A finite element solver for PDEs in MONOLIX

Raphaël Kuate, Marc Lavielle, Eric Blaudez, Kaelig Chatel, Jean-François Si Abdallah

A finite element solver for PDEs in MONOLIX

Raphaël Kuate, Marc Lavielle, Eric Blaudez, Kaelig Chatel, Jean-François Si Abdallah

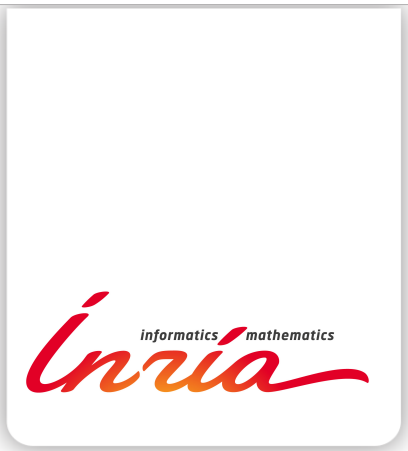
RESEARCH REPORT

N° 8717

March 2015

Project-Teams Popix

ISSN 0249-6399 ISRN INRIA/RR--8717--FR+ENG



A finite element solver for PDEs in Monolix

Raphaël Kuate*, Marc Lavielle†, Eric Blaudez‡, Kaelig Chatel§, Jean-François Si Abdallah¶

Project-Teams Popix

Research Report n° 8717 — March 2015 — 12 pages

Abstract:
We present a finite element method solver for partial derivatives equation in Monolix, a platform for population modeling of longitudinal data. We have implemented the well-known Lagrange finite element method in one, two and three dimensions of the space.

Key-words: finite elements, partial derivatives equations, solver, mesh, modeling, pharmacokinetics, pharmacodynamics, Monolix

* ✉: raphael.kuate@inria.fr
† ✉: marc.lavielle@inria.fr
‡ ✉: eric.blaudez@lixoft.net
§ ✉: kaelig.chatel@lixoft.net
¶ ✉: jean-francois.si_abdallah@lixoft.net

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Résolution numérique d'équations aux dérivées partielles par la méthode des éléments finis dans Monolix

Résumé : Ce document présente un solveur d'équations aux dérivées partielles utilisant la méthode des éléments finis, implémenté pour le logiciel Monolix.

Mots-clés : éléments finis, équations aux dérivées partielles, solveur, maillage, modélisation, pharmacocinétique, pharmacodynamique, Monolix

1 Introduction

MONOLIX (MOdèles NOn LInéaires à effets miXtes) [1, 19, 5, 25] is a platform of reference for model-based drug development. It combines the most advanced algorithms with unique ease of use. Pharmacometricians of preclinical and clinical groups can rely on MONOLIX for population analysis and to model pharmacokinetic/pharmacodynamic (PK/PD) and other complex biochemical and physiological processes [20, 18, 27]. MONOLIX is an easy, fast and powerful tool for parameter estimation in non-linear mixed effect models, model diagnosis and assessment, and advanced graphical representation.

The modeling of complex biological phenomenons with ordinary differential equations (ODEs) or with delay differential equations (DDEs) and tools available for parameter estimation [17, 20, 18, 27, 28, 4, 16] of these models are widely used in PK/PD modeling. However these ordinary differential equations neglect the spacial variation of the studied time varying phenomenons, thus the using of partial derivatives equations (PDEs) for the modeling of biological phenomenons [29, 33, 11, 31, 10, 32, 30, 2, 13] is more realistic when these phenomenons vary sensitively in the space; even if as consequence, this yields to more complex processes for the solution of these equations (PDE models), compared to ODEs/DDEs models. We have implemented a finite element solver for PDEs in MONOLIX, in order to use the powerful algorithms available in MONOLIX, for the estimation of parameters of models involving PDEs.

The document is organized as follows. In the second section we present shortly the main material used for PDEs. The third section is devoted to a description of the C++ code entries that must be provided by the user with the model, since the Mlxtran language used by MONOLIX dot not yet integrate a syntax for PDEs. In the last section, we present a three-dimensional example of estimation of parameters of a PDE model, computed with MONOLIX.

2 Material

We do not present the description of the finite element method itself [9, 12, 6, 23, 22], we give a brief list of the main material used.

