HAL
open science

Parameter estimation in non-linear mixed effects models with SAEM algorithm: extension from ODE to PDE

Emmanuel Grenier, Violaine Louvet, Paul Vigneaux

To cite this version:

Emmanuel Grenier, Violaine Louvet, Paul Vigneaux. Parameter estimation in non-linear mixed effects models with SAEM algorithm: extension from ODE to PDE. [Research Report] RR-8231, INRIA. 2013, pp.30. hal-00789135

HAL Id: hal-00789135

https://inria.hal.science/hal-00789135v1

Submitted on 19 Feb 2013

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.

Parameter estimation in non-linear mixed effects models with SAEM algorithm: extension from ODE to PDE

E. Grenier, V. Louvet, P. Vigneaux

RESEARCH REPORT
N° 8231
February 2013
Project-Team Numed

ISSN 0249-6399 ISRN INRIA/RR--8231--FR+ENG

Parameter estimation in non-linear mixed effects models with SAEM algorithm: extension from ODE to PDE

E. Grenier*, V. Louvet†, P. Vigneaux*

Project-Team Numed

Research Report n° 8231 — February 2013 — 30 pages

Abstract: Parameter estimation in non linear mixed effects models requires a large number of evaluations of the model to study. For ordinary differential equations, the overall computation time remains reasonable. However when the model itself is complex (for instance when it is a set of partial differential equations) it may be time consuming to evaluate it for a single set of parameters. The procedures of populational parametrization (for instance using SAEM algorithms) are then very long and in some cases impossible to do within a reasonable time. We propose here a very simple methodology which may accelerate populational parametrization of complex models, including partial differential equations models. We illustrate our method on the classical KPP equation.

Key-words: Parameter estimation, SAEM algorithm, Partial differential equations, KPP equation

* U.M.P.A., Ecole Normale Supérieure de Lyon, CNRS UMR 5669 & INRIA, Project-team NUMED
† Institut Camille Jordan, CNRS UMR 5208 & Université Lyon 1 & INRIA, Project-team NUMED

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Parameter estimation in non-linear mixed effects models with SAEM algorithm: extension from ODE to PDE

Résumé : L'estimation de paramètres dans des modèles à effets mixtes demande un très grand nombre d'évaluations du modèle à étudier. Pour des équations différentielles ordinaires, le temps de calcul total reste raisonnable. Toutefois, lorsque le modèle est plus complexe (par exemple lorsqu'il s'agit d'équations aux dérivées partielles), son évaluation peut être assez longue. Les techniques d'estimation de paramètres populationnelles (comme l'algorithme SAEM) sont alors très chères en temps de calcul, voire même impossibles à réaliser en temps raisonnable. Nous proposons dans cet article une méthode très simple pour accélérer l'estimation de paramètres populationnelle pour des modèles complexes, en particulier à base d'EDP. Nous illustrons cette méthode sur l'équation de KPP classique pour modéliser les problèmes de réaction-diffusion.

Mots-clés : Parameter estimation, SAEM algorithm, Partial differential equations, KPP equation

Introduction

A crucial step in the validation of a model is of course to compare it with real world data. This is usually done by using nonlinear regression techniques in the case of individual data, or by statistical approachs (for instance using a SAEM algorithm [6], [1]) in the case of populational data. In both cases, this requires a large number of evaluations of the model, for a large number of different sets of parameters. The evaluation time for a single set of parameters may be very long, in particular when partial differential equations are involved (it may go up to a few minutes, or a few hours, or even days). In this case, nonlinear regression algorithms or populational approaches can not be done within a reasonable time. It is therefore crucial to find methods to accelerate them.

One of the ideas to speed them up is to use the fact that, often, these procedures will evaluate the model for *nearby* sets of parameters. Namely, for nonlinear regression procedures, or for populational approaches, the sets of parameters will hopefully tend to the minimal set. Hence as the algorithm goes on, the distance between two successive sets of parameters goes to zero. In a populational approach, if the standard deviations of the individual conditional distributions are not too large, most of the sets of parameters will be in the same small region.

A natural idea arises: to speed up such procedures, it could be interesting to use already computed values of the model. Using interpolation methods, which are very fast, approximate values of the model can be inferred quickly, without new time consuming evaluations. If the approximation seems good enough, then an approximation of the model through interpolation may be enough. Else a time consuming evaluation of the complete model appears to be necessary.

If we go on with this idea of interpolation of already computed values of the model, two strategies appear

- *once for all precomputation:* the model is computed on a mesh before parameter identification procedure. Assuming that the parameters lie in a domain Ω, we compute the model for some points $P_i \in \Omega$. During the parameter identification procedure, we simply approximate the complex model by interpolation of the values at points P_i, which is very fast. The identification procedure can be done very quickly.

  The precomputation can be done on arbitrary meshes, or on structured meshes. Interpolation is easier to do on structured meshes, hence in this paper we illustrate this approach using a cubic structured mesh. However similar ideas may be applied to different structured or even non structured meshes, up to technical complications in the interpolation routines.

  The mesh can be *a priori* fixed (uniform mesh), or *adaptively* created. It is more precise and computationally more efficient to refine the mesh where the model has large changes, hence it is better to refine the mesh non uniformly, during the computation. Mesh refinement will be done according to some criterion or score, which must be carefully chosen.

- *interactive tabulation:* during the identification procedure, we interpolate the model using previously estimated sets of values. If the interpolation quality is too bad, we compute the precise value of the model at this set of parameters (which is time consuming but more precise). With this approach we can play with the various errors (interpolation error, numerical error, populational algorithm error).

  For instance when the populational algorithm converges, it tends to evaluate the model according to a Gaussian distribution. The model needs to be accurately evaluated at the centre of the Gaussian, but large errors may be acceptable in the Gaussian tail. Therefore it is interesting to refine the mesh along with the populational algorithm, near the minimum.

A natural idea is then to run populational algorithm for a while, then switch to precomputation refinement in the areas which are explored by the populational algorithm, and iterate. The complete study of the coupling between mesh approximation and populational algorithm is a delicate issue and will be detailed in a forthcoming work.

We note that the ideas developed in this paper are close to the methodologies of model order reduction (MOR) for which there exist a huge literature. For an overview of the various methods, we refer to the book [7] and the special issue [3], as well as references therein. More particularly on statistical inverse problems, from which we based our approach to estimate the parameters, we refer to the book [4].

Let us precise the applications we have in mind. We suppose that we have two or three dimensional images from a time varying phenomena. We do not want to tackle the whole complexity of the images. Instead we want to work with scalar data extracted from these images. For instance, we can think about the volume of a tumour extracted from an MRI image.

However we do not want to reduce to a model based on an ordinary differential equation, but instead we want to work with an underlying partial differential equation model, in order to keep trace of spatial effects (geometries of the solution and of the domain). Therefore the output of our model will be spatially averaged quantities of solutions of partial differential equations posed in bounded domains. For the tumour example above, the PDE can be the classical KPP equation, and the tumour volume is the integral of the tumoral concentration. This parameter dependent output is therefore a time sequence of scalars. One of the advantages is that the small size of the output avoids to deal with delicate storage problems.

