{var}(x) = \overline{x^2} - \overline{x}^2 = \overline{x^2}$$

Inserting (3.9) in the previous expression, and using $p = q = 0.5$ we obtain

$$\text{var}(x) = \Delta x^2 (4m^2 + N^2 - 4N\overline{m}) = ... = \Delta x^2 N = \Delta x^2 \frac{t}{\Delta t}$$

Defining $D = \frac{\Delta x^2}{2\Delta t}$, the value we find for σ is

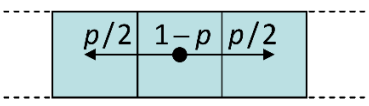
$$\sigma = \sqrt{\text{var}(x)} = \sqrt{2Dt} \qquad \textbf{(3.11)}$$

According to central limit theorem, as N approaches infinite, the binomial distribution tends to a gaussian with mean (3.10) and variance as in (3.11). Noticing that (3.10) and (3.11) are equivalent to mean and variance of the gaussian function (3.4).

This shows the equivalence between particle motion due to random walk and diffusion process.

3.3 Diffusion and random walk comparison

For a random walk step, forcing the particle to move left or right at each time step (Einstein-Smoluchowski algorithm) is less accurate than including the third possibility that the particle remains at the same position.



$\frac{p}{2}$ is the probability of the particle step to be towards the right, and the probability to be towards the left. $1-p$ is the probability to remain in the central voxel.

In the limit of a very big number of particles, *p*/2 represents the percentage of mass moving to the left or to the right after a time step.

The concentration variation in voxels $(l-1)th$ and $(l+1)th$ due to diffusion from the *l*-th voxel will be

$$dC_{l-1} = dC_{l+1} = C_l \cdot \frac{p}{2}$$

The concentration variation of the l-th voxel can be obtained by considering both the lost mass and the positive contribution from the adjacent voxels: $dC_l = C_{l-1}\frac{p}{2} + C_{l+1}\frac{p}{2}$.

The updated concentration of the l-th voxel after a time step Δt will be then:

$$C_l^{n+1} = C_l^n - pC_l^n + \frac{p}{2}C_{l+1}^n + \frac{p}{2}C_{l-1}^n$$

Dividing both members by Δt, the equation can be rewritten as:

$$\frac{C_l^{n+1} - C_l^n}{\Delta t} = \frac{-pC_l^n + \frac{p}{2}C_{l-1}^n + \frac{p}{2}C_{l+1}^n}{\Delta t},$$

This equation gives Fick's second law discretization if $p = \frac{2D\Delta t}{\Delta x^2}$.

The same result can be obtained in three dimensions considering $p/6$ as the percentage of mass moving by random walk to each one of the six orthogonal neighbours of point *ijk*. The temporal derivative can be written as:

$$\frac{C_{ijk}^{n+1} - C_{ijk}^n}{\Delta t} = \frac{-pC_{ijk}^n + \frac{p}{6}C_{i+1jk}^n + \frac{p}{6}C_{i-1jk}^n + ... + \frac{p}{6}C_{ijk-1}^n}{\Delta t} = \frac{\frac{p}{6}\left[-6C_{ijk}^n + C_{i+1jk}^n + C_{i-1jk}^n + ... + C_{ijk-1}^n\right]}{\Delta t}$$

Comparing it with Fick's temporal derivative:

$$\frac{\frac{p}{6}\left[-6C_l^n + C_{l1}^n + C_{l2}^n + ... + C_{lK}^n\right]}{\Delta t} = \frac{\frac{D\Delta t}{\Delta r^2}\left[-6C_{ijk} + C_{i+1jk} + C_{i-1jk} + C_{ij+1k} + C_{ij-1k} + C_{ijk+1} + C_{ijk-1}\right]}{\Delta t} \qquad \textbf{(3.12)}$$

This equation gives Fick's second law (3.8) if $p = \frac{6D\Delta t}{\Delta x^2}$ and $\Delta x = \Delta y = \Delta z$.

The diffusion process at distance r from a point source is described by the Gaussian function

$$C(\vec{r}, t) = \frac{M}{(4\pi Dt)^{3/2}} e^{-\frac{r^2}{4Dt}}$$

If C_{ijk}^0 is the concentration in the point source *ijk* at t=0, after a time step, the concentration of a point at distance r from the point source can be written, in first approximation, as pC_{ijk}^0, where $p = \frac{6D\Delta t}{r^2}$ due to the spherical symmetry of the concentration profile, which is equivalent to say that the x-axis can be oriented along $\vec{r}$, so that $\Delta x = r$.

If we consider N neighbouring nodes connected to a given node of the grid, the simplest criterion to determine the overall contribution from the N nodes is to average their single contributions.

3.4 Diffusion algorithm validation

3.4.1 2D Mesh generation and simulation

The first step in our work was validating the diffusion algorithm in 2 dimensions, by comparing it with the known analytical solution for diffusion equation, with diffusion starting from a central point of source.
The triangular mesh was built starting from a regular grid of points, using MATLAB DelaunayTriangulation function. This function, starting from the N points coordinates, generates a structure containing a Nx3 array, whose rows are the indexes of the three vertices of each triangle. (A triangulation is a meshing process which uses only given points as elements vertices).
To simulate free diffusion from a point source, it is necessary to calculate centre and area of each voxel of the mesh. The centre of a voxel is defined as the centre of mass of the mesh element, supposing a uniform density of the voxel.
Using the "neighbors" function, the indexes of the three triangles adjacent to each voxel are identified. The indexes are referred to the triangles represented by their vertices in the triangulation rows. Some triangles have less than three neighbours because they are located at the borders of the discretized region. In this case the missing indexes of the neighbours are replaced with "NaN".
Also, the distances between the centre of each triangle and its neighbours are calculated and saved in a Nx3 array.
Lastly, the voxel whose centre is the nearest to the geometrical centre of the meshed object was identified. This is required to simulate diffusion starting from a central point of source.

As a first test, vertices coordinates generated equilateral triangles, in order to work with the most regular mesh.

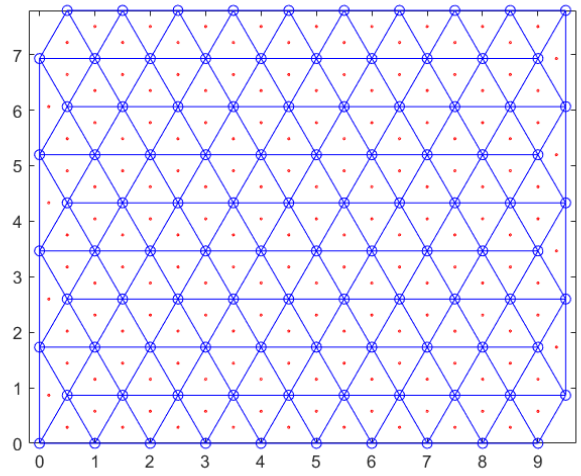


Figure 3.1 Equilateral triangular mesh. Vertices are connected by lines, whereas centers are indicated as small dots.

Diffusion was simulated using the discretization of Laplacian with three neighbours:

$$C_v^{n+1} = C_v^n \left[1 - \frac{p_{v1}}{3} - \frac{p_{v2}}{3} - \frac{p_{v3}}{3}\right] + \frac{p_{v1}}{3} C_{v1} + \frac{p_{v2}}{3} C_{v2} + \frac{p_{v3}}{3} C_{v3}$$

p_{vi} must represent probabilities, ergo simulation timestep Δt must be chosen little enough to have:

$$4\frac{D\Delta t}{d_{vi}^2} < 1 \qquad \text{for every } vi \qquad \textbf{(3.13)}$$

As initial condition, concentration is set to zero in every voxel except for the central one.
For the boundary absorbing boundary conditions were used, imposing $C = 0$ in the boundary voxels.

Simulated concentration was compared with theorical prediction for diffusion from a point of source in an infinite space:

$$C(d,t) = \frac{M}{4\pi Dt}\exp\left(-\frac{d^2}{4Dt}\right)$$

Where d is the distance with the centre of the central voxel and M is the number of moles.

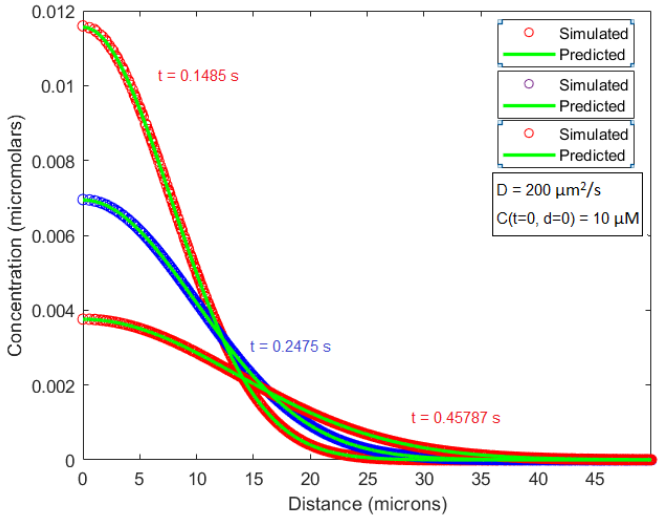


Figure 3.2 Simulation in equilateral triangular mesh

We can see the agreement between simulated and predicted concentration.

The algorithm was tested with a "deformed" grid: the coordinates of the points are varied by adding a random number from a uniform distribution multiplied by a deformation factor. When the mesh becomes very irregular, there is a deviation from the predicted curve, as shown in Figure 3.3 and Figure 3.4. Anyway, a mesh similar to the one in Figure 3.4 is extremely low quality. It is possible to avoid dealing with it by using an appropriate meshing software.

