View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by HAL-ENS-LYON

Numerical simulation of a stroke: numerical problems and methodology

Stéphane Descombes, Thierry Dumont

To cite this version:

Stéphane Descombes, Thierry Dumont. Numerical simulation of a stroke: numerical problems and methodology. 2007. <hal-00150951>

HAL Id: hal-00150951

https://hal.archives-ouvertes.fr/hal-00150951

Submitted on 4 Jun 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

NUMERICAL SIMULATION OF A STROKE : NUMERICAL PROBLEMS AND METHODOLOGY

STÉPHANE DESCOMBES AND THIERRY DUMONT

Contents

1. Introduction

The numerical simulation of a stroke is a challenging problem, with many sources of numerical difficulties: a complex geometry, a large number of nonlinear partial differential equations (PDE) to be solved; moreover some important difficulties are a consequence of the mathematical structure of the problem.

For the non specialists of scientific computing it could be surprising that there exist no ready made software solutions to this problem: but nonlinear PDE are actually too complicated mathematical objects to allow it. But for the mathematician, specialist of the theory of PDE and of their numerical approximation, there is a general framework to which the equations of the model belong: the so called systems of Reaction–Diffusion equations. These systems occur very often in the nature, from chemistry to pattern formation and predator–prey models. The authors have been working for some years on the mathematical analysis of numerical methods for Reaction-Diffusion systems, as well as on developing numerical software. The "stroke problem" has been and remains a constant and enlighting source of questions and problems, both from the mathematical point of view and from the

software engineering side; thus it will contribute to the elaboration of numerical tools for problems occuring in other parts of Science.

We often use simplified problems in spatial dimension 1 or 2. The reason is that such models can be solved very quickly, but the most part of the mathematical and numerical difficulties remains; from an other point of view they allow parameter fitting and numerical experimentations in a simplified way. On the other side, multi dimensional simulations are, nowadays, too expensive to allow such parameter experimentations.

The paper is organized as follows : In section 2, we define Reaction-Diffusion systems, in section 3, we analyze the 0-dimensional case which gives a system of ordinary differential equations. We show the numerical difficulties and give some definitions of stability. Section 4 is devoted to the n-dimensional case and we define the Alternate Direction Methods. Numerical simulations of an ischemic stroke are given in section 5.

2. A Reaction–Diffusion system

Let $m = 1, 2$, or 3 be the spatial dimension and let Ω be a subset of $\mathbb{R}^m$. Reaction-Diffusion systems are PDE systems of the form:

$$\begin{cases} \dfrac{\partial u_i}{\partial t}(\vec{x},t) - \varepsilon_i \Delta u_i(\vec{x},t) = f_i(u_1(\vec{x},t), \dots u_n(\vec{x},t)), & \vec{x} \in \Omega, \\ u_i(\vec{x},0) = u_i^0(\vec{x}), & \vec{x} \in \Omega. \end{cases} \tag{1}$$

ε_i, $i = 1, \dots, n$, belongs to $\mathbb{R}$, ($\varepsilon_i > 0$) and the functions u_i^0, $i = 1, \dots, n$, are the initial conditions; $\vec{x} = (x, y, z)$ in dimension 3, $\vec{x} = (x, y)$ in dimension 2 (and $\vec{x} = x$ in dimension 1).

Let us explain the origin of such systems and the notations :

(1) Most of the diffusion phenomenas are modelled by the *heat equation* (Joseph Fourier, 1822):

$$\frac{dT(\vec{x},t)}{dt} - \varepsilon \Delta T(\vec{x},t) = 0.$$

which describes the propagation of the temperature in an homogeneous media, as well as the mixing of gas and many other diffusion processes. The Laplacian operator Δ is defined (in dimension $n = 3$) by :

$$\Delta T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}.$$

In dimension 1, $\Delta T(x)$ is the second order derivative of u at x.

(2) Chemical reactions, without spatial effects (for example in perfectly mixed reactors), are modelled by systems of ordinary differential equations (ODE), which describe the time evolution of the concentrations of the different reactants u_i:

$$\frac{du_i(t)}{dt} = f_i(u_1(t), \dots u_n(t)), \ i = 1, 2, ...n.$$

(if we have n reactants).

(3) Let us now imagine chemical products which react together *and* diffuse (like pollutants in the atmosphere, for example): they will be governed by Reaction–Diffusion systems like (1).

A derivation of the heat equation and many examples of Reaction–Diffusion based models in biology can be found in [12] and [13].

A system like (1) must be equipped with boundary conditions : we will be interested only by no flux (homogeneous Neumann) boundary conditions. In this paper, Ω will be the brain. In the stroke model, n is equal to 20 (or more); some equations (about 10) do not have diffusion (that is to say that $\varepsilon_i = 0$). The model is defined by the functions f_i and we refer to [7] (and [6]) for more details.

Actually, Δ must be replaced by a more complicated operator, since the diffusion is not constant in the brain; we will keep Δ in the whole paper for the sake of simplicity, with no loss of generality.