Our strategy is to speed up the computation associated to the evaluation of the scalar time series associated to the solution of the full PDE. This is where the above description of precomputation philosophy comes into play. We couple a SAEM algorithm with evaluation of the model through interpolation on a precomputed mesh of the parameters domain.

This appears to be already very efficient on our tumour application. Namely we try to identify initial position, reaction coefficient and diffusion coefficient of a KPP reaction diffusion equation, using temporal series of measures of the integral of the solution of KPP equation (see section 4 for more details). Note that even though our model is a fully non linear partial differential equation, we base the parameters estimation on observations which are spatial integrals (i.e. scalar observations) of the solutions.
Typically, the SAEM algorithm requires around one million evaluations of KPP. On the one hand, a full computation of a KPP solution requires on average around 2 seconds. Therefore a simple run of SAEM for KPP lasts around 23 days 3 hours, and is not easily parallelized. On the other hand, in our present approach, the precomputation of the parameters' space mesh requires arount 1000 evaluations of KPP, which leads to a 40 minutes total time for the offline step. On 16 processors, it takes around 2mn30s. Then, in the online step of SAEM with interpolation, the computational time is approximately 8mn: the complete problem is therefore tractable (within a time which is of the order of the computational time for the ODE case), with a speed up of order 3000.

In forthcoming works we would like to combine this simple approach with more refined model order reduction techniques, in order to deal with image valued models, or models with a large number of parameters.

This paper is organised as follows. Sections 1 and 2 are devoted to the description of the SAEM algorithm and precomputations. Section 3 details the coupling between these two algorithms, and Section 4 is devoted to the application of this strategy for parameter estimations for KPP model, a classical simple model of reaction-diffusion wave propagation.

1 SAEM algorithm

1.1 Principle of populational approaches

Let us first recall the principles underlying populational approaches. The main idea is the following: instead of trying to fit each individual set of data, namely to find individual parameters, we look for the distributions of the individual parameters at the populational level.

More precisely, let us consider a model

$$y = \phi(t, Z)$$

where y is the observable, t the time, and Z the individual parameters. The model ϕ may be algebraic, may be a set of ordinary differential equations, or of partial differential equations.

Of course neither the models nor the measures of y are exact, and random errors should be added. Let us assume for simplicity that they follow a normal distribution law, with mean 0 and standard deviation ε_σ.

Let us consider N individuals and for each individuals, measures of y at times t_{ij} ($1 \leq i \leq N$, $1 \leq j \leq N_i$, N_i being the numbers of measures for the individual number i). We get measures y_{ij} such that

$$y_{ij} = \phi(t_{ij}, Z_i) + \varepsilon_{ij},$$

where

$$\varepsilon_{ij} \sim \mathcal{N}(0, \varepsilon_\sigma).$$

Let us consider a fixed individual $1 \leq i \leq N$. The probability of observing $(y_{ij})_{1 \leq j \leq N_i}$ knowing its individual set of parameters Z_i is

$$P\Big((y_{ij})|Z_i\Big) = \frac{1}{\sqrt{2\pi\varepsilon_\sigma}^{N_i}} \Pi_{j=1}^{N_i} e^{-\frac{(y_{ij}-\phi(t_{ij},Z_i))^2}{\varepsilon_\sigma}}.$$

If we look at this expression as a function of Z_i (the observations (y_{ij}) being given), we get the likelihood of Z_i (for a given i)

$$L(Z_i) = p\Big((y_{ij})|Z_i\Big).$$

The best fit Z_i^b

$$Z_i^b = \text{argmax } L(Z_i)$$

is then solution of the classical nonlinear regression problem

$$Z_i^b = \text{argmax } \sum_{j=1}^{N_i} e^{-(y_{ij}-\phi(t_{ij},Z_i))^2}.$$

However in many case, few data per individual are available, and the non linear regression problem can not be solved. In these cases, it is interesting to gather all the data and to follow a global populational approach.

The principle is to assume that the individual characteristics follow a (say) normal distribution, and to look for the average and standard deviation of each characteristic. Of course non normal laws can be treated in the same way. More precisely we assume that

$$Z_i \sim \mathcal{N}(\theta_m, \theta_\sigma)$$

where θ_m is the mean of the individual parameters θ_i in the population, and θ_σ their standard deviation. Let

$$\theta_{pop} = (\theta_m, \theta_\sigma)$$

be the set of all populational characteristics.

In a populational approach, we try to identify θ_m and θ_σ using the data of all the individuals. The ratio number of data / number of unknown parameters in then far better.

Let us now detail the populational approach. The probability that the individual i has characteristics Z_i is

$$P\Big(Z_i|\theta_{pop}\Big) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(Z_i-\theta_m)^2}{\theta_\sigma}},$$

hence the probability for the individual i to have characteristics Z_i and data (y_{ij}) is

$$P\Big((y_{ij})_j, Z_i|\theta_{pop}\Big) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(Z_i-\theta_m)^2}{\theta_\sigma}} \frac{1}{\sqrt{2\pi\varepsilon_\sigma}^{N_i}} \Pi_{j=1}^{N_i} e^{-\frac{(y_{ij}-\phi(t_{ij},Z_i))^2}{\varepsilon_\sigma}}.$$

At the population level, the probability to observe individuals with characteristics $(Z_i)_{1\leq i\leq N}$ and data $(y_{ij})_{ij}$ is then

$$P\Big((y_{ij})_{ij}, (Z_i)_i|Z_{pop}\Big) = \Pi_{i=1}^{N} \frac{1}{\sqrt{2\pi}} e^{-\frac{(Z_i-\theta_m)^2}{\theta_\sigma}} \frac{1}{\sqrt{2\pi\varepsilon_\sigma}^{N_i}} \Pi_{j=1}^{N_i} e^{-\frac{(y_{ij}-\phi(t_{ij},Z_i))^2}{\varepsilon_\sigma}}.$$

But $(Z_i)_i$ are not observed ("hidden variables"), therefore the probability of observing $((y_{ij})_{ij})$ knowing θ_{pop} is

$$P\Big((y_{ij})_{ij}|\theta_{pop}\Big) = \int P\Big((y_{ij})_{ij}, (Z_i)_i|\theta_{pop}\Big) dZ_1...dZ_n.$$

The likelihood of θ_{pop} is then

$$L(\theta_{pop}) = P\Big((y_{ij})_{ij}|\theta_{pop}\Big).$$

The usual approach leads to the search of $\theta^\star_{pop}$ which maximizes L.

$$\theta^\star_{pop} = \text{argmax}_\theta L(\theta). \tag{1}$$

1.2 SAEM algorithm

To solve (1) is a difficult problem, since it combines two complications: we have to optimize a nonlinear function, and this function is a multidimensional integral. The idea of SAEM algorithm (Stochastic Approximation Expectation Maximization algorithm) is to introduce a nearby problem which splits these two difficulties.