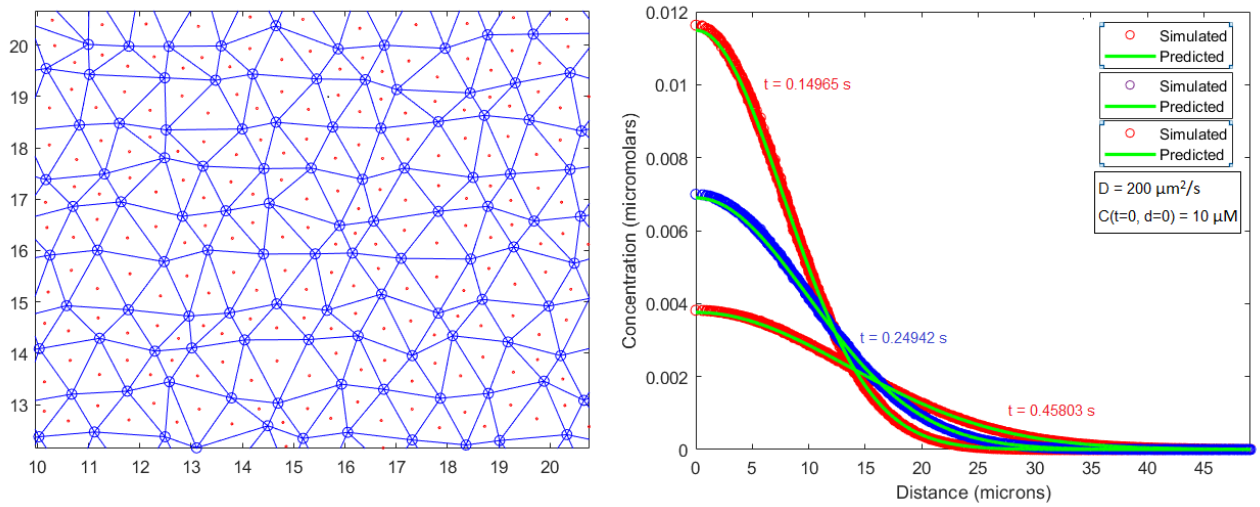


Figure 3.3 Left: Zoom of a Triangular mesh with deformation factor 0.5. Right: Simulation in triangular mesh with deformation factor 0.5

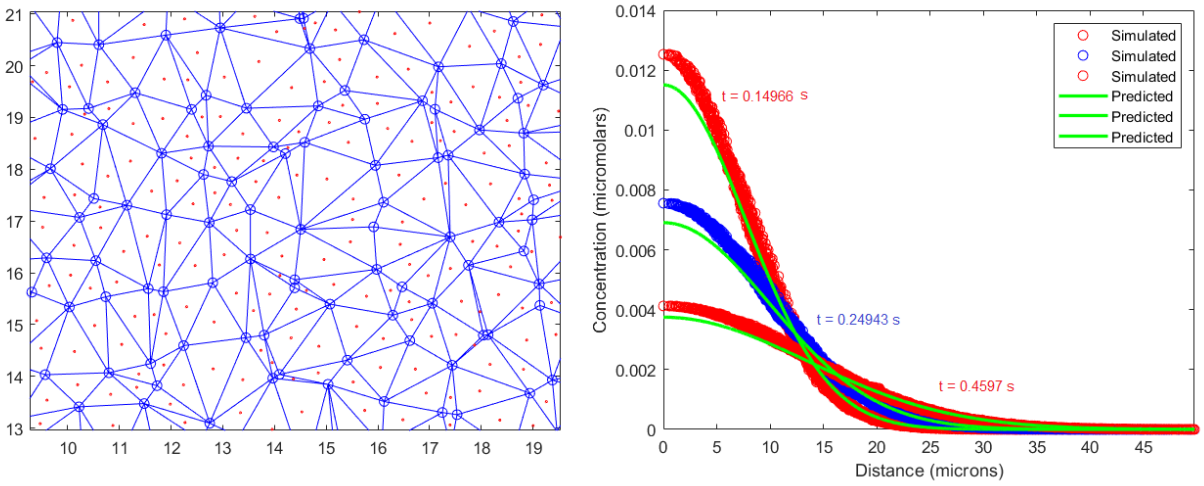


Figure 3.4 Left: 5 Triangular mesh with deformation factor 1. Right: Simulation in triangular mesh with deformation factor 1

3.4.2 3D Mesh generation and simulation

Similarly, the mesh was generated in a three-dimensional space. A three-dimensional grid of points was created, and their coordinates were varied using a deformation factor.
It was not possible to create a mesh composed by regular tetrahedra, because regular tetrahedra don't fill the space. (Senechal, 1981)
Mesh were created with DelaunayTriangulation function by deforming a regular grid of points.

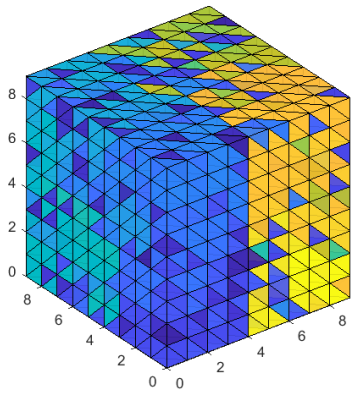


Figure 3.5 Grid of points triangulated with a tetrahedral mesh. This grid was deformed with a factor 0.5, re-meshed, and the performed simulation is in Figure 3.6 Simulation in a grid of points (deformation factor: 0.5) discretized with tetrahedral mesh generated with Matlab.

The algorithm for bulk voxels is the following:

$$C_v^{n+1} = C_v^n \left[1 - \frac{p_{v1}}{4} - \frac{p_{v2}}{4} - \frac{p_{v3}}{4} - \frac{p_{v4}}{4} \right] + \frac{p_{v1}}{4} C_{v1} + \frac{p_{v2}}{4} C_{v2} + \frac{p_{v3}}{4} C_{v3} + \frac{p_{v4}}{4} C_{v4}$$

The simulated concentration is compared with the theorical diffusive gaussian:

$$C(d,t) = \frac{M}{(4\pi Dt)^{3/2}} \exp\left(-\frac{d^2}{4Dt} \right)$$

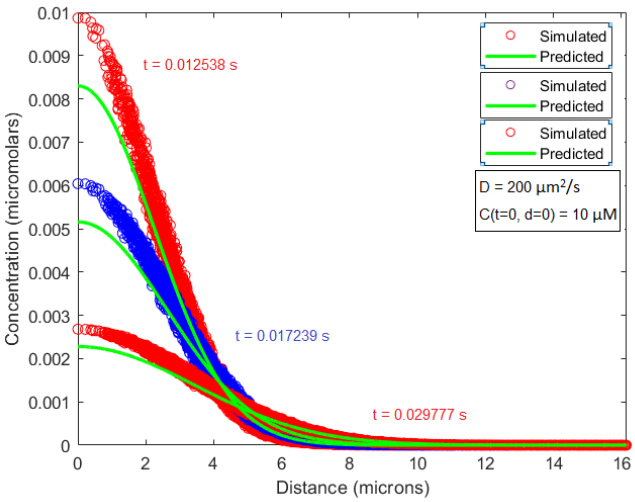


Figure 3.6 Simulation in a grid of points (deformation factor: 0.5) discretized with tetrahedral mesh generated with Matlab

The lack of agreement can be attributed to the lack of mesh quality. To solve this problem, we created the mesh with Jigsaw, a free Matlab plugin. (Engwirda, 2014)

Jigsaw accepts as input the triangulation of the surface of the object to be discretized, and builds the three-dimensional mesh approximating the surface with the voxels. Surface triangulation is obtained from the coordinates of grid points using Matlab "boundary" function.

It generates elements of high shape-quality, provides a good geometrical and topological approximation to the underlying surface, and satisfies a set of user-specified sizing constraints, like mesh maximum size and quality constraints, such as smoothness. To build the meshes, it uses both a Delaunay-refinement algorithm, and a frontal scheme. The first one starts from a coarse mesh and iterates a process of correction until the mesh satisfies all the constraints. The second one proceeds towards the internal, "filling" the volume with new elements (Engwirda, 2014).

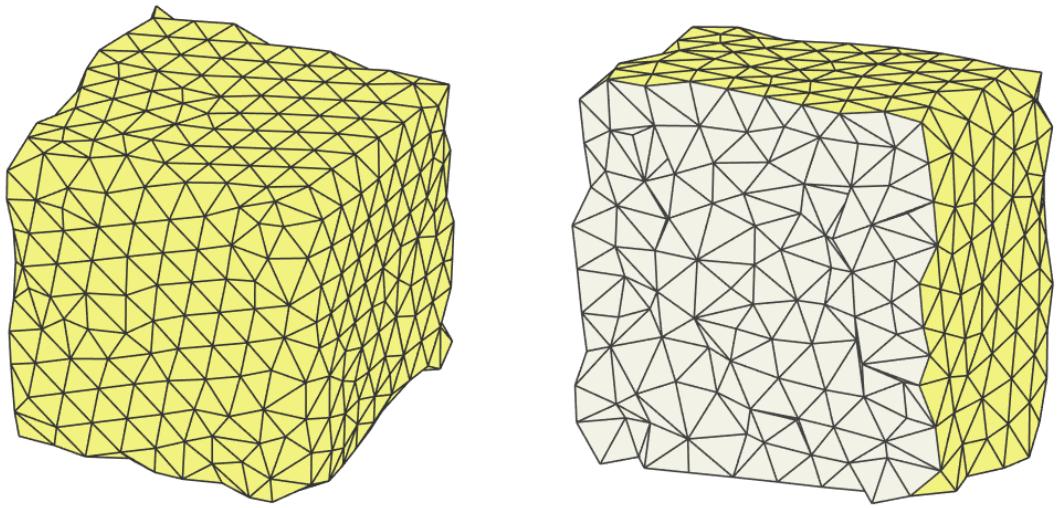


Figure 3.7 Example of deformed grid discretization with Jigsaw and a section of the object

Concentration in Figure 3.8 Simulation in a grid of points (deformation factor 0.5) discretized with tetrahedral mesh generated with was simulated using a Jigsaw mesh, maintaining the same parameters set for the simulation in Figure1. It is possible to see a better agreement with theorical prediction and a minor dispersion of concentration values.