A mathematical study of such systems of equations is out of the scope of this paper, but let us mention the existence in some systems of progressive waves: given an initial condition $u_i(\vec{x}, 0) = \phi_i(\vec{x}),\ i = 1 \dots n$, there exist solutions of the form $u_i(\vec{x}, t) = \phi_i(\vec{x} + t\vec{c})$ ($\vec{c}$ is thus a speed (and a direction) of propagation of the wave); this seems to be relevant of the depolarization waves observed in real strokes.

3. The 0-dimensional case: a stiff system of ODE

The 0-dimensional case is the most simplified model as possible: all the spatial effects are neglected and we focus on the solution of the system of ordinary differential equations :

$$\frac{du_i}{dt} = f_i(u_1, \dots u_n), \tag{2}$$

with the initial conditions $u_i(t = 0) = u_i^0$, for all i belonging to $\{1, \cdots, n\}$. We point out that u_i, $i = 1, \dots, n$, only depends on the time and that the most part of the modelling is actually *in* these equations [6] !

But even here, the numerical solution is not trivial, even if, in this case, efficient tools are available : the reason is the *stiffness* of the system.

3.1. Stiff ODE and stiff systems of ODE.

Let us start by a very classical example, with one ordinary differential equation :

$$\frac{du}{dt} = f(u),$$

with $u(t = 0) = \phi$. Let h be a time step, the most simple method available is the Explicit Euler method:

$$\frac{u_j - u_{j-1}}{h} = f(u_{j-1}),$$

with $u_0 = \phi$. We hope that u_j will approximate $u(jh)$, the exact solution at time $t = jh$. The word *explicit* means that we only need to apply f to some known quantities when computing with this method. Thus, the method is very simple and not computationally expensive. But the reader should look at the case

$$\frac{du}{dt} = -100\ u$$

and $\phi = 1$. The exact solution is $u(t) = e^{-100t}$; let us forget that we know it, and apply the Explicit Euler method with $h = 1$. We obtain: $u_{j+1} = (1 - 100\ h)\ u_j$, that is to say $u_j = (-99)^j$! The computed solution is clearly not bounded, and thus not acceptable.

Alternatively, we can also use Implicit Euler method :

$$\frac{u_j - u_{j-1}}{h} = f(u_j)$$

that is to say: $(u_j - h\ f(u_j)) = u_{j-1}$. The method is said to be *implicit* because f is applied to unknown quantities and thus, a (possibly) non linear equation must be solved at each step. If we apply it to the preceding example, we obtain: $u_j = u_{j-1}/101$, which is a bounded sequence, decreasing, even if it is not a very accurate approximation of the exact solution.

We recall that, if f is not linear, an algebraic equation must be solved at each step, which is computer time consuming.

In the case of the single linear equation

$$\frac{du}{dt} = -\lambda\ u,$$

the value $1/\lambda$ is the time constant of the equation, a measure of the time needed by the solution to relax. With the Explicit Euler method, the instability is a consequence of the choice of the time step h which is very large when compared to the time constant of the equation ($1/100$ with $\lambda = 100$). If we had choosen a very small time step in the preceding example ($h < 1/100$), the solution would have been acceptable. But actually, when solving true problems with the explicit method, the time step will be so small that the computing time will be much greater than with the implicit method.

If we want to solve systems of ODE, we will have to face with the problem that the different equations will have different time constants, for example 1 and $1/100000$, and that we do not want to be restricted in our choice of the time step by the "fastest" equation.

Let now λ be a complex number with a negative real part, the reader can keep in mind the example of $\lambda = -100$. We consider the ODE

$$\frac{du}{dt} = \lambda\ u,$$

with $u_0 = 1$. The solution of this ODE is still explicit and is given by $u(t) = e^{\lambda\, t}$. For this equation, the Explicit Euler method gives

$$\frac{u_j - u_{j-1}}{h} = \lambda\, u_{j-1},$$

thus

$$u_j = (1 + h\lambda)\, u_{j-1}.$$

The Implicit Euler method gives

$$u_j = \frac{u_{j-1}}{(1 - h\lambda)}.$$

Most of common numerical methods can be written in the form

$$u_j = R(h\lambda)u_{j-1}$$

with R a rational function. Following the previous remark, we do not want to be restricted in our choice of the time step. We then impose that $|R(h\lambda)| \leq 1$, without any restriction on h, thus the sequence can not explode and has the same behavior as the exact solution since for $t \geq 0$,

$$\left|e^{\lambda\, t}\right| \leq 1.$$

This property is called A-stability. An example of A-stable method is the Implicit Euler method. But we can do more: when the real part of λ is strictly negative, then

$$\left|e^{\lambda\, t}\right| < 1$$

and

$$\lim_{t\to+\infty} \left|e^{\lambda\, t}\right| = 0.$$

When R satisfies

$$\lim_{\Re z\to-\infty} |R(z)| < 1,$$

the method is *strongly* A-stable and when, in addition, R satisfies

$$\lim_{\Re z\to-\infty} |R(z)| = 0,$$

the method is said to be L-stable. To summarize when a numerical method is L-stable we have the behavior nearest to the exact solution and this independently of λ. This is why the L-stability is so important when the system has very different time constants and a L-stable method can be very efficient on a problem of the form

$$\frac{du}{dt} = Au,$$

with A a matrix with several eigenvalues of different scales.