Namely instead of trying to maximize L we focus on

$$Q(\theta|\theta') = \int \log P\Big((y_{ij})_{ij}, (Z_i)_i|\theta\Big) P\Big((Z_i)_i|\theta'\Big) dZ_1...dZ_N \tag{2}$$

where

$$P\Big((Z_i)_i|\theta\Big) = \frac{P\Big((y_{ij})_{ij}, (Z_i)_i|\theta\Big)}{g(\theta)},$$

$g(\theta)$ being the normalization factor

$$g(\theta) = \int P\Big((y_{ij})_{ij}, (Z_i)_i|\theta\Big) dZ.$$

Problem

$$\theta^+_{pop} = \text{argmax}_\theta \; Q(\theta|\theta) \tag{3}$$

is then an approximate version of (1). However it is much simpler to address since it is natural to introduce the following optimization algorithm

$$\theta_{k+1} = \text{argmax}_\theta \; Q(\theta|\theta_k). \tag{4}$$

As we will see, (4) appears to be very easy to solve. It remains to compute $Q(\theta|\theta_k)$. For this we use a Monte Carlo Markov Chain (MCMC) approach in order to get a sequence $(Y_l)_l$ of points, with distribution law $g(\theta_k)$. This is easily done since the computation of $P((y_{ij}),(Z_i)_i|\theta')$ is explicit. Note that Y_l is a population and is composed of N individuals Y_{li}.

We then approximate $Q(\theta|\theta')$ by

$$Q'(\theta|\theta_l) = -\frac{N}{2}\log\pi - \frac{\sum_i N_i}{2}\log(\pi\varepsilon_\sigma)$$

$$-\sum_l \sum_{1\le i\le N} \frac{(Y_{l,i}-\theta_m)^2}{\theta_\sigma} - \sum_l \sum_{1\le i\le N} \sum_{1\le j\le N_i} \frac{(y_{ij}-\phi(t_{ij},Y_{li}))^2}{\varepsilon_\sigma}.$$

Note that the opimization step (4) is explicit: θ_{k+1} is explicit using the expression of Q'.

SAEM appears to be very efficient and is widely used in applied and industrial problems, in particular in pharmacokinetics pharmacodynamics (PKPD) problems. However it is not possible to design parallel versions of SAEM, and SAEM is very long if the evaluation of the model ϕ is time consuming. The aim of this article is to couple SAEM with a parallel precomputation step.

2 Precomputation

2.1 Principle of precomputation

The idea is the following: to compute quickly a function, we interpolate it from precomputed values, on a grid. The main issue is to construct a grid in an efficient way:

- Interpolation should be easy on the mesh. Here we choose a mesh composed of cubes (tree of cubes) to ensure construction simplicity and high interpolation speed
- Mesh should be refined in areas where the function changes rapidly (speed of variation may be measured in various ways, see below).

2.2 Precomputation algorithm (non parallel version)

Let us describe the algorithm in dimension N. We consider J fixed probabilities $0 < q_j < 1$ with $\sum_{j=1}^J q_j = 1$ and J positive functions $\psi_j(x)$ (required precisions, as a simple example, take $\psi_j(x) = 1$ for every x). We start with a cube (or more precisely hyper-rectangle)

$$C_{init} = \Pi_{i=1}^N [x_{min,i}, x_{max,i}]$$

to prescribe the area of search.

The algorithm is iterative. At step n, we have $1 + 2^N n$ cubes C_i with $1 \le i \le 1 + 2^N n$, organized in a tree. To each cube we attach J different weights ω_i^j (where $1 \le j \le J$, see below for examples of weights), and the 2^N values on its 2^N summits.

First we choose j between 1 and J with probability q_j. Then we choose, amongst the leaves of the tree, the smallest index i such that

$$\frac{\omega_i^j}{\sup_{x \in C_i} \psi_j(x)}$$

is maximum. We then split the cube C_i in 2^N small cubes of equal sizes, which become 2^N new leaves of our tree, the original C_i becoming a node. To each new cube we attach J weights ω_i^j (see below).

Then we iterate the procedure at convenience. We stop the algorithm when a criterion is satisfied or after a fixed number of iterations. We then have a decomposition of the initial cube in a finite number of cubes, organized in a tree (each node having exactly 2^N leaves), with the values of f on each summit.

The crucial point is of course the choice of the weights ω_i^j, which may be linked to the volume of the hypercube, to the variation of the function to study on this cube, or to other more refined criteria.

If we want to evaluate f at some point x, we first look for the cube C_i in which x lies, and then approximate f by the interpolation f_{inter} of the values on the summits of the cube C_i. Note that this procedure is very fast, since, by construction, the cubes form a tree, each node having 2^N nodes. The identification of the cube in which x lies is simply a walk on this tree. At each node we simply have to compare the coordinates of x with the centre of the "node" cube, which immediately gives in which "son" x lies. The interpolation procedure (approximation of $f(x)$ knowing the values of f on the summits of the cube) is also classical and rapid (linear in the dimension N).

Note that it is possible to include more information than simply the values of f, like its gradient or higher order derivatives which sometimes are simply computed using derived models. Interpolation of f in small cubes may then be done by higher order elements.

2.3 Precomputation algorithm (parallel version)

This approach may be parallelized in various way. The best approach is to parallelize the computation of summits. We construct iteratively two ordered lists: a list of evaluations to do, and a list of cubes. The list of cubes is initially void, and the list of evaluations to do is 3^N (the summits of the first cube, and the summits of the first splitting of this cube, in this order).

The list of cubes is ordered according to the weights, as described in the non parallel algorithm.

When all the summits of a sub-cube are computed, it is added to the list of cubes.

When the list of evaluations to do is void, we split the first cube of the list of cubes, which creates at most $3^N - 2^N$ new evaluations to do (some of the new points may be already computed).

When a processor has completed an evaluation, it begins to compute the first point of the list of evaluations to do.

This algorithm insures an optimal use of the processors (no double computations, equal loads between processors).

This algorithm is very versatile. One of the processor (the "master") handles summit list and cube list, updates them and regulates the work of the other $P-1$ "slave" processors. The number of involved processors may be as large as wanted. The time spent in communications will be negligible with respect to the computation time (in the case of complex models like PDEs).

2.4 Weights

Let us now detail some examples of weights ω_i^j. The simplest weight is the volume of the cube C_i. The algorithm then behaves like a classical dichotomy and builds a regular mesh.

Let f_k with $1 \leq k \leq 2^N$ denotes the 2^N values of f at the summits of C_i. Let

$$f_m = \frac{1}{2^N} \sum_{k=1}^{2^N} f_k$$

be their average. Then we may define ω_i as

$$\omega_i^1 = \frac{1}{2^N} \sum_{k=1}^{2^N} |f_k - f_m|.$$

With this weight the mesh will be refined near areas of variations of f.