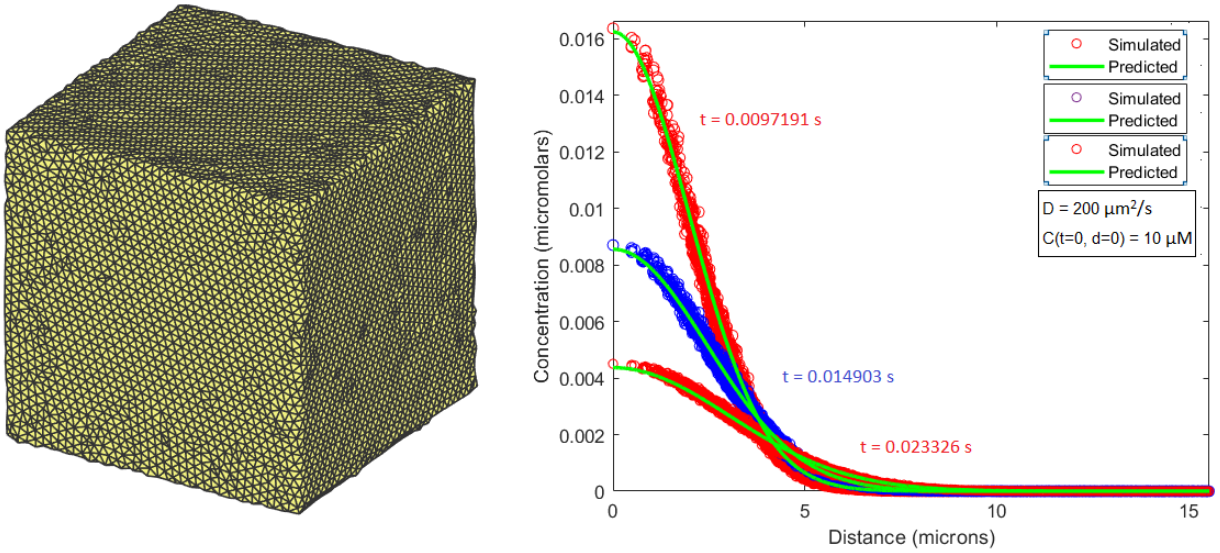


Figure 3.8 Simulation in a grid of points (deformation factor 0.5) discretized with tetrahedral mesh generated with Jigsaw

The simulation was also performed in a spherical region, in order to get the most regular mesh pattern.
The sphere is created using "meshgrid" function. The image of the cartesian equation for the sphere $V = x^2 + y^2 + z^2$ is defined in the points of the meshgrid. Then with "isosurface" function, the surface triangulation is created as a given value of V. The triangulation is the input for Jigsaw.

The comparison shows the agreement with the theorical prediction.

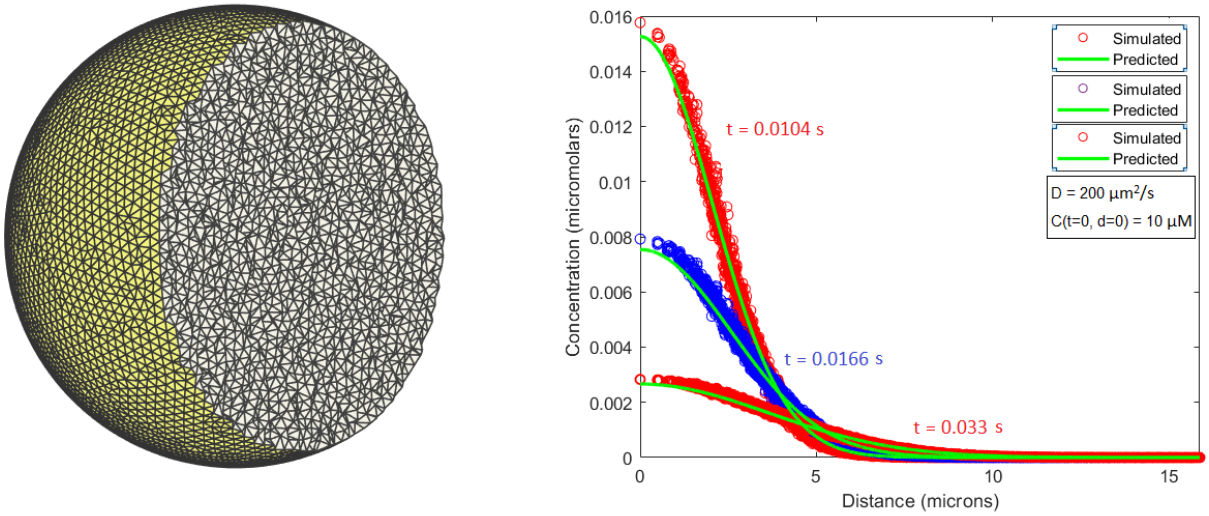


Figure 3.9 Section of a spherical tetrahedral mesh generated with Jigsaw and simulation performed in such domain.

To test diffusion speed, we compared sigma parameter for theorical and simulated distributions in Figure 3.8 and Figure 3.9.

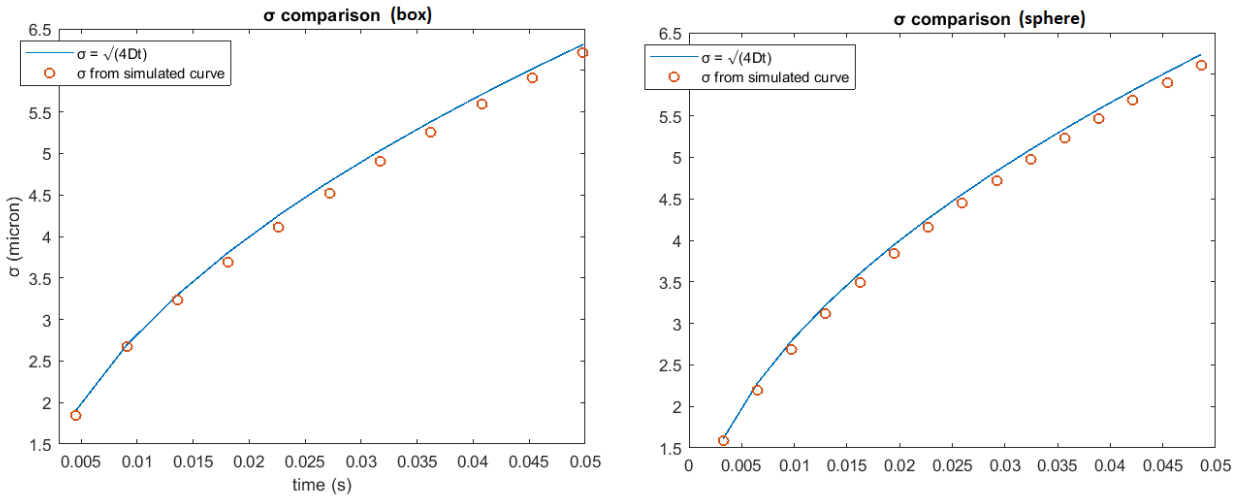


Figure 3.10 Comparison between theorical and simulated sigma for the box (left) and for the sphere (right)

Simulated sigma is slightly smaller than theorical sigma. This was also visible from concentration distribution plot, as simulated curves are usually slightly more narrow and higher.
From Figure 3.10 Comparison between theorical and simulated sigma for the box (left) and for the sphere (right)we can say that simulated diffusion in the sphere is slightly slower than theorical diffusion.
Anyway, the relative error between simulated and predicted concentration is not greater than 5%.

3.4.3 Boundary conditions and permeability

The previous comparisons are valuable until border effects are negligible. Once considered the interaction with borders, the two simplest situations are completely absorbing and reflecting boundary conditions. The first condition is obtained by setting the concentration in the border as zero.
The second condition is realized modifying the diffusion algorithm:

$$C_v^{n+1} = C_v^n \left[1 - \frac{p_{v1}}{4} - \frac{p_{v2}}{4} - \frac{p_{v3}}{4} - \frac{p_{v4}}{4} \right] + \frac{p_{v1}}{4} C_{v1} + \frac{p_{v2}}{4} C_{v2} + \frac{p_{v3}}{4} C_{v3} + \frac{p_{v4}}{4} C_{v4}$$

Let's assume that voxel *v* lacks neighbour *v1*: p_{v1} must be equal to zero. It means that the probability of diffusion through the border face of the voxel towards the external one is equal to zero (p_{v1}), and also the probability of diffusion from the outside to the inside (p_{v1}). This condition expresses the impermeability of tetrahedra border faces.
The concept of permeability of tetrahedra faces can be generalized to a value between zero (impermeability) and one (total permeability).

3.4.4 Infinite cylinder

At last, Diffusion interface allows to test diffusion in a cylinder.
A cylinder is created, as for the sphere, using "meshgrid" and "isosurface" functions. Equations for cylinder bases must be considered too, in order to create a closed surface.
The theorical curve describing diffusion in a cylinder with infinite height and initial concentration equal to zero everywhere except for plane $z = 0$ is (Mammano & Bortolozzi, 2010):

$$C(z,t) = \frac{M}{\sqrt{4\pi Dt}} \exp\left(-\frac{z^2}{4Dt}\right)$$

For the simulation, cylinder height must be much greater than its radius. Initial concentration is set to a given positive value in the voxels whose vertices have two z-coordinates greater than zero and two z-coordinates less than zero, in order to approximate diffusion starting from a plane with uniform concentration.