The property to show very different time constants in the systems of ODE is known as *stiffness* (see [9]). It can be checked by looking at the eigenvalues $\lambda_i, i = 1, \cdots, n$ of the jacobian matrix J of $f_i(x_1, \cdots, x_n), i = 1, \cdots, n$ defined by

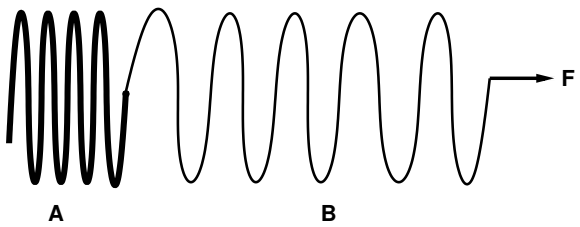
$$J_{ij} = \frac{\partial f_i}{\partial x_j},\ 1 \le i, j \le n.$$

If the ratio

$$\frac{\max_i |\lambda_i|}{\min_j |\lambda_j|}$$

is large, the system is said to be stiff.

To get an intuitive idea of stiffness in ODE systems, the reader can think to a system of two coupled springs, with very different stiffness, submitted to a time varying force F:



the interesting dynamics of the system is the dynamic of the spring B (if the stiffness of A is very large, the movements of A will be approximatively 0). If we try to integrate the system of ODE associated to this system of springs with an explicit method, we will need to use time steps small enough to integrate the dynamic of the the strong spring A; but with methods adapted to stiff systems of ODE (which are all implicit methods) we will be able to choose time steps governed by the dynamics of B.

3.2. **An example of stiff system : The stroke system.** We have computed the eigenvalues of the Jacobian (J was approximated numerically by finite differences), near the equilibrium (that is to say for u such that $f_i(u)$ is about zero, for all $i = 1, \cdots, n$).

We found (in increasing order):

$(-115191, -26522.1, -439.861, -346.076, -63.0174, -51.6393,$

$-17.5434, -0.0859597)$ ... plus some values near zero and thus, the system is *very stiff*.

We also computed the associated eigenvectors; we could then verify that none of them is parallel to one axis: there is consequently no hope to eliminate some unknowns, and the full system of ODE must be solved.

3.3. **Numerical methods for stiff systems of ODE.** Suppose that we want to solve the stroke's system of ODE for $t \leq 4000$ (seconds). With an explicit method, like the Explicit Euler method which is not L-stable, we would have to choose a time step h less than 10^{-6} seconds, and consequently perform about 4.10^9 time steps, which is not acceptable, even if each step is not numerically expensive.

Different methods can be used for such stiff systems but all are implicit (and thus, algebraic systems of equations must be solved), and consequently they are computationally expensive, but it is the price to pay to be A-stable and eventually L-stable. The Implicit Euler method is L-stable but with a poor precision : at time $T = nh$, one can measure some norm $||\ ||$ of the difference between the computed solution u_h and the real solution u. If

$$||u(t) - u_h(nh)|| = O(h^p)$$

the method is said to be of order p. We naturally want p as large as possible (because this implies that a reduction of the time step by 2 (for example) will improve the solution by about 2^p), with a given computing cost. Unfortunately the Implicit Euler method is only of order 1. Getting stability and high order with the same method is a challenging mathematical numerical problem. An answer is given by the so called RadauIIA methods. For our ordinary differential equation :

$$\frac{du}{dt} = f(u),$$

with $u(t = 0) = \phi$, the RadauIIA method of order 3 which is A-stable is given by by the resolution of the two-stages non linear problem

$$k_1 = f\left(u_{j-1} + \frac{h}{12}(5k_1 - k_2)\right),$$
$$k_2 = f\left(u_{j-1} + \frac{h}{4}(3k_1 + k_2)\right)$$

and

$$\frac{u_j - u_{j-1}}{h} = \frac{3k_1 + k_2}{4}.$$

The most performant method that we tried is the so called Radau5 method, the RadauIIA method of order 5 with step size control, which belongs to the category of A-stable and L-stable methods (see [9]) and is built with a three-stages non linear problem. Actually, classical ODE toolboxes (like in Matlab) are robust enough to solve this problem, and simulations can be found in [6].

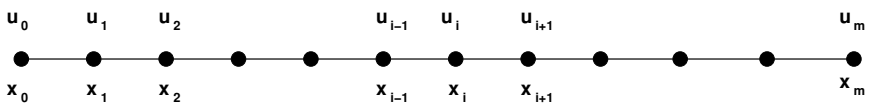


FIGURE 1. Finite differences in dimension 1.

The key idea is that A-stable (or L-stable) method + high precision allow to choose large time steps: as a result, the method, even if each step is expensive, will be much less computer time demanding then the explicit methods.

4. The n-dimensional case

PDEs have no explicit solutions and it is necessary to approximate them with discrete computable counterparts.

4.1. **Spatial discretization.** All the spatial derivatives must be replaced by discrete versions: the most simple possibility (which is the one we use) is finite differences: given an n-dimensional mesh, derivatives are approximated by quotients. For example, in dimension 1 (see figure 1), $\partial u/\partial x(x_i)$ will be replaced by $(u_{i+1} - u_i)/\tau$ where the x_i's are the nodes of the mesh, $\tau = x_{i+1} - x_i$, and u_i the approximated values of u at x_i.