An other possibility is to define

$$\omega_i^\infty = \sup_{1 \leq k \leq 2^N} |f_k - f_m|.$$

The mesh will also be refined near areas of variations of f, but in a slightly different way.

This last weight may be multiplied by the volume of the cube in order to avoid excessive refinement near discontinuities, which leads to

$$\omega_i^{BV} = vol(C_i) \sup_{1 \leq k \leq 2^N} |f_k - f_m|$$

Another way to construct weights is for instance to evaluate how well f is approximated by affine functions, namely

$$\omega_i^{lin}(C_i) = \inf_{g \in \mathcal{L}} \sup_{x_i} |f_i - g(x_i)|$$

where x_i are the summits of C_i and where $\mathcal{L}$ is the set of affine functions.

Let us now discuss the choice of the ψ_i functions. In some applications it is important to evaluate accurately f in some areas of C_{init}, whereas crude approximations are sufficient in other ares of C_{init}. Let us give an example. Let $\phi(x)$ be some positive function (weight),with $\int \phi(x)dx = 1$. To evaluate $\int_{C_{init}} \phi(x)f(x)dx$ with a precision ε it is sufficient to evaluate $f(x)$ with a precision $\varepsilon/|C_{init}|\phi(x)$. In this case we define

$$\psi_1(x) = \frac{1}{\phi(x)}$$

and consider $\omega_1 = vol$.

2.5 Numerical illustrations

In this section we take $J = 1$ and $\psi_1(x) = 1$ for every x. Let us begin by a simple one dimensional function

$$f(x) = \frac{1}{1 + 1000x^2}.$$

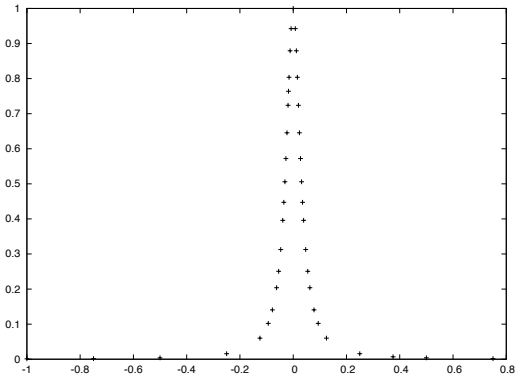

Figure 1: Example in one space dimension

With the second weight, the mesh (see figure 1) is refined near $x = 0$ and is almost uniform in the "f" direction, which is exactly what we want.

In two dimensional space, let us take for instance the following function,

$$f(x, y) = \tanh(20(x + 0.3)) \tanh(10(y - 0.3))$$

which has large variations near $x = -0.3$ and $y = 0.3$. The corresponding mesh (see figure 2) is refined near these two axes.

Figure 3 shows the mesh refinement with $f = 1_{(x-0.3)^2+(y+0.3)^2<0.3}$ (third weight).

2.6 Errors

In practice the evaluation of the function f is not exact. It involves numerical schemes, which are often very time consuming, as soon as partial differential equations are involved. The function f is therefore approximated, up to a numerical error ε_{num}. In our method we approximate f by interpolations of approximate values of f. The global error ε of our method is therefore the sum of two terms

$$\varepsilon = \varepsilon_{num} + \varepsilon_{interp}.$$

Let us discuss here these two error terms.

- The evaluation of the model is not exact, but depends on numerical approximations. For a partial differential equation, let h be the typical size of the mesh. Then the time step k will usually be linked to h by $k \sim C_0 h^\alpha$ for some constants C_0 and α. Typically, $\alpha = 1, 2$. The error of the numerical method is then

  $$\varepsilon_{num} = C_1 h^\beta$$

  for some constant β. The number of cells in the mesh is of order h^{-d} where $d = 2, 3$ is the physical dimension. The number of time steps is of order $h^{-\alpha}$, hence the computational cost of an evaluation is

  $$\tau_{num} = C_2 h^{-d-\alpha}$$

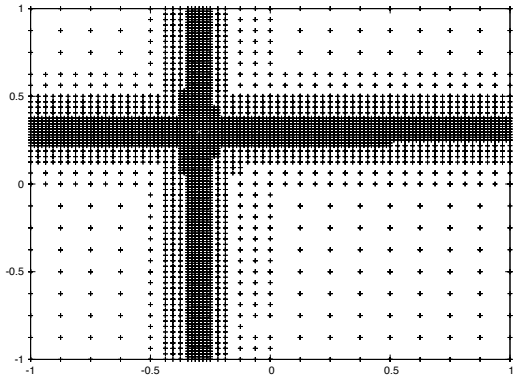

Figure 2: Product of tangents

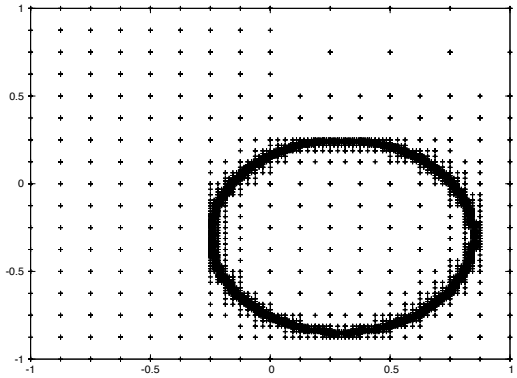

Figure 3: Characteristic function of a disk

The constant C_2 depends on the parameter of the model, often in a severe way, particularly in the case of sharp front propagation (if one thinks about a PDE model for travelling waves). As $\alpha \geq 1$, this means that τ_{num} changes rapidly with h. For simple methods, $\beta = 1$. In this case, to double the precision we need to multiply the computation time by $2^{d+\alpha} \geq 2^{d+1}$, namely 8 if $d = 2$ or 16 if $d = 3$.

If the numerical method is more accurate the situation is better. However even if $\beta = 2$, to double the precision requires to multiply computation time by 4 if $d = 3$.

- Interpolation error ε_{interp}. This error is the maximum difference between f and the interpolation of its exact values at the grid points.

If we want to get an interpolation with a given global error ε, what is the best decomposition of ε into ε_{interp} and ε_{num} ?

If we want to interpolate f with a precision ε_{interp}, we will need $O(1/\varepsilon_{interp})$ points in each variable (for instance if f is smooth and Lipschitz continuous). This requires $O(\varepsilon_{interp}^{-N})$ evaluations of f. Each evaluation must be done with a precision ε_{num} leading to a total computation time of order

$$\tau_{interp} = \varepsilon_{interp}^{-N} \varepsilon_{num}^{-(d+\alpha)/\beta}.$$

Let us define ε by $\varepsilon_{num} = \eta\varepsilon$. Then

$$\tau_{interp} = \varepsilon^{-N-(d+\alpha)/\beta} (1-\eta)^{-N} \eta^{-(d+\alpha)/\beta}.$$

This time is minimal provided

$$N\eta = \frac{d+\alpha}{\beta}(1-\eta)$$

which gives

$$\varepsilon_{num} = \frac{d+\alpha}{d+\alpha+\beta N}\varepsilon$$

as an optimal choice.