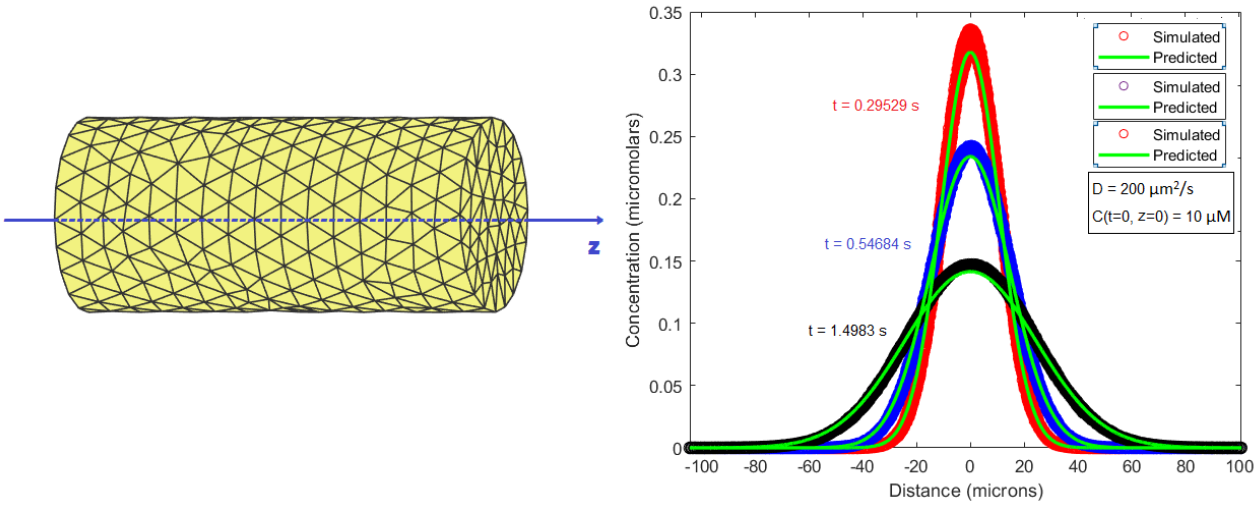


Figure 3.11 : Portion of the meshed cylinder and simulation in a meshed cylinder with height = 100*radius

3.4.5 Mass estimation

To compare concentration evolution in space and time with a theorical functions (equation (1), (2), (3)), it is necessary to know the number of diffusing moles *M*.
If voxels have a similar volume or area, or the mesh is good quality, it is sufficient to calculate *M* as the initial concentration in the central voxel, and multiplying it by its volume or its area.
Otherwise, if voxels sizes are very different from each other, the previous estimation may be incorrect. In fact, our diffusion algorithm doesn't consider voxels volume. In a single step, the number of moles diffusing from i-th voxel to its j-th neighbour with a different volume is not conserved.

A possible solution is to calculate *M* at every iteration of the algorithm as $M(t) = \sum_{i=1}^{N} C(i,t)Vol(i)$, where *N* is the number of voxels.
To improve this estimation, the mesh has been divided into concentric shells with a given width *dx*. In each shell, concentration is calculated as shell volume multiplied by the mean concentration of the voxels belonging to the shell. To each shell corresponds a fixed middle distance from the source.
This estimation doesn't always lead to a perfect estimation, but it works quite fine with very irregular mesh.

4 Implementation in SimulCell

We implemented tetrahedral mesh discretization and the new diffusion algorithm in SimulCell and then we adapted other software features.

4.1 Geometry

In "Geometry 3D" section, the first step was generating the mesh using JigSaw.
The object is created like before, defining its volume in a regular grid of points. For mesh discretization, it is possible to smooth the object previously created, to obtain a more regular surface. This way the meshing software will not be forced to generate a bad quality mesh in order to approximate the surface curvature.
Then centres, neighbours list, distance with neighbours centres were implemented. This kind of information cannot be found a priori in an unstructured mesh, differently from the cube discretization.

Patterns definition could not use the same method as for cubes, because of the various connectivity between the mesh voxels.

- External surface was found selecting tetrahedra lacking at least one neighbour.
- Internal volume was found selecting tetrahedra belonging to volume and not belonging to external surface
- Centre was found using geometrical conditions
- Plane perpendicular to x-axis (with equation $y = y_{center}$) was found selecting the voxels whose vertices had two x-coordinates greater than x_{center} and two x-coordinates smaller than x_{center}.
  The other planes were found analogously.
- Line parallel to x-axis (with equation $y = y_{center} \cup z = z_{center}$) was found selecting the voxels whose vertices had two y-coordinates greater than y_{center} and two y-coordinates smaller than y_{center} and two z-coordinates greater than z_{center} and two z-coordinates smaller than z_{center}.
  The other lines were found analogously.

4.2 Laplacian discretization

For Laplacian discretization, matrix representing permeabilities, defined as the permeability of each voxel with its four neighbours, was extended by adding the permeabilities of the neighbours with the central voxel. In general, these permeabilities are different, as expressed by Laplacian discretization in equation (3.9) and described in (3.4.3). SimulCell "Initial Conditions" panel allows to define permeability from a pattern to another, so this distinction is necessary (also for cubes discretization).
The coefficients of the Laplacian term for each voxel were modified in order to consider the right distance with the neighbouring voxels and multiplied by their corresponding permeability.
It was also added an option to calculate the size of the maximum timestep executable by the equation solver, to avoid negative concentrations, as explained in (3.15).
To check the right implementation of the code, diffusion in the same sphere with reflecting border was simulated both in SimulCell and outside the software. The following picture is the superposition of the plots at three different values for the time.

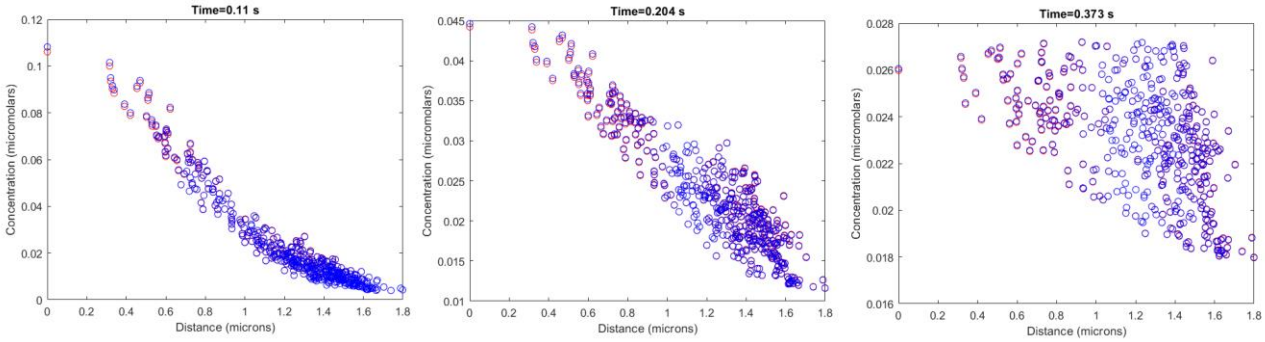


Figure 4.1 Comparison between diffusion performed outside SimulCell (red) and inside SimulCell (blue). Diffusion in a sphere with radius 1.8 µm and reflecting boundaries. In the third picture we can see the effect of reflecting boundary becomes relevant and concentration is approaching stationary conditions.

4.3 Analyze results

In "Analyze Results" panel, a function to visualize the meshed object and its patterns was implemented. When showing a pattern, the remaining voxel are set to become transparent. Pattern definition was explained in paragraph 4.1 and can be found in the Appendix (7).

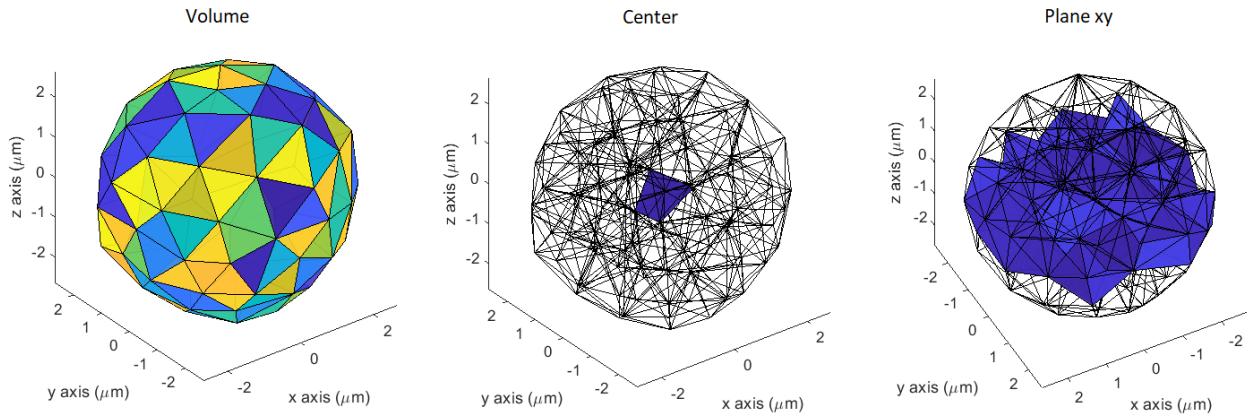


Figure 4.2 Example of patterns definable in SimulCell. The sphere is roughly discretized in order to allow visualizing the single voxels belonging to a pattern.

At last, several adjustments were done in the software code, in order to add new features and to fix some bugs of the previous version.