In the same way, second order derivatives are approximated by

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{\tau^2},$$

and the Laplacian in dimension 2 (see figure 2) will be approximated by

$$\frac{u_{i-1,j} - u_{i+1,j} - 4u_{i,j} + u_{i,j-1} - u_{i,j+1}}{\tau^2},$$

and so on.

The continuous unknown u is replaced by a vector of unknowns at the points x_i and the laplacian Δ is replaced by a matrix A. The matrix A has only 3 non zero coefficient in dimension 1 (5 in dimension 2, 7 in dimension 3).

As result a PDE like the heat equation :

$$\frac{du}{dt} - \varepsilon \Delta u = f$$

will be replaced by a (linear) system of ODEs :

$$\frac{d\vec{U}}{dt} - \varepsilon A\vec{U} = \vec{F}$$

with $\vec{U}$ a vector of all the discrete unknowns on the finite difference mesh.

Concerning the complexity, we can make the two following remarks :

- if the mesh is built on the pixels of a MNR picture, the number of nodes in the mesh will generally be more than 10^5 in dimension 2 and the resulting number of unknowns (for 20 equations) will be about 210^6. In dimension 3 such a fine resolution seems untractable, but one must be prepared to manage a set of 10^8 unknowns.

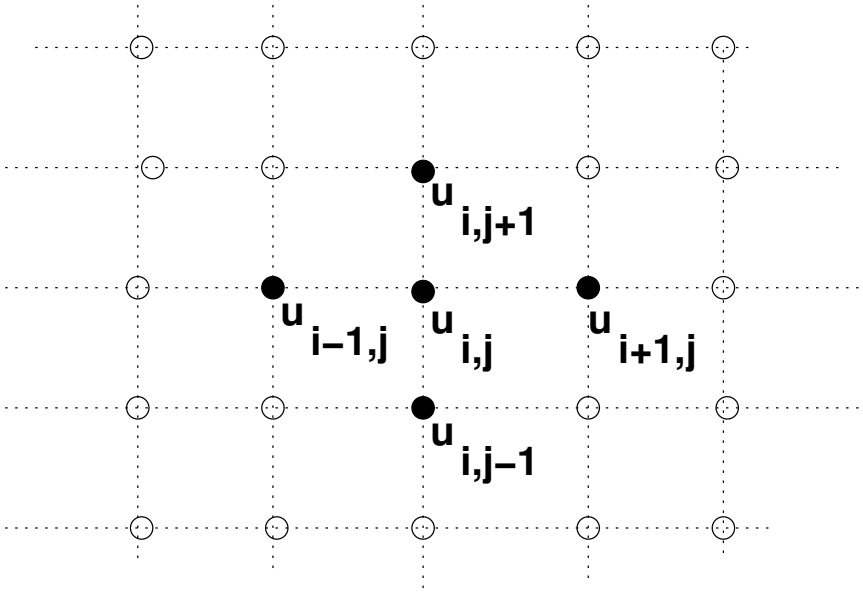


FIGURE 2. Finite differences in dimension 2.

- for the heat equation, the system of ODEs is stiff and consequently implicit method must be used.

4.2. **Time discretization.** The finite difference approach applied to the stroke's system of Reaction–Diffusion equations will create a large system of ODE's to be solved (about 210^6 unknowns in 2d). Let us first present two approaches which *cannot* be used:

4.2.1. *The method of lines.* If we look at one Reaction-Diffusion equation, we need to solve, after spatial discretization, a system of the form:

$$\frac{d\vec{U}}{dt} = \varepsilon A\vec{U} + \vec{F}(\vec{U})$$

One can think to use directly a solver of systems of ODEs (this is the so called method of lines), but let us recall that the system is stiff and, due to the term $\vec{F}(\vec{U})$, it is non-linear. At each time step, a large system of algebraic equations will be solved and that is very time consuming. But moreover, as we want to solve a system of 20 Reaction-Diffusion equations the numerical cost would be largely too expensive, even on supercomputers: we need some sort of "divide and conquer" approach.

4.2.2. *Implicit–Explicit Methods.* Let us present the simplest of these methods, for the case of one Reaction-Diffusion equation. $\vec{U}_k$ denoting the approximated solution at time kh, the method is:

$$\frac{\vec{U}_{k+1} - \vec{U}_k}{h} - \varepsilon A\vec{U}_{k+1} = \vec{F}(\vec{U}_k).$$

One must solve a linear system (because diffusion is taken implicitly) at each step:

$$(h^{-1}Id - \varepsilon A)\vec{U}_{k+1} = \textit{some known quantity.}$$

but the nonlinear term is taken explicitly.

More precise (but not really more expensive) methods of the same type are described and analyzed in [1].

The main advantage of these methods is that only *linear* system must be solved and, in the case of systems of Reaction–Diffusion systems, these systems of linear equations are decoupled ("divide and conquer"!), which result in a very time efficient algorithm.