2.1 The methods

Model The user specifies:

- The model in its strong form. Let u be the unknown function of the model. The model is given as a linear combination of the following operators: $\frac{\partial u}{\partial t},\ u,\ \nabla u,\ Div(u),\ \Delta u$.
- The finite element discretization space: P_0, P_1, P_2 or a combination of these spaces in the case of a system of PDEs.
- The boundary conditions: Dirichlet, Neumann, Fourrier-Robin.
- The initial condition.
- The source term or right-hand side function.

The weak (variational) from of the equations are then deduced form these informations. We use Lagrange finite element discretization.

Timestep In numerical simulation of time-dependent equations, numerical instabilities can occur, particularly for explicit numerical schemes [24, 14, 26, 3, 21]. We use an implicit time discretization of the equations and deduce the timestep with respect to the Courant-Friedrichs-Lewy [7, 8] condition.

2.2 The mesh

In the case of a mono-dimensional equation, the number of edges or the size of edges and the bounds of the mesh are given. For two-dimensional and three-dimensional meshes, a mesh file in *Freefem++* [15] format (.mesh) is given, other formats will be integrated in the future.

3 Use case

The model description and the tasks dedicated do PDEs are not yet included into the Mlxtran language. The user must then give informations to the software in an unusual format.

3.1 The project

We provide as examples three tests cases which are located in the folder Example = $HOME/lixoft/monolix/monolix433-pde/demos/pde, where one can find three projects:

- pde1D_project.mlxtran
- pde2D_project.mlxtran
- pde3D_project.mlxtran

The associated models can be found in the folder Example/libraryMLXTRAN. However these models do not really use the [EQUATION] section. These models can be run from MONOLIX only if one uses the wright C++ code that goes with, since it cannot yet be generated automatically by the software, the language Mlxtran used by MONOLIX being not ready for this. So, there exists three folders in the folder Example/monolixPdePlugins:

- test1D
- test2D
- test3D

Each of these directories contains two C++ files: PopModelDefCallbacks.cpp, PopModelDefCallbacks.h, which describe the pde solver calling by MONOLIX. Once the user opens the project with MONOLIX GUI, another GUI opens and he must select the C++ header and source files associated with the running project, example for the project pde1D_project.mlxtran, one must select monolixPdePlugins/test1D/PopModelDefCallbacks.h (header file) and PopModelDefCallbacks.cpp (source file) of the same directory.