5 Diffusion in a sphere with calcium influx

We simulated the following reaction in SimulCell:

$$[Ca^{2+}]+[B]\underset{k_{off}^{B}}{\overset{k_{on}^{B}}{\rightleftarrows}}[CaB]\,.$$

It represents chelation of calcium Ca from a buffer molecule B to form the complex CaB (see paragraph 1.1). We added diffusion terms for calcium and buffer to the corresponding kinetic equations for the three variables:

$$\frac{\delta Ca}{\delta t} = -k_{on} Ca * B + k_{off} CaB + D_{Ca} * \nabla^2 Ca$$

$$\frac{\delta B}{\delta t} = -k_{on} Ca * B + k_{off} CaB + D_{B} * \nabla^2 B$$

$$\frac{\delta CaB}{\delta t} = k_{on} Ca * B - k_{off} CaB$$

Simulation was performed in a sphere with radius 15 µm discretized with a tetrahedral mesh until t = 1 s. As initial condition, we set equilibrium values of concentration for Ca, B and CaB, uniform in the volume.

Ca(t=0) = 0.1 µM	B(t=0) = 33.66 µM	CaB(t=0) = 16.34 µM	
K_{on} = 930 s^{-1}	K_{off} = 930 s^{-1}	D_{Ca} = 200 $\mu m^2/s$	D_B = 200 $\mu m^2/s$

Table 5.1 Initial values for variables and parameters used in the simulation.

Between t = 0.1 s and t = 0.2 s we set a calcium influx from a point of the sphere. The influx was defined as follows:

```
%Please write BELOW your statement by using variable and parameter names (no inputs).
if t>0.1 && t<0.2
    INFL =1000;
else
    INFL = 0;
end
```

To realize it in SimulCell, we defined the influx statement in “point along x1” pattern.

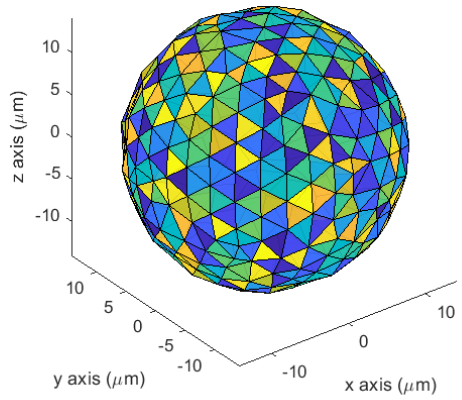


Figure 5.1 Entire volume pattern. The sphere is discretized in about 4000 voxels.

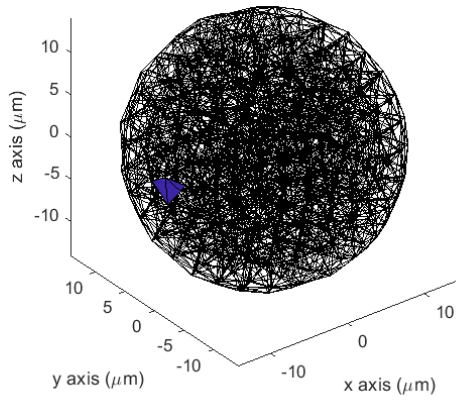


Figure 5.2 “Line along x1” pattern. It is the voxel with calcium influx at t= 0.1 s

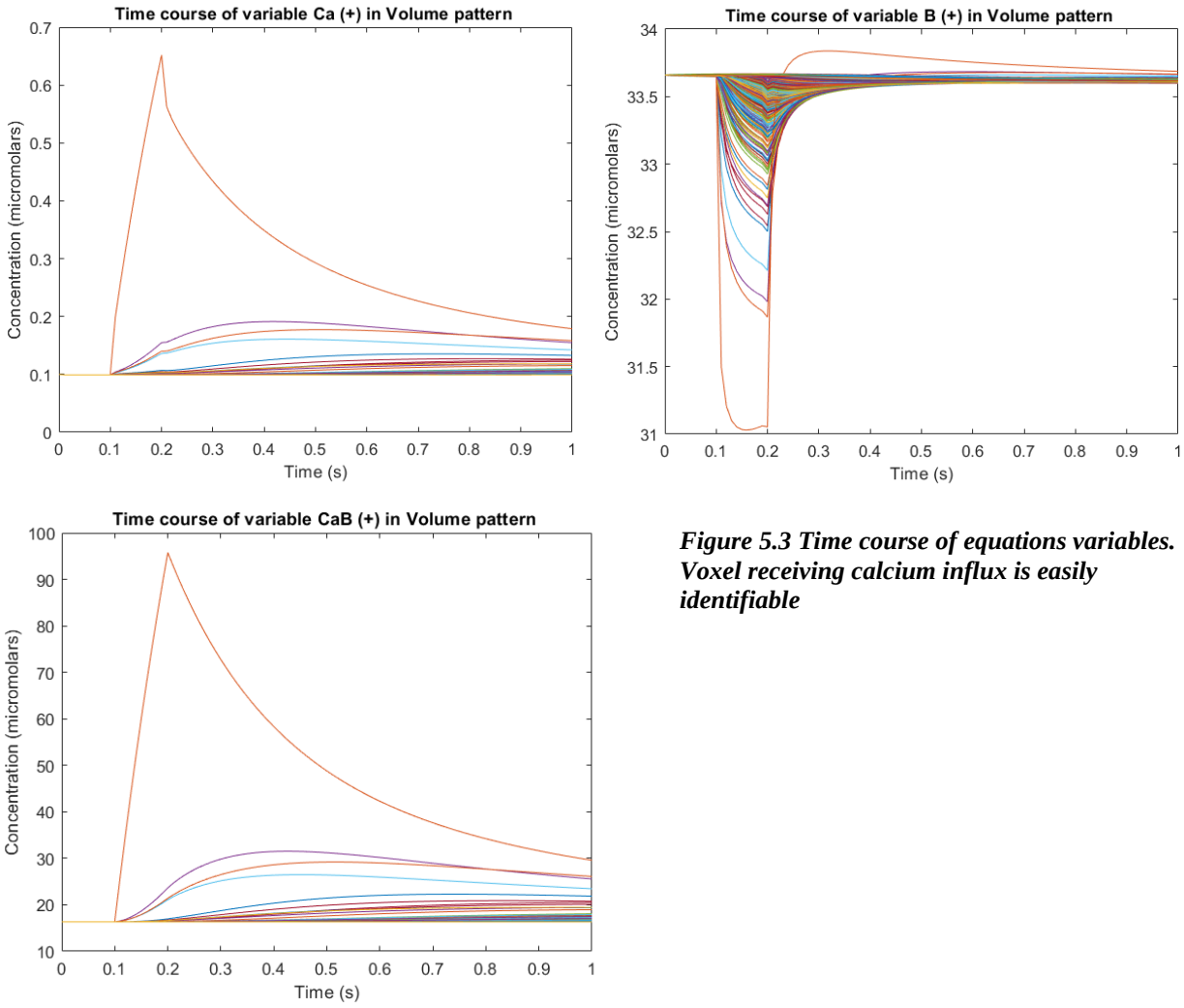


Figure 5.3 Time course of equations variables. Voxel receiving calcium influx is easily identifiable

Simulation results are shown in Figure 5.3.
We can see that for t < 0.1 s, the system is at dynamic equilibrium, as foreseen. Right after the influx, calcium starts to diffuse, its concentration in the source voxel decreases, while it increases in the other voxels.
We can also notice calcium buffer recombination. As a high calcium concentration is available, it binds with the buffer, whose concentration abruptly decreases during the influx.
Instead, CaB concentration increases during the influx. Its concentration decrease, in the source voxel after the impulse, is not due to diffusion towards other voxels, as, in the reaction diffusion equations, CaB diffusive term was neglected. As calcium and buffer leave the central voxel, for t > 0.2 s, the complex concentration starts to be greater than the equilibrium value, so a fraction gets unbound into calcium and buffer.

This simulation was designed to approximate an event of calcium influx into a living cell from a channel opened for calcium signalling.
For instance, in cells like sensory cells, Ca^{2+} concentration increases abruptly because of opening of voltage-dependent channels. The cell localizes calcium signals in time and space to achieve a high bandwidth of signal transmission at specific sites, so it is reasonable to simulate an influx from a single voxel.
These channels create “hotspots”, microdomains of elevated calcium ion concentration. (Bortolozzi, Lelli, & Mammano, 2008)

6 Conclusions

This thesis work has developed a new simple algorithm for simulating diffusion in volumes described by unstructured tetrahedral (3D) and triangular (2D) meshes. The algorithm was derived starting from the concept of random walk and from the discretization of Fick's second law in a structured grid, and then generalized to a mesh with an arbitrary number of neighbouring voxels. In respect to other widely used algorithms developed by Finite Elements Methods (FEM), such as the Galerkin method, our algorithm does not require the solution of a linear system, so it is extremely fast and simple to be implemented in the equation solver code. On the other hand, the computational accuracy is intrinsically limited by the mesh geometry.

The diffusion algorithm was validated simulating a central point of source in a 2D or 3D space meshed by a large number of voxels. The result was compared to the analytical solution of the diffusion equation in an infinite isotropic medium. We obtained a good agreement with the theoretical prediction, especially using regular meshes.

The algorithm was validated also considering the interaction of the diffusing solute with the volume boundary. In particular, we simulated the case of an infinite cylinder with totally reflecting boundaries as the analytical solution is known. The good agreement with the theory allowed to validate the diffusion algorithm.

Tetrahedral mesh generation and simulation features were implemented in SimulCell software, which previously worked with structured cubic meshes. To validate the implemented algorithm, diffusion simulations were performed with identical conditions inside and outside SimulCell software.

Using SimulCell, a reaction-diffusion equation involving a calcium influx at the membrane and an exogenous fluorescent buffer (OGB1) was simulated in a meshed sphere with typical cell dimensions. Parameters and initial conditions for the simulation were set in order to reproduce the Ca2+ influx through a calcium channel in a living cell.