But the drawback is that, due to the explicit computation of the reaction terms, these method are adapted only to systems with non stiff reaction terms. Let us recall that the stroke's system is very stiff, and these method can only work with very small time steps (about 10^{-6}), which would result in an prohibitive computing time (about 4 10^9 steps for one hour of simulation: remember that if each step is not very time consuming, we need to solve n large linear systems!).

Finally, the only possible methods which can (and proved to be able to) solve the stroke's problem seems to be one of :

4.2.3. *The "alternate direction methods".* The idea is as old as numerical analysis and was used and analyzed by the Soviet school in the years 60 (see for example [11]). In these times, the main interest was the economy of computer memory. The idea, applied to spatially discretized Reaction–Diffusion equations is to solve alternatively problems R_h (the Reaction) and D_h (the Diffusion) defined by :

- R_h : starting from some initial condition, solve for a time step of h :

$$\frac{d\vec{U}}{dt} = \vec{F}(\vec{U}),$$

- D_h : starting from some initial condition, solve for a time step of h :

$$\frac{d\vec{U}}{dt} = \varepsilon A\vec{U}.$$

One can define different numerical methods :

- in the Lie methods one apply successively D_h and R_h (or R_h and D_h) to $\vec{U}_k$ to get $\vec{U}_{k+1}$ (ie $\vec{U}_{k+1} = R_h o D_h\ \vec{U}_k$, where o denotes the classical composition of functions).
- in the Strang method [16] one apply successively, $R_{h/2}$, D_h and $R_{h/2}$ (or $D_{h/2}$, R_h and $D_{h/2}$) to $\vec{U}_k$ to get $\vec{U}_{k+1}$ (ie $\vec{U}_{k+1} = R_{h/2} o D_h o R_{h/2}\ \vec{U}_k$).

Let us quote the main advantages of these methods :

(1) *A "divide and conquer" approach:*
   - Reaction and Diffusion are decoupled,
   - The solution of the D_h problems, in the case of a system of n Reaction–Diffusion equations is reduced to the solution of n independant heat equations, and thus the complexity is reduced. One must keep in mind that the cost (measured in number of multiplications and additions) of the solution of a linear system of equations of size m grows much faster than m; consequently, the solution of n systems of size m is much less expensive than the solution of one system of size nxm.

- Concerning the R_h problems, one immediately see that they are decoupled in systems of ODEs (as many systems of size n as nodes in the finite difference mesh) and that *all* these systems are independant.
- We deduce that the R_h problems can be treated very efficiently in parallel (on shared memory machines, as well as on distributed memory machines),
- The D_h problems being decoupled in n independant problems, parallelism is possible with at most n processors. A higher level of parallelism would need more sophisticated techniques, like domain decomposition.

(2) *Robustness:* If the solvers used for D_h and R_h problems are adapted to stiff problems, these methods are also adapted to stiff systems of Reaction–Diffusion (see [5], [3], [4] and [14]).

Concerning the precision, the Lie methods are of order 1, and the Strang methods are of order 2. Following [3], it seems better to use the $R_{h/2}oD_hoR_{h/2}$ method than the $D_{h/2}oR_hoD_{h/2}$ one (as the Reaction term is stiff).

4.3. **Implementation.** We have implemented the alternate directions methods into a comprehensive software for the solution of Reaction–Diffusion systems (publicly available, see [8]). Nowadays the code solves 1 dimensional and 2 dimensional problems. The adaptation for 3d problems is a work in progress.

- For the solution of the D_h problems, the time discretization must be of order 2 in order to keep the second order of Strang method; we tried different methods : mainly the Crank-Nicolson method and L-stable methods.
- For the solution of the R_h problems, one must use methods adapted to stiff problems, of order at least 2. As each R_h problem is a Cauchy type problem, musltistep methods are not adapted and implicit Runge–Kutta methods can be used (see [9]). Here, we got the best performances with the RADAU5 method [9], both from the robustness point of vue and for the computing time.

The code is multi-threaded and well adapted to shared memory machines.

5. Numerical Results

5.1. **One dimensional simulations.** One dimensional simulations are a precious help for exploring the effect of constants in the model, avoiding biases due to the geometry. They are a necessity for the fitting of some parameters, such as diffusion coefficients. Actually, the values of diffusion coefficients that one can find in the literature [15] is varying by a factor of more than 10 ! In [7] the model and our code where used to explore the effect of the diffusion on the evolution of glutamate concentrations, allowing to find where, and in which cases, the diffusion introduces significant differences with the 0-dimensional model.

5.2. **Two dimensional simulations.** For these simulations, the finite difference mesh was modeled on a MNR image which results in a set of about $1.25\ 10^5$ nodes. We have used the Strang $R_{h/2}oD_hoR_{h/2}$ method, D_h being solved by the Crank-Nicolson method and $R_{h/2}$ by the RADAU5 method.

The unknowns are described in [7]. Let us recall that, in this modelisation the brain is divided in 3 components at small scale: the neurons medium, the glial cells and the extracellular domain. The unknowns are modelized at medium scale. The

proportion of neurons is ρ_n; the proportion of glial cells is ρ_a and $0 \leq \rho_n + \rho_a \leq 1$. ρ_n and ρ_a are among the unknowns of the problem. Other unknowns are the electric potential in neurons and astrocytes (V_n and V_a), and for each of the 3 components the concentration of the ions K^+, Na^+, Ca^{2+}, Cl^- and the glutamate glu^-.