3.2 Code associated with the project

We present some important details about the contains of the source file which must be associated with the project, we use for example the code associated with the project pde3D_project.mlxtran: Example/monolixPdePlugin/test3D/PopModelDefCallbacks.cpp, in a three-dimensional case.

```
⋮
//headers
#include "PopModelDefCallbacks.h"
⋮
//code prefix for  all non-local  functions (that must be called by the software engine)
/**** C3d053180824c4637a5ad04fe5de33750 ************************************************/

//parameters defined in the model: Example/libraryMLXTRAN/pde3D.txt,
```

```
//in the same order as given there
#define C_MLX indPar[0]
#define k_MLX indPar[1]
#define lambda_MLX indPar[2]
#define alpha_MLX indPar[3]
⋮
/****** Model functions, must be of the type pde::PdeFunc defined as:
 int Plugin_CALL C3d053180824c4637a5ad04fe5de33750NameOfTheFunction(double t, const  double* X,
                                          void * P,  double* Y)
 {
 // t: time
 // X: coordinates of the point in the space
 // Y: the output of the function evaluation.

   do some thing;
   return 0;
 }
 *********/
//example: define right-hand side function
int Plugin_CALL  C3d053180824c4637a5ad04fe5de33750f(double t, const  double*X, void * P,
                                                    double* Y)
{
    double x = X[0], y = X[1], z = X[2];
    Y[0] = 1.0e-5*t*std::abs((x+y-10*x*y*z -3.0e-8*t) *std::sin(6.3*x*y*z +3.2e-6*t+ y*x*y)*2 +
    1.6e-4*std::log(10.0+std::abs(100.0*x*y*y*x +2.0*x*y +44.0/std::abs(18.0 +z*x*y*y*x*x))));
    return 0;
}

//define boundary conditions
int Plugin_CALL C3d053180824c4637a5ad04fe5de33750Neumann1(double t, const  double*X,
                                           void * P, double* Y){    ...  }
int Plugin_CALL C3d053180824c4637a5ad04fe5de33750Neumann2(double t, const  double*X,
                                             void * P, double* Y){    ...  }
int Plugin_CALL C3d053180824c4637a5ad04fe5de33750Fourrier3(double t, const  double*X,
                                             void * P, double* Y){    ...  }
int Plugin_CALL C3d053180824c4637a5ad04fe5de33750Fourrier4(double t, const  double*X,
                                             void * P, double* Y){    ...  }

//initialization of the calculations engine
void C3d053180824c4637a5ad04fe5de33750Initialize(AbsSystem* system)
{
    AbsStatesIndiv* const systemAbsStates = system->asStatesIndividual();
    systemAbsStates->setNbStateParameters(0);
    AbsOdeIndiv* const systemAbsOde = system;
    systemAbsOde->setNbCompartments(5);
    systemAbsOde->loadOdeImplementation(&C3d053180824c4637a5ad04fe5de33750Open,
                                &C3d053180824c4637a5ad04fe5de33750ResetDynamics,
                                &C3d053180824c4637a5ad04fe5de33750HeedOccStart,
```

```
                                &C3d053180824c4637a5ad04fe5de33750HeedObservation,
                                &C3d053180824c4637a5ad04fe5de33750HeedRegUpdate,
                                &C3d053180824c4637a5ad04fe5de33750HeedDose,
                                &C3d053180824c4637a5ad04fe5de33750EndLag,
                                &C3d053180824c4637a5ad04fe5de33750EndAbsorption);
//sensors definition
double sensors[15]; //15 = number of sensors(5) * dimension of the space
sensors[0] = 0.0; sensors[1] = 0.1; sensors[2] = 0.8;//coordinates x,y,z of each sensor
sensors[3] = 0.5; sensors[4] = 0.6; sensors[5] = 0.41;
sensors[6] = 0.4; sensors[7] = 0.19; sensors[8] = 0.7;
sensors[9] = 0.8; sensors[10] = 0.91; sensors[11] = 0.1;
sensors[12] = 0.92; sensors[13] = 0.41; sensors[14] = 0.21;
//provide the mesh file, example on linux:
std::string meshFile = "/home/lixoft/lixoft/monolix/monolix433-pde/
demos/pde/monolixPdePlugins/resources/Cube1.mesh";

//initialization of the pde solver
/***
2D  and 3D cases:
pdeSetSolver(const char* meshFile, int dimension, double t0, double tf,
             const double*  sensors, int sizeSensors, pde::Solver solverType);

1D  case:
pdeSetSolver1dNp(double x0, double xL, int nbpoints, double t0, double tf,
                 double* sensors, int sizeSensors,  pde::Solver solverType);
pdeSetSolver1dDx(double x0, double xL, double dx, double t0, double  tf,
                 const double* sensors, int sizeSensors, pde::Solver solverType);

meshFile: name (absolute path) of  the mesh file
dimension: dimension of the space(1,2 or 3)
[t0, tf]: simulation time interval
sensors: table of sensors coordinates
sizeSensors: the size of the table sensors
solverType: linear system resolution method (LU,CG(conjugate gradient),
                                             BiCG(bi-conjugate gradient))
[x0, xL]: 1D domain
nbpoints: number of nodes of the 1D mesh
dx: size of the edges of the 1D mesh
 Note: the interpolation space  is fixed for now to piecewise linear finite
       elements, some other solver features such as tolerance are fixed.
 ***/

systemAbsOde->pdeSetSolver(meshFile.c_str(),3,0.0,480.*360,sensors,15,pde::LU);
//define some constant functions
double Dirichlet5 = 16.0;
double Dirichlet6 = 18.0;
double g0 = 16.5;

//set boundary conditions
```

```
    /***
    pdeSetBoundaryConditionsFunc(pde::BoundType num, pde::PdeFunc funy0, int label,
                                 double coeff=1.0);
    pdeSetBoundaryConditionsScal(pde::BoundType num, double y0, int label, double coeff=1.0);

    num: the type of boundary condition (DIRICHLET, NEUMANN, FOURRIER)
    funy0, y0: boundary condition
    label: a number specifying on which boundary of the mesh the  condition must be  applied.
           This number must be available in the mesh file as label of boundary elements.
    coeff: the exchange coefficient, for example heat exchange (Fourrier conditions) between
           the unknown u and the function h on the boundary Γ: ∂ u / ∂ n_Γ  = coeff*(h − u)

    ***/
    systemAbsOde->pdeSetBoundaryConditionsFunc(pde::FOURRIER,
                                        &C3d053180824c4637a5ad04fe5de33750Fourrier3,3);
    systemAbsOde->pdeSetBoundaryConditionsFunc(pde::FOURRIER,
                                        &C3d053180824c4637a5ad04fe5de33750Fourrier4,4);
    systemAbsOde->pdeSetBoundaryConditionsFunc(pde::NEUMANN,
                                        &C3d053180824c4637a5ad04fe5de33750Neumann1,1);
    systemAbsOde->pdeSetBoundaryConditionsFunc(pde::NEUMANN,
                                        &C3d053180824c4637a5ad04fe5de33750Neumann2,2);
    systemAbsOde->pdeSetBoundaryConditionsScal(pde::DIRICHLET,Dirichlet5,5);
    systemAbsOde->pdeSetBoundaryConditionsScal(pde::DIRICHLET,Dirichlet6,6);

    //set right hand side function
    /***
    pdeSetRhsFunc(pde::PdeFunc funy0);
    pdeSetRhsScal(double y0) ;
    ***/
    systemAbsOde->pdeSetRhsFunc(&C3d053180824c4637a5ad04fe5de33750f);

    //set initial condition
    /***
    pdeSetInitValueFunc(pde::PdeFunc funy0);
    pdeSetInitValueScal(double y0);
    ***/
    systemAbsOde->pdeSetInitValueScal(g0);
}

//update the model parameters on each call of Monolix
void C3d053180824c4637a5ad04fe5de33750Open(AbsSystem* system)
{
    AbsStatesIndiv* const systemAbsStates = system->asStatesIndividual();
    double* const customState = systemAbsStates->stateParametersBegin();
    AbsOdeIndiv* const systemAbsOde = system;
    double t0_MLX = systemAbsOde->lastQuiescTime();
    double* const odeIni = systemAbsOde->lastQuiescComptBegin();
    const double* const indPar = systemAbsStates->occasionParametersBegin();
```

```
    t0_MLX = 0.0;
    systemAbsOde->startQuiescency(t0_MLX);

   //set new values of  the model parameters into the pde solver
   /***
   pdeSetModelCoeffGrad(double a0, double a1, double a2, const double* a3)
   for the model: a0*∂ u/∂ t - a1*Δ u + a2*u + a3· ∇ u = f
   pdeSetModelCoeffDiv(double a0, double a1, double a2, double a4)
   for the model: a0*∂ u/∂ t - a1*Δ u + a2*u + a4*Div(u) = f
   ***/
    systemAbsOde->pdeSetModelCoeffDiv(C_MLX,k_MLX,lambda_MLX,alpha_MLX);
}
⋮
#undef C_MLX
#undef k_MLX
#undef lambda_MLX
#undef alpha_MLX
⋮
```