2.7 Generic functions

Let C_{init} be the unit cube to fix the ideas. If f is a "generic C^1 function", then to get a precision ε we need to refine the mesh until the size of the sub-cubes is less than $\varepsilon/\|\nabla f\|_{L^\infty}$, which leads to approximately

$$N(\varepsilon) = \Big(\frac{\|\nabla f\|_{L^\infty}}{\varepsilon}\Big)^N$$

sub-cubes. If T_m is the average computation time for one single evaluation of f, the global computation time over N_{proc} processors is

$$T(\varepsilon) = \frac{T_m}{N_{proc}}\Big(\frac{\|\nabla f\|_{L^\infty}}{\varepsilon}\Big)^N.$$

To fix the ideas, if $T_m = 1min$, and $\|\nabla f\|_{L^\infty}/\varepsilon = 16$ (fourth refinement), we get the following computation times

	$N_{proc}=1$	$N_{proc}=8$	$N_{proc}=128$	$N_{proc}=1024$	$N_{proc}=10000$
$N=1$	$16mn$	$2mn$	$8s$	$1s$	–
$N=2$	$4h30$	$32mn$	$2mn$	$15s$	$2s$
$N=3$	$3days$	$8.5h$	$32mn$	$4mn$	$25s$
$N=4$	$1.5month$	$5.6d$	$8.5h$	$1h$	$6mn$
$N=5$	–	$3m$	$5.6d$	$17h$	$1.7h$
$N=6$	–	–	$3m$	$11d$	$1.1d$
$N=7$	–	–	–	$6m$	$18d$

For the fifth refinement ($\|\nabla f\|_{L^\infty}/\varepsilon = 32$), we get

	$N_{proc}=1$	$N_{proc}=8$	$N_{proc}=128$	$N_{proc}=1024$	$N_{proc}=10000$
$N=1$	$32mn$	$4mn$	$16s$	$2s$	–
$N=2$	$17h$	$2h$	$8mn$	$1mn$	$6s$
$N=3$	$22d$	$3d$	$4h30$	$32mn$	$3.3min$
$N=4$	–	$3m$	$5.6d$	$17h$	$1.7h$
$N=5$	–	–	$6m$	$22d$	$2.3d$

As we see, under these conditions, $N = 4$ or $N = 5$ or $N = 6$ is the practical limitation of our method, even on supercomputers. It is therefore crucial to improve our strategy in order to refine the grid only in area of interests or to use special properties of f to reduce computational cost. We review a few possible strategies in the next paragraphs.

2.8 Remarks

2.8.1 Functions with sharp transitions

If f has large constant areas, with sharp transitions between them, the situation is in fact better. For instance if $f = 1_D$ where $D \subset C_{init}$, then with ω_i^1 or ω_i^∞ the mesh will be highly non homogeneous and will focus on ∂D. Let N' be the dimension of ∂D. To localize D with a precision ε we will need

$$C\frac{|\partial D|}{\varepsilon}^{N'}$$

sub-cubes. The interesting dimension is now N' and not N. For this type of functions we gain $N - N'$ dimensions in terms of computational time.

2.8.2 Monotonic functions

If f is monotonic with respect to all of its variables, then it is sufficient to compute its values on two summits to control the value of f in the whole sub-cube (the "upper right" and the "lower left" summits). For such functions, less evaluations are required for the last refinement. Namely when we split a cube for the last time, we need not to compute all the summits of the 2^N sub-cubes. This is equivalent to the gain of one dimension.

2.8.3 Parameter sensitivity

In general some parameters will have more influence than others. Let

$$S_i = \|\partial_i f\|_{L^\infty}.$$

Let us assume, up to a change of labels, that $S_1 \geq S_2 \geq ... \geq S_N$. If we want to get a precision of order ε it is useless to take into account some of the parameters in a first approach. More precisely, if $S_M + S_{M+1} + ... + S_N < \varepsilon$ then we may fix the values of the variables M, ..., N with a resulting error less than ε. The dimension of the model then reduces to M.

2.8.4 Iterative refinement

The idea is at first to create a rough grid (with 4^N or 8^N cubes), which leads to a rough interpolation function f^1_{inter}. We then use f^1_{inter} as a first approximation in the optimization algorithm. This indicates where the minimum probably lies. During the next step we focus on the 2^N cubes near the possible minimum. We then refine these cubes and split each sub-cube in 4^N.

After this procedure, we are left with sub-cubes of size 1/16 near the possible minimum. However we have only computed $4^N + 2^N \times 4^N$ or $8^N + 4^N \times 2^N$, namely $O(8^N)$ cubes instead of $O(16^N)$ cubes. Hence dimension N reduces to $N/2$, which is a great improvement. This procedure may be iterated, which leads to even greater improvements.

2.8.5 Concluding remarks

The simplest strategy is very expensive, and limits the number of parameters to $N = 4$ or $N = 5$. However, iterative refinement or parameter sensitivity analysis, or monotonicity efficiently reduce the computational cost. Dimensions of order 8 to 10 may be reachable.

To go above these dimensions, many iterative refinements may be helpful if the problem has a sufficiently good behaviour.

3 Two couplings between precomputation and SAEM

3.1 Precomputation before SAEM

The easiest way is first to run the precomputation step, in order to get a mesh with a given interpolation precision.

The precomputation step is long but can be parallelized very efficiently. Its output is an approximation of the model ϕ_{app} through grid interpolation.

The SAEM step is then done on the approximate model ϕ_{app}. It is a much faster step. It converges to $\theta^+_{pop,app}$, approximation of the most likely population parameter $\theta_{pop,app}$ for the model ϕ_{app}. As the precision of the interpolation increases, ϕ_{app} converges to ϕ and therefore $\theta^+_{pop,app}$ goes to θ^+_{pop}. As the number of individuals and data increases, θ^+_{pop} converges to θ_{pop}.

Note that the precomputations can be reused to deal with another set of data. This is particularly useful if a new individual is added in the study, or if new data are added.

We will illustrate this approach in the next section on the KPP equation.

3.2 Simultaneous precomputation and SAEM

The main drawback of the previous approach is that the precomputation step will mesh the whole parameter domain, whereas SAEM algorithm will concentrates on particular areas of the individual parameters. Most of the precomputations will therefore be useless, and it is more efficient to concentrate the precomputation where SAEM requires them. In this paragraph, we propose a mix of precomputation and SAEM.

First, we choose some initial precision $\varepsilon_{initial}$ and run the precomputation step with this initial precision. We get a first model approximation ϕ_1, such that $|\phi - \phi_1|$ is of order $\varepsilon_{initial}$.