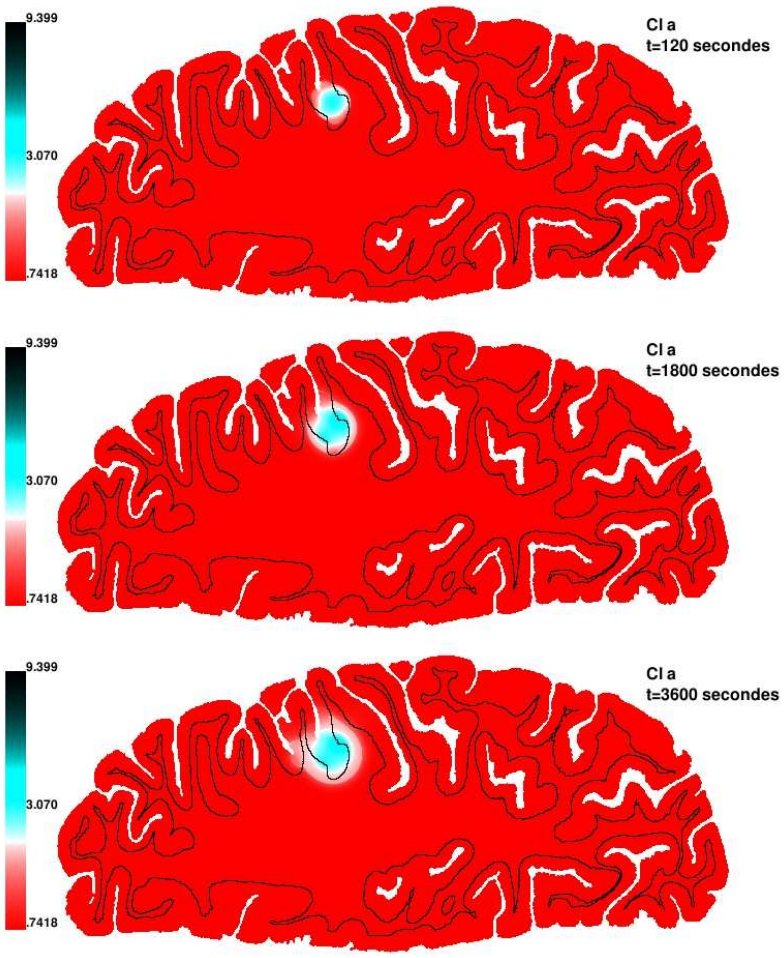


FIGURE 3. Evolution of the density of Cl^- ions in the astrocytes domain.

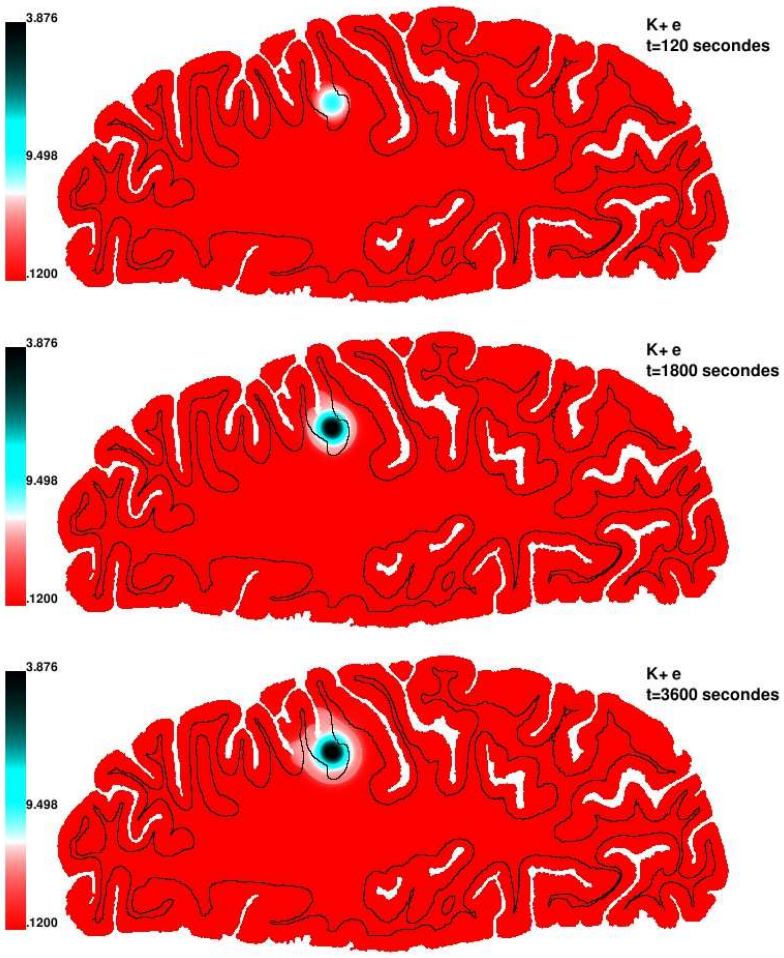


FIGURE 4. Evolution of the density of K^+ ions in the extra cellular domain.