4 Estimation of parameters of PDEs using Monolix

Let Ω be a tri-dimensional domain, and let $\Gamma_{\mathcal{D}} \cup \Gamma_{\mathcal{F}} \cup \Gamma_N$ be the boundary of Ω. Denote n_X the outer normal to the boundary of Ω on the bound X. The pde model used is a parabolic heat equation, with Dirichlet, Neumann and Fourrier-Robin boundary conditions, described as follows:

$$\begin{cases} C\dfrac{\partial u}{\partial t} - k\Delta u + \lambda u + \alpha \cdot \nabla u = & f, \text{ on } \Omega \times [0,T], \\ \dfrac{\partial u}{\partial n_{\Gamma_N}} = & g, \text{ on } \Gamma_N \times [0,T], \\ \dfrac{\partial u}{\partial n_{\Gamma_{\mathcal{F}}}} = & h - u, \text{ on } \Gamma_{\mathcal{F}} \times [0,T], \\ u = & u_{\mathcal{D}}, \text{ on } \Gamma_{\mathcal{D}} \times [0,T], \\ u(.,0) = & u_0 \text{ on } \Omega, \text{ at } t = 0. \end{cases} \tag{4.1}$$

Functions $f, g, h, u_0, u_{\mathcal{D}}$ have been chosen such that the solution u of the model solved with piecewise linear Lagrange finite element varies between 5 and 30 over the domain Ω and the time interval $[0,T]$, figure 1.