We then run SAEM algorithm, which converges to some approximation θ_1 of the population parameters θ_{pop}. SAEM algorithm also gives for each individual $1 \leq i \leq N$ a sequence of individual parameters $(Y_{k,i})_k$ with distribution, the conditional distribution of Z_i

$$P_i^1(Z_i) = \frac{P\Big((y_i)_i, Z_i|\theta_1\Big)}{g_1},$$

where g_1 is the normalization constant

$$g_1 = \int P\Big((y_i)_i, Z_i|\theta_1\Big) dZ_1.$$

For the individual i, SAEM will run the model ϕ_1 using the parameters $(Y_{ki})_k$. Therefore SAEM will evaluate the individual parameters Z for individual i with probability $P_i^1(Z)$.

As all the individuals are independent, SAEM needs to evaluate the model ϕ_1 at Z with probability

$$P^1(Z) = \frac{1}{N} \sum_{1 \leq i \leq N} P_i^1(Z).$$

In many applications, the distribution functions P_i^1 tends to be very localised, hence P^1 is negligible outside a small parameter set.

It is useless to try to compute accurately the models in areas where P^1 is very small. On the contrary we need to focus the precomputation step on areas where P^1 is large.

After this first run of SAEM, we will therefore again run the precomputation algorithm with a precision $\varepsilon_1(Z)$ which will depend on Z. The aim is to compute the model with high precision where P^1 is large, and with little precision where P^1 is small. However we have to take care that θ_1 is only an approximation of θ_{pop}. There is no unique way to design $\varepsilon_1(Z)$. A simple way is to define

$$\varepsilon_1(Z) = \min_{|Z'| \leq \delta_1} \frac{\varepsilon_{initial}}{1 + \lambda_1 P^1(Z + Z')}.$$

In this expression, δ_1 will take care of the uncertainty on ε_1, and λ_1 is linked to the desired precision enhancement.

After precomputation with $\varepsilon_1(Z)$, we then again run SAEM algorithm, which gives a new approximation θ_2 of θ_{pop} and an new distribution P^2. By iteration we define θ_n and P^n. We choose δ_1, a sequence which slowly goes to 0 and λ_1, a sequence which goes to infinity.

As $\lambda_n \to +\infty$, θ_n goes to θ_{pop}. If λ_n growths to fast or if δ_n decreases to fast, then the convergence is slower.

The mixed algorithm can then be implemented as follows: at each time we have a precomputation mesh $M(t)$, a population guess $\theta(t)$ and distribution $P(t, Z)$.

- One processor runs SAEM, using the mesh $M(t)$. This gives a population guess $\theta(t)$ and a distribution $P(t, Z)$.
- One cluster of processor improves the mesh in order to fit precision

$$\min_{|Z'| \leq \delta(t)} \frac{\varepsilon_{initial}}{1 + \lambda(t) P(t, Z + Z')},$$

  where $\lambda(t)$ goes slowly to $+\infty$ and $\delta(t)$ goes slowly to 0.

4 Application: parametrization of a KPP model with Monolix

We want to illustrate the previous methodology in the context of the estimation of the parameter associated to the so called KPP equation: it is a reaction-diffusion model described by a PDE which was mathematically studied in the pioneering work of Kolmogoroff, Petrovsky and Piscounoff [5]. Such kind of equation is also sometimes referred to as the Fisher equation, introduced in the context of the theory of evolution [2]. Actually, due to its nature, such a model can be used in numerous fields to better understand *propagation* phenomena (flame propagation, species invasion, etc) thanks to the existence of particular solutions called "travelling waves".

In this section, we generate a virtual population of solutions of the KPP equation, assuming Gaussian distributions on its parameters, and adding noise. We then try to recover the distributions of the parameters by an SAEM approach (using Monolix software [8]). For this we first precompute solutions of the KPP equation on a regular or non regular mesh, and then run SAEM algorithm using interpolations of the precomputed values of KPP equation (instead of the genuine KPP). We discuss the effect of noise and the precision of the parameters estimates.

4.1 Presentation of the problem

We consider the following classical version of the KPP (or reaction-diffusion) equation:

$$\partial_t u - \nabla.(D\nabla u) = Ru(1-u), \tag{5}$$

where $u(x)$ is the unknown concentration (assumed to be initially a compact support function, for instance), D the diffusion coefficient and R the reaction rate. These equations are posed in a domain Δ with Neumann boundary conditions. Note that the geometry of the domain Δ can be rather complex (e.g. when u is the density of tumor cells in the brain). Initially the support of u is very small and located at some point $x_0 \in \Delta$. Therefore we may assume that

$$u(T_0, x) = \alpha 1_{|x-x_0|\leq\varepsilon}, \tag{6}$$

for some time T_0 (in the past). It is well known that for (5)-(6), there exist a propagation front (separating zones where $u \equiv 1$ and zones where $u \ll 1$). The size of the invaded zone (= the zone where u is close to 1) is defined as

$$S(t) = \int_\Delta u(t,x)dx$$

or by

$$S(t) = vol\Big(u \geq \sigma\Big),$$

where $\sigma > 0$ is some detection threshold.

We assume that we have data from a population of individuals at various times $t_1, ..., t_N$. For an individual, let S_1, ..., S_N be these data. If we want to compare aforementioned KPP model with data, we have to look for solutions of (5) with initial data (6) such that $S(t_i)$ is close to S_i for $1 \leq i \leq N$.

As a first approximation, α and ε may be fixed to given values (e.g. $\alpha \leq 1$, $\varepsilon \ll 1$)[1]. It remains to find T_0, x_0, D and R such that

$$\theta(x_0, D, R) = \sum_{1 \leq i \leq N} |S(t_i) - S_i|^2$$

is minimum. It is very long to minimize θ since each evaluation of θ requires the resolution of a complete partial differential equation in a complex domain.

If now we have a collection of individuals P_j ($1 \leq j \leq M$), with N_j data for the individual $\#j$ (sizes $S_{i,j}$ at times $t_{i,j}$), we may be interested in a populational approach to parametrize the model. Namely, we may be concerned in the distribution of the model parameters in the population which maximize the likelihood of the observations and may want to look for the mean and standard deviation of each parameter in the population, e.g.

$$D \sim \mathcal{N}(\bar{D}, D^\sigma)$$

(normal distribution with mean $\bar{D}$ and standard deviation D^σ) and similarly for R, x_0 and T_0. Other probabilities may be considered (uniform law, log normal law, ...).

The maximization of the likelihood of the observations (through SAEM algorithm) leads to a large number of evaluations of the model, with a huge computational cost. We will therefore test our method on this problem.

Remark. An example of such problem is given by clinical data of patients with brain tumors called gliomas. The density of tumor cells is given by u and the size of the tumor is given by $S_{i,j}$ which is measured with MRI. But the spectrum of application of the method is as wide as the one of the fields described by KPP equation.

4.2 Technical details

The following method may be applied to any domain Δ. For sake of simplicity, we present the results in the one dimensional case but the same approach can be done with 2D or 3D images.