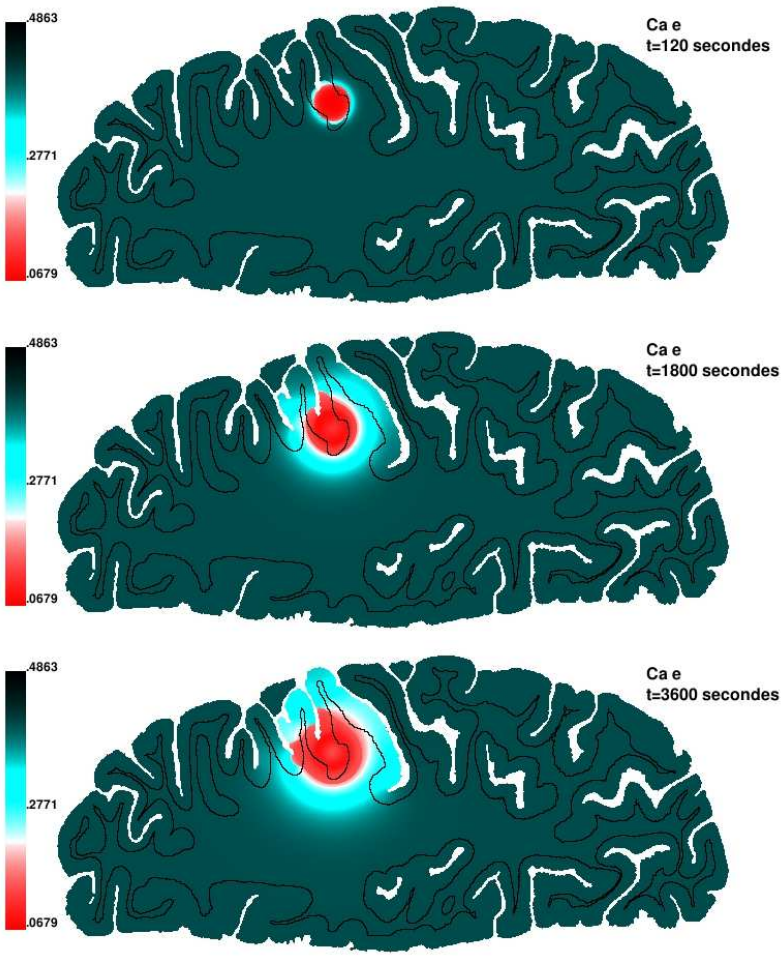


FIGURE 5. Evolution of the density of Ca^{2+} ions in the extra cellular domain.

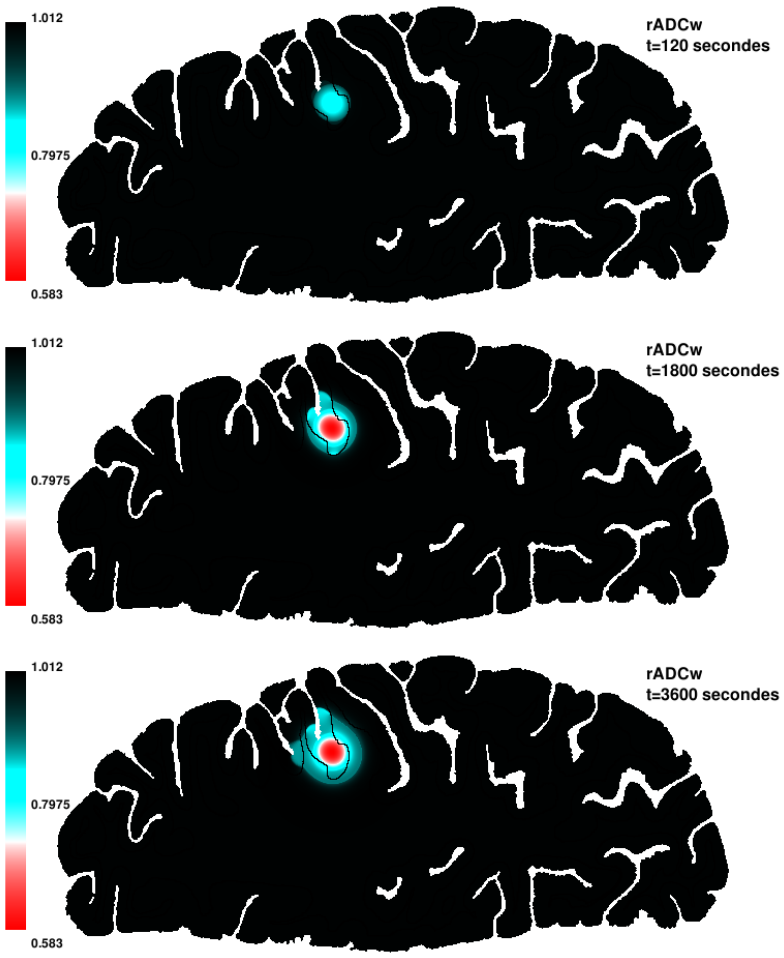


FIGURE 6. Evolution of the ratio of ADCW.