4.1 pde3D_project.mlxtran

Data The data used for the estimations of parameters C, k, λ, α of model (4.1) are simulated for 15 individuals, using the same model with the following values of the parameters:

$$C_{\text{pop}} = 5.0 \times 10^5, \ k_{\text{pop}} = 10^{-3}, \ \lambda_{\text{pop}} = 1, \ \alpha = 10^{-4} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

The data have been obtained from the solution u at five sensors, where the parameters are log-normally distributed:

$$a = a_{\text{pop}} e^{\eta}, \quad \eta \sim \mathcal{N}(0, \omega_a^2), \ \omega_a = 0.2, \ a = C, k, \lambda.$$

For each individual we use five measurements in space (taken on sensors) and 480 records in time for each sensor. The initial values of the parameters for the estimation algorithm are:

$$C_0 = 3.5 \times 10^5, \ k_0 = 1.5 \times 10^{-3}, \ \lambda_0 = 1.25.$$

The initial standard deviation of the random effects is $\omega = 0.5$ for each parameter.

Parameter estimation The results of the estimation of these parameters with MONOLIX in are presented the following. The mesh used has 64 vertices and 162 tetraheda. The estimation algorithm used about 400 iterations, for a population of 15 individuals.

```
****************************************************************
*      pde3D_project.mlxtran
*      November 24, 2014 at 12:52:26
*      Monolix version: 4.3.2
****************************************************************

Estimation of the population parameters

                parameter
C_pop         :  5.66e+05
k_pop         :   0.00111
lambda_pop    :     0.936
alpha_pop     :    0.0001

omega_C       :      0.32
omega_k       :     0.156
omega_lambda  :     0.228
omega_alpha   :         0

Population parameters estimation...

Elapsed time is 4.55e+04 seconds.
CPU time is 2.29e+04 seconds.

Individual parameters estimation algorithm...
  Elapsed time is 155 seconds.
  CPU time is 155 seconds.
```

pde2D_project.mlxtran In this project, we compute the estimation of parameters C, k, λ, α of model (4.1) in a bi-dimensional case. The data protocol is the same as done for the previous project.

pde1D_project.mlxtran In this project, we compute the estimation of parameters k, λ, α of the following model,

$$\frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial x^2} + \lambda u + \alpha \frac{\partial u}{\partial x} = 0.$$