Note that the equation is left unchanged if we multiply D and R by some constant and divide time by the same constant. This reduces by one the number of parameters. The independent parameters of the model are: D/R and x_0. Note that in three dimensional space we would have two more parameters y_0 and z_0.

We first give a priori bounds on these various parameters: $0 \leq x_0 \leq 1$. For R and D, we assume, following classical values of the literature in the context of glioma modelling, $7.2 \times 10^{-3} \leq R \leq 4.0 \times 10^{-2}$ and $2.5 \times 10^{-7} \leq D \leq 13.9 \times 10^{-7}$. This leads to $6.2 \times 10^{-6} \leq D/R \leq 1.9 \times 10^{-4}$. For rescaled time, we will consider $0 \leq t \leq 8000$. We note that for all KPP simulation, the initial solution has a support with $\varepsilon = 0.03$ and $\alpha = 1$.

Note that D/R is related to the "width" of the progressive waves, and the computation costs increases drastically as D/R decreases. Typically, one evaluation lasts between a few seconds and a few minutes on a single processor, using standard PDE solvers, depending on the value of D/R.

As described in previous sections, if the weight is the volume, the precomputation grid is uniform. It takes 40mn7s for the fifth uniform splitting (4^5 cubes, see Figure 4) on two cores

[1]Note that to have the existence of a travelling wave associated to the invasion front, the maximum of $u(T_0)$ should be sufficiently close to 1. If this is not the case, diffusion will be dominant for small times and then, for longer times, there will be a global growth associated to reaction and no travelling wave.

(one master and one worker) of a quad-core AMD Opteron (2.7 GHz) and 40 hrs 31 mn 10 s for the eighth splitting (4^8 cubes). Of course this could be parallelized, with a complete efficiency: 2 mn 54 s on 16 cores (one master and 15 workers) for the fifth uniform, and 3 hrs 3 mn18 s for the eighth splitting, that means 14 times less than with only one worker. We implemented such parallelized algorithms in Python.

For the uniform grid, the parallelization is trivial as all the computations are independent. For the non-uniform grid, we define, for each iteration of the refinement, all the summits to be computed, and do all the computation in parallel. The determination of these summits has to be done in a sequential way. The cost of the KPP computation is long enough to keep a good efficiency of the parallelization.

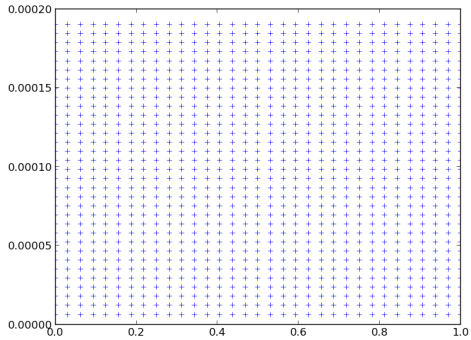

Figure 4: An example of a uniform mesh of the space of parameters (fifth uniform splitting). The two parameters are $w = D/R$ and x_0.

Approximate evaluation of the model through interpolation is very fast (far below 1 second). As a consequence, a Monolix run lasts typically 10 minutes with our current implementation; it depends also on the number of estimated parameters. This is more than for a simple ordinary differential equation, but is still a reasonable time compared to an inverse problem approach. We detail a few specific examples in the next section.

4.3 Results of the parameters estimation

Within the KPP framework, we can separate the nature of the parameters between those related to reaction/diffusion (R and D) and those related to the space (x_0). We will present here several tests of parameters estimation. To do so, we consider a population of 100 individuals characterized by (D, R, x_0). Random parameters are generated with a lognormal distribution. The individuals are generated by solving (5)-(6) with the associated set (D, R, x_0). From these solutions (eventually perturbed by a given noise), we extract 101 values in time to obtain individual time series $\{t_{i,j}, S_{i,j}\}$, $i = 1..101$, $j = 1..100$. These series will be given to Monolix as data to perform the parameter estimation of this population.
We begin by using an homogeneous precomputed grid with 1089 points (fifth splitting, $(2^n + 1)^2$ points with $n = 5$, see figure 4), solutions of KPP (Tests 1 to 3, below): we investigate the ability of the Monolix software to estimate the parameters for three populations characterized by their levels of noise with respect to the exact KPP solution : first population has zero noise (i.e. it is an "exact" solution of KPP), second has a 5% noise and third has a 10% noise.
Then, to illustrate the interest of having an inhomogeneous precomputed grid, we performed

parameter estimation (Test 4, below) with a 500 points grid (see figure 5) built with the 'BV' weight (see 2.4).

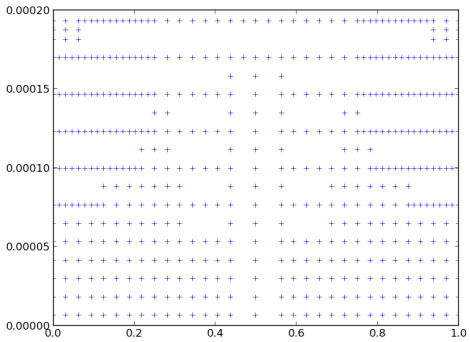

Figure 5: An example of an inhomogeneous mesh of the space of parameters (with 500 points). The two parameters are $w = D/R$ and x_0. We can see that the finer zones have a smaller size than the size the homogeneous mesh of Figure 4 (which contains 1089 points): the mesh is here more refined in zones where the model has strong variations and coarsened in zones where the variations are small.

4.3.1 Test 1: x_0 fixed

We fix $x_0 = 0.63$. And we tune Monolix to perform an estimation of R and D. We refer the reader to the Monolix documentation [8] for a full description of what is achieved by this software and of the outputs which can be obtained.

We begin by describing the results and some outputs in the case of a population whose discrete data come from the resolution of the full KPP equation (i.e. a population without noise). To fix the ideas, Figure 6 shows the data and the curves fitted thanks to the model and the obtained parameters by the Monolix run, for 12 individuals (note that the same quality is obtained for the 100 individuals; for sake of brevity, we thus limit the presentation to 12 of them).

More precisely, we can compare the results of the mean parameters of the population obtained by Monolix (see Table 1, column "E1") and the mean "theoretical" parameters used to build the population (see Table 1, column "Theor"). This allows to quantify the ability of Monolix to estimate the parameters. We can see that the results are fairly good and are associated to a good convergence of the SAEM algorithm (see Figure 7). In addition the results of Table 1 can be illustrated by the Monolix output of observed *v.s.* predicted data, both for the population and individual data (see Figure 8, left and right respectively).

Let us note that the convergence graphs of the SAEM (Figure 7) and the comparison of population and fitted data (Figure 6) are of the same quality for all the following tests (SAEM algorithm will always be converged). By the way, in the following, we will not show these kind of graphs and we will only give the meaningful information, that is the "Table of results and errors" as well as the "observed *v.s.* predicted" graphs.