7 Appendix

Function for computing concentrations in two or three dimensional space with absorbing or reflecting borders

```
function A3(Nplot,T,D,c,n,IND,DIST,coord,areas,centers,b_c)

%The function simulates for T seconds m1 multi-dimensional diffusion process,
%where m1 concentration c (mols) is present at time t=0 in the n-th
%point of the grid (indexed by ind array) composed by N points.
%- dt is the suggested time step (in s) while D is the diffusion
%coefficient (micron^2/s).
%- ind is m1 NxM matrix containing the M neighbouring points of each point.
%- dist is m1 NxM matrix containing the M neighbouring distances of each point from its neighbours.
%- coordinates is the array containing the point coordinates in the 1,2,or
%3 dimensional space.
% centers is the N x n_d matrix containing the coordinates of the center of
% each voxel.

% Ac is m1 N-vector containing, for the n-th voxel, the surface of the triangle whose vertices
% are the centers of the neighbours of the n-th voxel.

ind = IND;
dist = DIST;

N=size(ind,1);
M=size(ind,2);
n_d=size(coord,2);

%We introduce virtual distances and neighbours for border voxels. This
%would not change results, but will allow to perform a matricial
%computation

if M==4
    ind_NaN3 = isnan(ind(:,2));
    ind(ind_NaN3,2)=ind(ind_NaN3,1);
    dist_NaN3=isnan(dist(:,2));
    dist(dist_NaN3,2)=dist(dist_NaN3,1);
end

ind_NaN2 = isnan(ind(:,M-1));
ind(ind_NaN2,M-1)=ind(ind_NaN2,M-2);
dist_NaN2=isnan(dist(:,M-1));
dist(dist_NaN2,M-1)=dist(dist_NaN2,M-2);

ind_NaN=isnan(ind(:,M));
ind(ind_NaN,M)=ind(ind_NaN,M-1);
dist_NaN=isnan(dist(:,M));
dist(dist_NaN,M)=dist(dist_NaN,M-1);

%Initialize the simulation
C=zeros(N,1);
C(n)=c;

% Adsorbing or reflecting conditions

if b_c ==1
C(dist_NaN)=0;
end

if n_d==1
    R=sqrt((centers(:)-centers(n)).^2);
elseif n_d==2
    R=sqrt((centers(:,1)-centers(n,1)).^2+(centers(:,2)-centers(n,2)).^2);
elseif n_d==3
    R=sqrt((centers(:,1)-centers(n,1)).^2+(centers(:,2)-centers(n,2)).^2+(centers(:,3)-
centers(n,3)).^2);
end
C_near=zeros(N,M);
C3=C_near;
for k=1:M
    C3(:,k)=C;
end
C_near=C3(ind);
```

```
%find dt

p_max = 0.99;

dist_inv = zeros(N,1);   %somma degli inversi dei quadrati delle distanze con i vicini,
%ne cerco il massimo, che corrisponde m1 p_max

for i=1:M
dist_inv = dist_inv + 1./(dist(ind(:,i))).^2;
end

dt = p_max*M/(2*n_d*D)/max(dist_inv);
%dt=dt/10;

%calculate p matrix

p=2*n_d*D*dt./dist.^2/M;
if b_c == 1
elseif b_c == 2
      p(ind_NaN,M)=0;
      p(ind_NaN2,M-1)=0;
    if n_d == 3
      p(ind_NaN3,M-2)=0;
    end
end
p_sum=sum(p,2);
p0=1-p_sum;

%Perform diffusion simulation until time T
t=dt;
r=0:max(R)/1e4:max(R);
count=1;

%Calculate rounded distances
%Find the best dx
% dx=max(R)/length(R)*100;
dx=0.5;
rounded_R = dx*round(R/dx);
[sorted_R,ind_sort] = sort(rounded_R);     %ascendig order
[uniq_R,ind1] = unique(sorted_R);          %find unic elements and their indexes in sorted_R
total_masses = zeros(1,length(ind1));

%Check for homogeneity of uniq_R
diffR=diff(uniq_R);
diffR=dx*round(diffR/dx);
if sum(diffR~=dx)>1
    disp('Warning! Increase dx to obtain a better estimate of the total mass!');
end

if (n_d == 2)
    vol = pi*(((uniq_R+dx/2).^2)-((uniq_R-dx/2).^2));
    vol(1) = pi*((uniq_R(2)-(dx/2)).^2);
end
if (n_d == 3)
    vol = 4/3*pi*(((uniq_R+dx/2).^3)-((uniq_R-dx/2).^3));
    vol(1) = 4/3*pi*((uniq_R(2)-(dx/2)).^3);
end

%%%%%%%%

while t<T
    C=p0.*C+sum(C_near.*p,2);

    if b_c == 1
       C(dist_NaN)=0;
    end

    %Plot results
    if mod(count,Nplot)==0
        pause(0.1);
        if count == Nplot
            %Plot the concentrations
            figure
            h=plot(R,C,'ro','MarkerSize',6,'DisplayName','Simulated');
            %set(gca,'xlim', [0 ]);
            %set(gca,'ylim',[0 max(C)]);
            xlabel('Distance (microns)');
            ylabel('Concentration (micromolars)');
```

```
            h_tit=title(['Time=' num2str(t) ' s']);

            %Plot the Gaussian curve
            hold on
            C_sort = C(ind_sort);
            for i = 1:length(ind1)
                if i == 1
                    total_masses(1) = mean(C_sort(1:ind1(2)-1))*vol(i);
                elseif i==length(ind1)
                    total_masses(i) = mean(C_sort(ind1(i):end))*vol(i);
                else
                    total_masses(i) = mean(C_sort(ind1(i):ind1(i+1)-1))*vol(i);
                end
            end
            m=sum(total_masses);
            G = m/((sqrt(4*pi*D*t))^n_d)*exp((-r.^2)/(4*D*t));
            h2=plot(r,G,'g','LineWidth',2,'DisplayName','Predicted');
            set(h_tit,'string',['Time=' num2str(t) ' s']);
        else
            C_sort = C(ind_sort);
            for i = 1:length(ind1)
                if i == 1
                    total_masses(1) = mean(C_sort(1:ind1(2)-1))*vol(i);
                elseif i==length(ind1)
                    total_masses(i) = mean(C_sort(ind1(i):end))*vol(i);
                else
                    total_masses(i) = mean(C_sort(ind1(i):ind1(i+1)-1))*vol(i);
                end
            end
            m=sum(total_masses);
            G = m/((sqrt(4*pi*D*t))^n_d)*exp((-r.^2)/(4*D*t));
            set(h2,'xdata',r,'ydata',G);
            set(h_tit,'string',['Time=' num2str(t) ' s']);
            set(h,'xdata',R,'ydata',C);
        end
    end
    if mod(count,Nplot) == 0
        count=count+1
    else
        count=count+1;
    end
    t=t+dt;
    %Update values
    for k=1:M
        C3(:,k)=C;
    end
    C_near=C3(ind);
end
```