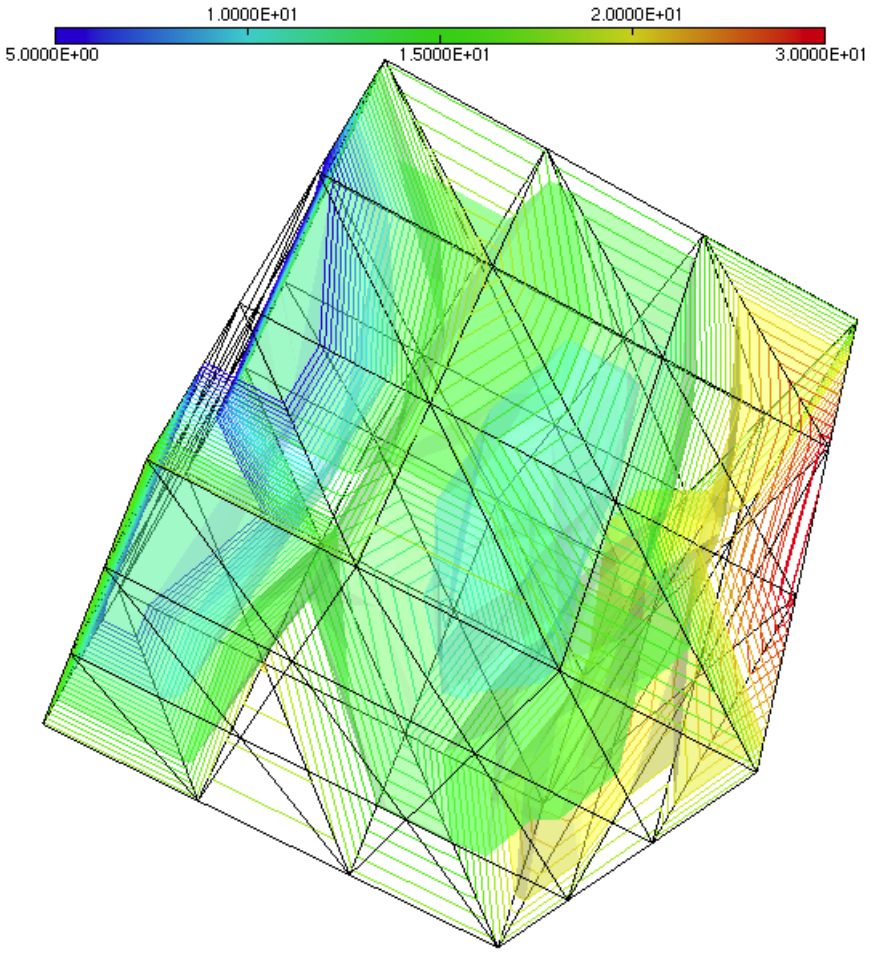


Figure 1: Iso-lines and iso-surfaces of the finite element solution of the model (4.1) at time $t = T$. The domain $\Omega = [0, 1] \times [0, 1] \times [0, 1]$ is a cube, the time $t \in [0, T]$, $T = 48 \times 3600s$. The mesh has 64 vertices and 162 tetraheda.

The parameters, the initial condition and the boundary conditions have been chosen as described in [33, §4.2], with the same data generation protocol.

References

[1] Monolix *A software for the analysis of nonlinear mixed effects models.* http://www.lixoft.eu/products/monolix/documentation/.

[2] JessieL.-S. Au, Peng Guo, Yue Gao, Ze Lu, MichaelG. Wientjes, Max Tsai, and M.Guillaume Wientjes. Multiscale tumor spatiokinetic model for intraperitoneal therapy. *The AAPS Journal*, 16(3):424–439, 2014.

[3] Christopher TH Baker and Christopher AH Paul. Computing stability regions-Runge-Kutta methods for delay differential equations. *IMA Journal of Numerical Analysis*, 14(3):347–362, 1994.

[4] C.T.H. Baker, G.A. Bocharov, and C.A.H. Paul. Mathematical modelling of the interleukin-2 T-cell system: A comparative study of approaches based on ordinary and delay differential equations. *Journal of Theoretical Medicine*, 1(2):117–128, 1997.

[5] PhylindaL.S. Chan, Philippe Jacqmin, Marc Lavielle, Lynn McFadyen, and Barry Weatherley. The use of the SAEM algorithm in MONOLIX software for estimation of population pharmacokinetic-pharmacodynamic-viral dynamics parameters of maraviroc in asymptomatic HIV subjects. *Journal of Pharmacokinetics and Pharmacodynamics*, 38(1):41–61, 2011.

[6] Philippe G. Ciarlet. *Finite Element Method for Elliptic Problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[7] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100(1):32–74, 1928.

[8] R. Courant, K. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM J. Res. Dev.*, 11(2):215–234, March 1967.

[9] A. Ern and J.-L. Guermond. *Theory and practice of finite elements*, volume 159 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2004.

[10] Hermann B. Frieboes, Mary E. Edgerton, John P. Fruehauf, Felicity R.A.J. Rose, Lisa K. Worrall, Robert A. Gatenby, Mauro Ferrari, and Vittorio Cristini. Prediction of drug response in breast cancer using integrative experimental/computational modeling. *Cancer Research*, 69(10):4484–4492, 2009.

[11] Jana L. Gevertz. Computational modeling of tumor response to vascular-targeting therapie;part i: Validation. *Computational and Mathematical Methods in Medicine*, 2011.

[12] Vivette Girault and Pierre-Arnaud Raviart. *Finite element methods for Navier-Stokes equations : theory and algorithms*. Springer series in computational mathematics. Springer-Verlag, Berlin, New York, 1986. Extended version of : Finite element approximation of the Navier-Stokes equations.

[13] Emmanuel Grenier, Violaine Louvet, and Paul Vigneaux. Parameter estimation in non-linear mixed effects models with SAEM algorithm: extension from ODE to PDE. Research Report RR-8231, February 2013.

[14] Gopal K. Gupta, Ron Sacks-Davis, and Peter E. Tescher. A review of recent developments in solving ODEs. *ACM Comput. Surv.*, 17(1):5–47, March 1985.

[15] F. Hecht. New development in freefem++. *J. Numer. Math.*, 20(3-4):251–265, 2012.

[16] Raphaël Kuate, Marc Lavielle, Eric Blaudez, Kaelig Chatel, Jerome Marquet, and Jean-François Si Abdallah. A delay differential equation solver for MONOLIX & MLXPLORE. Research Report RR-8489, February 2014.

[17] Marc Lavielle. *Mixed effects models for the population approach: models, tasks, methods and tools*. Chapman & Hall/CRC Biostatistics Series. Taylor and Francis, Hoboken, NJ, 2014.