6. Perspectives: toward three dimensional more realistic simulations

Nowadays, the code [8] gives satisfactory results in dimension 2 (and 1). The computing time on a 4 core Opteron computer (2.6 Ghz) is about one hour for a simulation time of one hour. Three dimensional simulations will be much more demanding:

- the number of nodes in the discretization will grow up to more than 10^6, and thus, the computation time for the R_h problems will grow in the same manner,
- concerning the solution of the D_h problems , we have used a direct method [2] in dimension 2. In dimension 3, the amount of necessary memory would become too large[1] with the same method.

We are currently implementing the following strategy :

(1) try to determine in which parts of Ω the solution is "smooth" (that is to say that the gradients are small), and do not solve R_h on all the nodes in these parts; do it adaptively in time;
(2) use Domain Decomposition for the D_h problems: many iterative strategies are known, see for example [10], to turn the solution of D_h into iterative solutions on sub-domains, which can be computed in parallel.
(3) use also an adaptative discretization for D_h, on the sub-domains.

The code will then be really parallel and will be exploitable on distributed memory computers.

More realistic simulations also need a more elaborated model with certainly more unknowns.

7. Conclusion

We have obtained the first dimensional simulations of ischemic strokes, which, even if simplified, are sufficient enough to show that a useful numerical model can be, and will be obtained, even if a lot of work remains to be done. We want to emphasis that only the use of innovative and sophisticated numerical methods can lead to this result. Also, the experience accumulated in the numerical software will certainly be used in other fields.

References

[1] Georgios Akrivis, Michel Crouzeix, and Charalambos Makridakis. Implicit-explicit multistep methods for quasilinear parabolic equations. *Numer. Math.*, 82(4):521–541, 1999.
[2] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999.
[3] S. Descombes, T. Dumont, and M. Massot. Operator splitting for stiff nonlinear reaction-diffusion systems: order reduction and application to spiral waves. In *Patterns and waves (Saint Petersburg, 2002)*, pages 386–482. AkademPrint, St. Petersburg, 2003.
[4] Stéphane Descombes. Convergence of a splitting method of high order for reaction-diffusion systems. *Math. Comp.*, 70(236):1481–1501 (electronic), 2001.
[5] Stéphane Descombes and Marc Massot. Operator splitting for nonlinear reaction-diffusion systems with an entropic structure: singular perturbation and order reduction. *Numer. Math.*, 97(4):667–698, 2004.
[6] Marie-Aimee Dronne, J-P. Boissel, and Grenier E. A mathematical model of ion movements in grey matter during a stroke. *J. of Theoritical Biology*, 2006.

[1] we need about 5 Giga bytes in dimension 2

[7] Marie-Aimee Dronne, Grenier Emmanuel, Dumont Thierry, Hommel Marc, and Boissel Jean-Pierre. Role of astrocytes in grey matter during a stroke: A modelling approach. *Brain Research*, 1138:231–242, 2007.
[8] T. Dumont. Numerical software for reaction–diffusion systems. Source code and documentation at: `http://ciel.ccsd.cnrs.fr/`, 2007.
[9] E. Hairer and G. Wanner. *Solving ordinary differential equations. II*, volume 14 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1996. Stiff and differential-algebraic problems.
[10] L. Halpern. Absorbing boundary conditions and optimized Schwarz waveform relaxation. *BIT*, 46(suppl.):S21–S34, 2006.
[11] G. I. Marchuk. Splitting and alternating direction methods. In *Handbook of numerical analysis, Vol. I*, Handb. Numer. Anal., I, pages 197–462. North-Holland, Amsterdam, 1990.
[12] J. D. Murray. *Mathematical biology. I*, volume 17 of *Interdisciplinary Applied Mathematics*. Springer-Verlag, New York, third edition, 2002. An introduction.
[13] J. D. Murray. *Mathematical biology. II*, volume 18 of *Interdisciplinary Applied Mathematics*. Springer-Verlag, New York, third edition, 2003. Spatial models and biomedical applications.
[14] David L. Ropp and John N. Shadid. Stability of operator splitting methods for systems with indefinite operators: reaction-diffusion systems. *J. Comput. Phys.*, 203(2):449–466, 2005.
[15] B.E. Shapiro. Osmotic forces and gap junctions in spreading depression: a computational model. *J. Comput. Neurosci.*, 10:877–896, 2001.
[16] Gilbert Strang. On the construction and comparison of difference schemes. *SIAM J. Numer. Anal.*, 5:506–517, 1968.

Unité de Mathématiques Pures et Appliquées - CNRS UMR 5669 - Ecole Normale Supérieure de Lyon, 46, Allée d'Italie, 69364 LYON Cedex 07 France
E-mail address: `sdescomb@umpa.ens-lyon.fr`

Université de Lyon, Université Claude Bernard Lyon 1, Institut Camille Jordan and CNRS (UMR 5028) 43 boulevard du 11 novembre 1918 69622 Villeurbanne cedex France
E-mail address: `tdumont@math.univ-lyon1.fr`