Function for 3D mesh generation

```
function [n,ind,dist,coord,areas,centers] = mesh3d_griglia(Nx,Ny,Nz,fact,radiobut)

%given the number of points along x,y,z, the function builds a regular grid of points and
%triangulates it with matlab, or it finds the object surface triangulation and generates mesh with
%JigSaw. It calculates centers, volumes, neighbours, distances etc.

Nxyz = Nx*Ny*Nz;

x1 = zeros(1, Nx*Ny);
y1 = x1;
z = zeros(1, Nxyz);

for i=0:(Ny-1)
    x1(1, (1+Nx*i): ((i+1)*Nx)) = 0:(Nx-1);
end

for i=0:(Ny-1)
    y1(1, (1+Nx*i): ((i+1)*Nx)) =i*ones(1,Nx);
end

z(1, 1: Nx*Ny) = zeros(1,Nx*Ny);
x = x1;
y = y1;
for j=1:(Nz-1)
    x = [x,x1];
```

```
    y = [y,y1];
    z(1, (1+j*Nx*Ny):(j+1)*Nx*Ny) = j*ones(1,Nx*Ny);

end

%%
for i=1:Nxyz
    x(i) = x(i) + rand*fact;
end

for i=1:Nxyz
    y(i) = y(i) + rand*fact;
end

for i=1:Nxyz
    z(i) = z(i) + rand*fact;
end
%%
coord = [x',y',z'];
%%
if radiobut==1,   %matlab mesh
    DT = delaunayTriangulation(x.',y.',z.');
    tri = DT.ConnectivityList;
    figure;
    [K,v] = convexHull(DT);
    trisurf(K,DT.Points(:,1),DT.Points(:,2),DT.Points(:,3));

    [tri_row, ~] = size(tri);

    % triangles centers

    centers = zeros(tri_row,3);

    centers(:,1) = (x(tri(:,1))+x(tri(:,2))+x(tri(:,3))+x(tri(:,4)))/4;
    centers(:,2) = (y(tri(:,1))+y(tri(:,2))+y(tri(:,3))+y(tri(:,4)))/4;
    centers(:,3) = (z(tri(:,1))+z(tri(:,2))+z(tri(:,3))+z(tri(:,4)))/4;

    %triangles neighbours
    ind = neighbors(DT);

    for i=1:tri_row
        ind(i,:) = sort(ind(i,:));
    end

    %distances

    dist = NaN(tri_row,4);

    vicini_NaN = isnan(ind);


    for i=1:tri_row

        if (vicini_NaN(i,2)==1)
            dist(i,1) = sqrt( ( centers(ind(i,1),1) - centers(i,1) )^2 + ( centers(ind(i,1),2) -
centers(i,2) )^2 + ( centers(ind(i,1),3) - centers(i,3) )^2 );

        elseif (vicini_NaN(i,3)==1 )
            dist(i,1) = sqrt( ( centers(ind(i,1),1) - centers(i,1) )^2 + ( centers(ind(i,1),2) -
centers(i,2) )^2 + ( centers(ind(i,1),3) - centers(i,3) )^2 );
            dist(i,2) = sqrt( ( centers(ind(i,2),1) - centers(i,1) )^2 + ( centers(ind(i,2),2) -
centers(i,2) )^2  + ( centers(ind(i,2),3) - centers(i,3) )^2 );

        elseif (vicini_NaN(i,4)==1 )
            dist(i,1) = sqrt( ( centers(ind(i,1),1) - centers(i,1) )^2 + ( centers(ind(i,1),2) -
centers(i,2) )^2 + ( centers(ind(i,1),3) - centers(i,3) )^2 );
            dist(i,2) = sqrt( ( centers(ind(i,2),1) - centers(i,1) )^2 + ( centers(ind(i,2),2) -
centers(i,2) )^2  + ( centers(ind(i,2),3) - centers(i,3) )^2 );
            dist(i,3) = sqrt( ( centers(ind(i,3),1) - centers(i,1) )^2 + ( centers(ind(i,3),2) -
centers(i,2) )^2 + ( centers(ind(i,3),3) - centers(i,3) )^2 );

        elseif(ind(i,4)>0)
            dist(i,1) = sqrt( ( centers(ind(i,1),1) - centers(i,1) )^2 + ( centers(ind(i,1),2) -
centers(i,2) )^2 + ( centers(ind(i,1),3) - centers(i,3) )^2 );
            dist(i,2) = sqrt( ( centers(ind(i,2),1) - centers(i,1) )^2 + ( centers(ind(i,2),2) -
centers(i,2) )^2  + ( centers(ind(i,2),3) - centers(i,3) )^2 );
            dist(i,3) = sqrt( ( centers(ind(i,3),1) - centers(i,1) )^2 + ( centers(ind(i,3),2) -
centers(i,2) )^2 + ( centers(ind(i,3),3) - centers(i,3) )^2 );
```

```
            dist(i,4) = sqrt( ( centers(ind(i,4),1) - centers(i,1) )^2 + ( centers(ind(i,4),2) -
centers(i,2) )^2 + ( centers(ind(i,4),3) - centers(i,3) )^2 );
        end

    end

    %volumes

    areas = zeros(tri_row,1);

    for i=1:tri_row
        areas(i) = 1/6*abs(det([x(tri(i,2))-x(tri(i,1)), y(tri(i,2))-y(tri(i,1)), z(tri(i,2))-
z(tri(i,1));
            x(tri(i,3))-x(tri(i,1)), y(tri(i,3))-y(tri(i,1)), z(tri(i,3))-z(tri(i,1));
            x(tri(i,4))-x(tri(i,1)), y(tri(i,4))-y(tri(i,1)), z(tri(i,4))-z(tri(i,1));]));
    end

    center = [Nx/2,Ny/2,Nz/2];

    d1 = zeros(tri_row,1);

    for i=1:tri_row
        d1(i) = sqrt( (center(1)-centers(i,1))^2 + (center(2) - centers(i,2))^2 + (center(3)-
centers(i,3))^2);
    end

    [~, n] = min(d1);

    msgbox('Done!');

else % JigSaw meshing
    k = boundary(coord,0.1);

    %% Use JigSaw
    geom.point.coord = coord;
    geom.tria3.index = k;

    geom.point.coord(:,4) = 0;
    geom.tria3.index(:,4) = 0;

    try
        savemsh('prova1.msh',geom);%New function
    catch
        makemsh('prova1.msh',geom);%Old function was makemsh
    end

    opts.geom_file = 'prova1.msh'; % file specifying the geom
    opts.mesh_file = 'prova1.msh'; % file for the output mesh
    opts.jcfg_file = 'prova1.jig'; % file for JIGSAW's config.


    %opts.hfun_kern = 'constant';     %OLD PARAMETER
    opts.hfun_hmax = 0.12/(Nx+Ny+Nz)*3*10;
    %opts.hfun_hmin = opts.hfun_hmax / 20;
    %     -- setup the mesh-config for JIGSAW
        opts.mesh_kern = 'delfront';        % use the "frontal" kernal
        opts.mesh_dims = 3;                 % produce tetra output
%     opts.mesh_vol3 = 0.001;

%         opts.geom_feat = true ;

    mesh = jigsaw(opts);                % call JIGSAW

    figure
    drawmesh(mesh);
```

[....]

```
end
```

Patterns definition in SimulCell, Geometry3D

```
%Patterns need to be re-defined with indexes referred to mesh

%%% Whole volume = Pat1
dataM.geometry.patterns.Pat1.V = ones(length(dataF.MeshedObjects.tria4.index),1);
dataM.geometry.patterns.Pat1.indexes = find(dataM.geometry.patterns.Pat1.V==1);
```

```
%%% Surface = Pat2
dataM.geometry.patterns.Pat2.indexes = find(neighbours(:,4)==0);
dataM.geometry.patterns.Pat2.V = (zeros(length(dataF.MeshedObjects.tria4.index),1));
dataM.geometry.patterns.Pat2.V(dataM.geometry.patterns.Pat2.indexes) = 1;

%%% Internal volume = Pat3
dataM.geometry.patterns.Pat3.V = (ones(length(dataF.MeshedObjects.tria4.index),1));
dataM.geometry.patterns.Pat3.V = dataM.geometry.patterns.Pat3.V - dataM.geometry.patterns.Pat2.V;
dataM.geometry.patterns.Pat3.indexes = find(dataM.geometry.patterns.Pat3.V==1);

%%% Center = Pat4
center = zeros(tri_row,3);
for i=1:tri_row
    center(i,:) = center_geom;
end
d1 = sqrt ( (center(:,1)-centers(:,1)).^2 + (center(:,2) - centers(:,2)).^2 + (center(:,3)-
centers(:,3)).^2);
[~, n] = min(d1);
dataM.geometry.patterns.Pat4.V = zeros(length(dataF.MeshedObjects.tria4.index),1);
dataM.geometry.patterns.Pat4.V(n) = 1;
dataM.geometry.patterns.Pat4.indexes = find(dataM.geometry.patterns.Pat4.V==1);

%%% 'Line along x' = Pat5
dataM.geometry.patterns.Pat5.V = zeros(length(dataF.MeshedObjects.tria4.index),1);
diffy(:,1) = mesh.point.coord(tria4(:,1),2)-centers(n,2);
diffy(:,2) = mesh.point.coord(tria4(:,2),2)-centers(n,2);
diffy(:,3) = mesh.point.coord(tria4(:,3),2)-centers(n,2);
diffy(:,4) = mesh.point.coord(tria4(:,4),2)-centers(n,2);
diffz(:,1) = mesh.point.coord(tria4(:,1),3)-centers(n,3);
diffz(:,2) = mesh.point.coord(tria4(:,2),3)-centers(n,3);
diffz(:,3) = mesh.point.coord(tria4(:,3),3)-centers(n,3);
diffz(:,4) = mesh.point.coord(tria4(:,4),3)-centers(n,3);

for i=1:length(diffy)
diffy(i,:) = sort(diffy(i,:));
diffz(i,:) = sort(diffz(i,:));
   if diffy(i,1)*diffy(i,4)<0 && diffz(i,1)*diffz(i,4)<0
   dataM.geometry.patterns.Pat5.V(i)=1;
   end
end
dataM.geometry.patterns.Pat5.indexes = find(dataM.geometry.patterns.Pat5.V==1);

%%% 'Line along y' = Pat6
dataM.geometry.patterns.Pat6.V = zeros(length(dataF.MeshedObjects.tria4.index),1);
diffx(:,1) = mesh.point.coord(tria4(:,1),1)-centers(n,1);
diffx(:,2) = mesh.point.coord(tria4(:,2),1)-centers(n,1);
diffx(:,3) = mesh.point.coord(tria4(:,3),1)-centers(n,1);
diffx(:,4) = mesh.point.coord(tria4(:,4),1)-centers(n,1);
diffz(:,1) = mesh.point.coord(tria4(:,1),3)-centers(n,3);
diffz(:,2) = mesh.point.coord(tria4(:,2),3)-centers(n,3);
diffz(:,3) = mesh.point.coord(tria4(:,3),3)-centers(n,3);
diffz(:,4) = mesh.point.coord(tria4(:,4),3)-centers(n,3);
for i=1:length(diffx)
diffx(i,:) = sort(diffx(i,:));
diffz(i,:) = sort(diffz(i,:));
   if diffx(i,1)*diffx(i,4)<0 && diffz(i,1)*diffz(i,4)<0
   dataM.geometry.patterns.Pat6.V(i)=1;
   end
end
dataM.geometry.patterns.Pat6.indexes = find(dataM.geometry.patterns.Pat6.V==1);

%%% 'Line along z' = Pat7
dataM.geometry.patterns.Pat7.V = zeros(length(dataF.MeshedObjects.tria4.index),1);
diffx(:,1) = mesh.point.coord(tria4(:,1),1)-centers(n,1);
diffx(:,2) = mesh.point.coord(tria4(:,2),1)-centers(n,1);
diffx(:,3) = mesh.point.coord(tria4(:,3),1)-centers(n,1);
diffx(:,4) = mesh.point.coord(tria4(:,4),1)-centers(n,1);
diffy(:,1) = mesh.point.coord(tria4(:,1),2)-centers(n,2);
diffy(:,2) = mesh.point.coord(tria4(:,2),2)-centers(n,2);
diffy(:,3) = mesh.point.coord(tria4(:,3),2)-centers(n,2);
diffy(:,4) = mesh.point.coord(tria4(:,4),2)-centers(n,2);
for i=1:length(diffx)
   diffx(i,:) = sort(diffx(i,:));
   diffy(i,:) = sort(diffy(i,:));
   if diffx(i,1)*diffx(i,4)<0 && diffy(i,1)*diffy(i,4)<0
   dataM.geometry.patterns.Pat7.V(i)=1;
   end
end
```

```
dataM.geometry.patterns.Pat7.indexes = find(dataM.geometry.patterns.Pat7.V==1);

%%% 'point along x1' = Pat8 e Pat9   (Pat8 = intersection between Pat2 and Pat5)
dataM.geometry.patterns.Pat8.V = zeros(length(dataF.MeshedObjects.tria4.index),1);
dataM.geometry.patterns.Pat9.V = zeros(length(dataF.MeshedObjects.tria4.index),1);
[~,p] = min(centers(dataM.geometry.patterns.Pat5.indexes,1));
dataM.geometry.patterns.Pat8.indexes = dataM.geometry.patterns.Pat5.indexes(p);
[~,p] = max(centers(dataM.geometry.patterns.Pat5.indexes,1));
dataM.geometry.patterns.Pat9.indexes = dataM.geometry.patterns.Pat5.indexes(p);
dataM.geometry.patterns.Pat8.V(dataM.geometry.patterns.Pat8.indexes) = 1;
dataM.geometry.patterns.Pat9.V(dataM.geometry.patterns.Pat9.indexes) = 1;
%%% 'point along y1' = Pat10 e Pat11
dataM.geometry.patterns.Pat10.V = zeros(length(dataF.MeshedObjects.tria4.index),1);
dataM.geometry.patterns.Pat11.V = zeros(length(dataF.MeshedObjects.tria4.index),1);
[~,p] = min(centers(dataM.geometry.patterns.Pat6.indexes,2));
dataM.geometry.patterns.Pat10.indexes = dataM.geometry.patterns.Pat6.indexes(p);
[~,p] = max(centers(dataM.geometry.patterns.Pat6.indexes,2));
dataM.geometry.patterns.Pat11.indexes = dataM.geometry.patterns.Pat6.indexes(p);
dataM.geometry.patterns.Pat10.V(dataM.geometry.patterns.Pat10.indexes) = 1;
dataM.geometry.patterns.Pat11.V(dataM.geometry.patterns.Pat11.indexes) = 1;

%%% 'point along z1' = Pat12 e Pat13
dataM.geometry.patterns.Pat12.V = zeros(length(dataF.MeshedObjects.tria4.index),1);
dataM.geometry.patterns.Pat13.V = zeros(length(dataF.MeshedObjects.tria4.index),1);
[~,p] = min(centers(dataM.geometry.patterns.Pat7.indexes,3));
dataM.geometry.patterns.Pat12.indexes = dataM.geometry.patterns.Pat7.indexes(p);
[~,p] = max(centers(dataM.geometry.patterns.Pat7.indexes,3));
dataM.geometry.patterns.Pat13.indexes = dataM.geometry.patterns.Pat7.indexes(p);
dataM.geometry.patterns.Pat12.V(dataM.geometry.patterns.Pat12.indexes) = 1;
dataM.geometry.patterns.Pat13.V(dataM.geometry.patterns.Pat13.indexes) = 1;
```