[18] Marc Lavielle and Cyprien Mbogning. An improved SAEM algorithm for maximum likelihood estimation in mixtures of non linear mixed effects models. *Statistics and Computing*, pages 1–15, 2013.

[19] Marc Lavielle and France Mentré. Estimation of population pharmacokinetic parameters of saquinavir in HIV patients with the MONOLIX software. *Journal of Pharmacokinetics and Pharmacodynamics*, 34(2):229–249, 2007.

[20] David Makowski and Marc Lavielle. Using SAEM to estimate parameters of models of response to applied fertilizer. *Journal of Agricultural, Biological, and Environmental Statistics*, 11(1):45–60, 2006.

[21] Linda R. Petzold, Laurent O. Jay, and Jeng Yen. Numerical solution of highly oscillatory ordinary differential equations. *Acta Numerica*, 6:437–483, 1 1997.

[22] Olivier Pironneau. *Finite element methods for fluids*. Masson Chichester, Paris, 1989. Ouvrage paru en français dans la coll. Recherches en mathématiques appliquées.

[23] A. Quarteroni, R. Sacco, and F. Saleri. *Méthodes Numériques: Algorithmes, analyse et applications (French Edition)*. Springer, 1 edition, September 2007.

[24] E.H Rogers. Stability and convergence of approximation schemes. *Journal of Mathematical Analysis and Applications*, 20(3):442 – 453, 1967.

[25] S Ruiz, JM Conil, B Georges, F Ravat, T Seguin, P Letocart, O Fourcade, and S Saivin. Ceftazidime dosage regimen recommendations in burn patients based on a monolix population pharmacokinetic study. *Critical Care*, 16(1):1–189, 2012.

[26] J.M. Sanz-Serna and M.N. Spijker. Regions of stability, equivalence theorems and the Courant-Friedrichs-Lewy condition. *Numerische Mathematik*, 49(2-3):319–329, 1986.

[27] Radojka M. Savic, France Mentré, and Marc Lavielle. Implementation and evaluation of the SAEM algorithm for longitudinal ordered categorical data with an illustration in pharmacokinetics-pharmacodynamics. *The AAPS Journal*, 13(1):44–53, 2011.

[28] J Srividhya, MS Gopinathan, and Santiago Schnell. The effects of time delays in a phosphorylation–dephosphorylation pathway. *Biophysical chemistry*, 125(2):286–297, 2007.

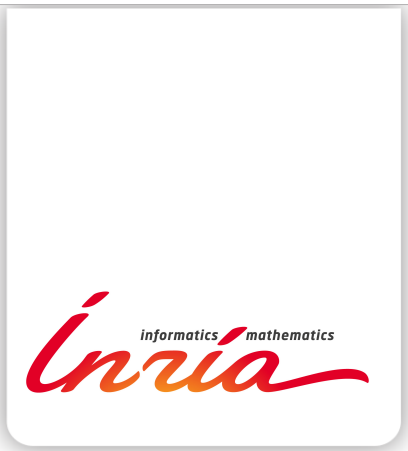
[29] S.M. Wise, J.S. Lowengrub, and V. Cristini. An adaptive multigrid algorithm for simulating solid tumor growth using mixture models. *Mathematical and Computer Modelling*, 53(1-2):1 – 20, 2011.

[30] S.M. Wise, J.S. Lowengrub, H.B. Frieboes, and V. Cristini. Three-dimensional multispecies nonlinear tumor growth: Model and numerical method. *Journal of Theoretical Biology*, 253(3):524 – 543, 2008.

[31] Steven Wise, Junseok Kim, and John Lowengrub. Solving the regularized, strongly anisotropic cahn-hilliard equation by an adaptive nonlinear multigrid method. *Journal of Computational Physics*, 226(1):414 – 446, 2007.

[32] Min Wu, Hermann B. Frieboes, Steven R. McDougall, Mark A.J. Chaplain, Vittorio Cristini, and John Lowengrub. The effect of interstitial pressure on tumor growth: Coupling with the blood and lymphatic vascular systems. *Journal of Theoretical Biology*, 320(0):131 – 151, 2013.

[33] Xiaolei Xun, Jiguo Cao, Bani Mallick, Arnab Maity, and Raymond J. Carroll. Parameter estimation of partial differential equation models. *Journal of the American Statistical Association*, 108(503):1009–1020, 2013.



RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399