Laplatian coefficients definition in SimulCell, formatOdeMEX3D

```
for k=1:L_var
    perm = dataM.variables.laplacian{k};
    perm_new = zeros(L_vox,8);
    for j=1:L_vox
        perm_new(j,1) = perm(j,1); %neighbour 1
        perm_new(j,2) = perm(j,2); %neighbour 2
        perm_new(j,3) = perm(j,3); %neighbour 3
        perm_new(j,4) = perm(j,4); %neighbour 4
        %bisogna distinguere tra la permeabilità del voxel centrale verso i
        %vicini (perm1, perm2, perm3, perm4) e la permeabilità dei vicini
        %verso il centrale perm1_n, perm2_n, perm3_n, perm4_n)
        perm_new(j,5) = perm(neighbours(j,1),find(neighbours(neighbours(j,1),:)==j));
        if neighbours(j,2)~=0
        perm_new(j,6) = perm(neighbours(j,2),find(neighbours(neighbours(j,2),:)==j));
        else
        perm_new(j,6) = 0.0;
        end
        if neighbours(j,3)~=0
        perm_new(j,7) = perm(neighbours(j,3),find(neighbours(neighbours(j,3),:)==j));
        else
        perm_new(j,7) = 0.0;
        end
        if neighbours(j,4)~=0
        perm_new(j,8) = perm(neighbours(j,4),find(neighbours(neighbours(j,4),:)==j));
        else
        perm_new(j,8) = 0.0;
        end
        %distances between voxel centers
        coeff1 = num2str(perm_new(j,5)/dist(j,1)^2*1.5);
        coeff2 = num2str(perm_new(j,6)/dist(j,2)^2*1.5);
        coeff3 = num2str(perm_new(j,7)/dist(j,3)^2*1.5);
        coeff4 = num2str(perm_new(j,8)/dist(j,4)^2*1.5);
        coeff5 = num2str((perm_new(j,1)/dist(j,1)^2 + perm_new(j,2)/dist(j,2)^2 +
perm_new(j,3)/dist(j,3)^2 + perm_new(j,4)/dist(j,4)^2)*1.5);
        coeff_str{j} = {coeff1 coeff2 coeff3 coeff4 coeff5};
    end
    dataM.variables.laplacian_new{k} = coeff_str;
end
```

[….]

Writinig string coefficients and variables for Laplacian discretization

```
if neighbours(j,1) == 0   %vicino mancante: non viene aggiunto il termine del laplaciano
                s_add1 = '0.0';
                else
                s_add1=['s(' NumToStr{L_vox*(k-1)+ neighbours(j,1) } ')']; % neighbor 1
                end

                if neighbours(j,2) == 0
                s_add2 = '0.0';
                else
                s_add2 = ['s(' NumToStr{L_vox*(k-1)+neighbours(j,2)} ')']; % neighbor 2
                end

                if neighbours(j,3) == 0
                    s_add3 = '0.0';
                else
                    s_add3 = ['s(' NumToStr{L_vox*(k-1)+ neighbours(j,3) } ')']; % neighbour 3
                end

                if neighbours(j,4) == 0
                    s_add4 = '0.0';
                else
                    s_add4=['s(' NumToStr{L_vox*(k-1)+neighbours(j,4)} ')']; % neighbour 4
                end

                s_add = [ s_add1 '*' coeff1 '+' s_add2 '*' coeff2 '+' s_add3 '*' coeff3 '+' s_add4
'*' coeff4 '-' s_add12 '*' coeff5 ];

           end

        %Insert Laplacian in the equation
              if dataM.cubes == 1
                for kk = flipdim(int_str,2),
                    eq(kk:(kk+length(Lapl)-1))=[];
                    eq=[eq(1:(kk-1)) '(' s_add ')/' num2str(dx2) eq(kk:end)];
                end
              elseif dataM.mesh == 1
                for kk = flipdim(int_str,2),
                    eq(kk:(kk+length(Lapl)-1))=[];
                    eq = [eq(1:(kk-1)) '(' s_add ')' eq(kk:end)];
                end
              end
```

[….]

8 Bibliography

Bortolozzi, M., Lelli, A., & Mammano, F. (2008). Calcium microdomains at presynaptic active zones of vertebrate hair unmasked by stochastic deconvolution. In *Cell Calcium* (p. 158-168).

Carafoli, E., & Krebs, J. (2016). Why Calcium? How Calcium Became the Best Communicator? *Journal of biological chemistry*.

Fischer, H. P. (2008). Mathematical Modeling of Complex Biological Systems. *Alcohol Research Health; 31(1)*, p. 49-59.

Issa, N. P. (1994). Clustering of Ca2+ channels and Ca2+-activated K+ channels at fluorescently labeled presynaptic active zones of hair cells. In *Neurobiology (91)* (p. 7578-7882).

Lelli, A. P., Martini, M., Ciubotaru, C. D., Prigioni, I., Valli, P., Rossi, M. L., & Mammano, F. (2003). Presynaptic Calcium Stores Modulate Afferent Release in Vestibular Hair Cells. *The journal of neuroscience, 23*(17), p. 6894-6903.

Mammano, F., & Bortolozzi, M. (2010). Ca2+ Imaging: Principles of Analysis and Enhancement. In *Measurement Methods* (p. 57-80). Humana Press, a part of Springer Science+Business Media.

Metcalfea, G., Speetjensb, M., Lesterc, D., & Clercxd, H. (2012). Beyond Passive: Chaotic Transport in Stirred Fluids. In *Advances in applied mechanics* (Vol. 45, p. 109-188).

Nowycky, M. C. (1993). Time course of calcium and calcium-boundbuffers following calcium influx in a model cell. *Biophysical Journal*(64), p. 77-91.

Rispoli, G. (2001). Dynamics of intracellular calcium in hair cells isolated from the semicircular canal of the frog. In *Cell Calcium 30(2)* (p. 131-140).

Roberts, W. (1994). Localization of Calcium Signals by a Mobile Calcium Buffer in Frog Saccular Hair Cells. *The Journal of Neuroscience*, p. 3246-3262.

Rudin, A., & Choi, P. (2013). Diffusion in polymers. In *The Elements of Polymer Science & Engineering (Third Edition)* (p. 275-304).

SimWiki. (s.d.). Tratto da https://www.simscale.com/docs/content/simwiki/preprocessing/whatisamesh.html.

Tucker, T. F. (1995). Confocal Imaging of Calcium Microdomains and Calcium Extrusion in Turtle Hair Cells. In *Neuron (15)* (p. 1323-1335).

Wikipedia. (2019). Tratto da Leggi di Fick.

Wikipedia. (2019, May). Tratto da Types of mesh.