Then, we performed the same run but with a population of individuals who are perturbed

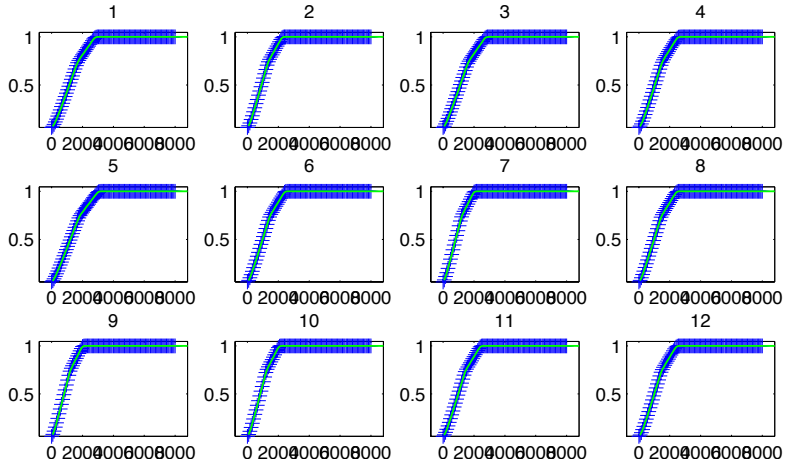

Figure 6: Test 1: 12 individuals of the population (in blue) and the corresponding estimated curves (in green) computed by Monolix using the KPP model.

	Theor	E1		E2		E3	
			error		error		error
R	0.0238	0.0238	0%	0.024	0.8%	0.0247	3.8%
D	$8.20e^{-7}$	$7.93e^{-7}$	-3.3%	$7.85e^{-7}$	-4.3%	$7.58e^{-7}$	-7.6%
ω_R	0.189	0.186	-1.6%	0.189	0%	0.227	20%
ω_D	0.189	0.187	-1.1%	0.197	4.2%	0.21	11%

Table 1: Test 1: Results (from Monolix) and errors for the mean parameters of the population. Column E1 refers to a population without noise (see text). Column E2 (resp. E3) refers to a population with a 5% (resp. 10%) noise.

with a random noise of amplitude 5% (i.e. since the data are between 0 and 1, a maximum noise of 0.05). The results of the Monolix run are given in Table 1 (column E2) for the estimated parameters of the whole population. Naturally, the effect of noise can be seen in these results where we note a slightly decreasing accuracy of the estimation. But the results are still quite good. These facts are confirmed on Figure 9 where the noise induces a slight dispersion of the points cloud.

Finally, we performed the same run but with a population of individuals who are perturbed with a random noise of amplitude 10% (i.e. since the data are between 0 and 1, a maximum noise of 0.1). The results of the Monolix run are given in Table 1 (column E3) for the estimated parameters of the whole population. Again, this additional noise leads to a poorer estimation of the parameter but the accuracy is still reasonable, taking into account the significant amount of noise. The observed *v.s.* predicted data are shown on Figure 10.

4.3.2 Test 2: R and D fixed

We fix $R = 10^{-2}$ and $D = 10^{-6}$. We tune Monolix to perform an estimation of x_0. And we proceed as for the Test 1, on 3 populations with respect to the noise on the data (none, 5% and 10%).

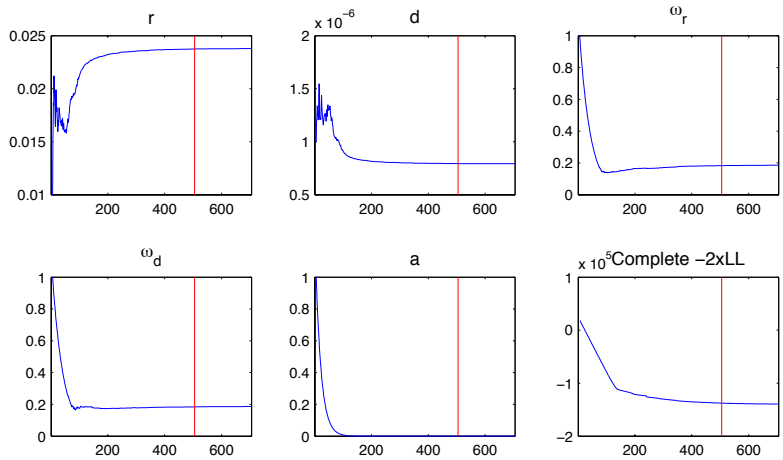

Figure 7: Test 1: Graphs showing the convergence of the SAEM algorithm in Monolix. Case with no noise.

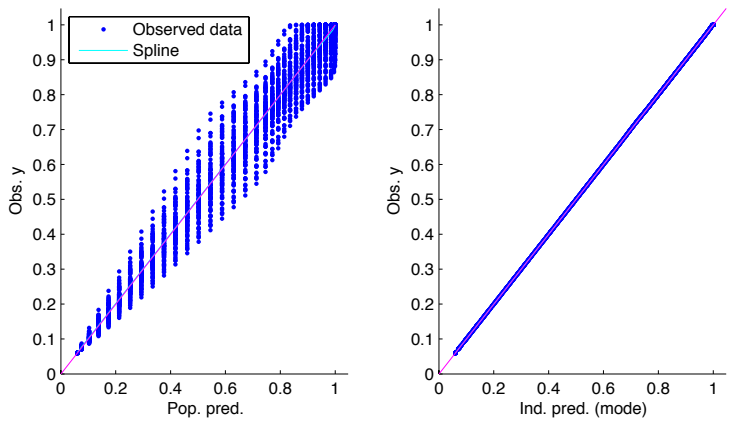

Figure 8: Test 1: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with no noise.

Results are summarized in Table 2. Again, with no noise, parameter estimation is very good. Adding some noise induces a poorer estimation of the parameters but the accuracy is still very good. We note that for the two levels of noise, results are the same in terms of mean populational parameters, but the individual parameters are not the same. Dispersion associated to the noise can be seen by comparing Figures 11, 12 and 13.

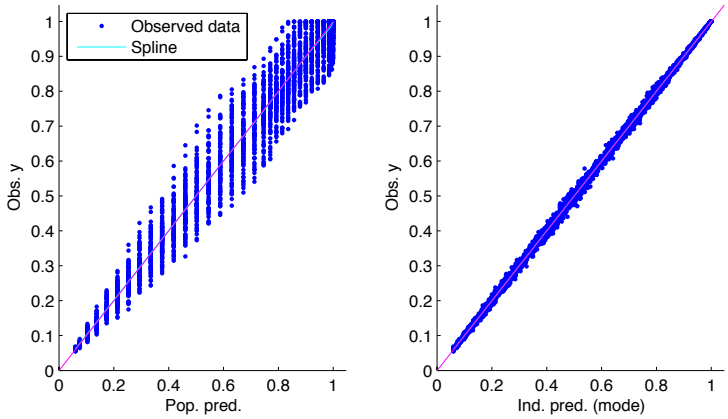

Figure 9: Test 1: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with 5% noise.

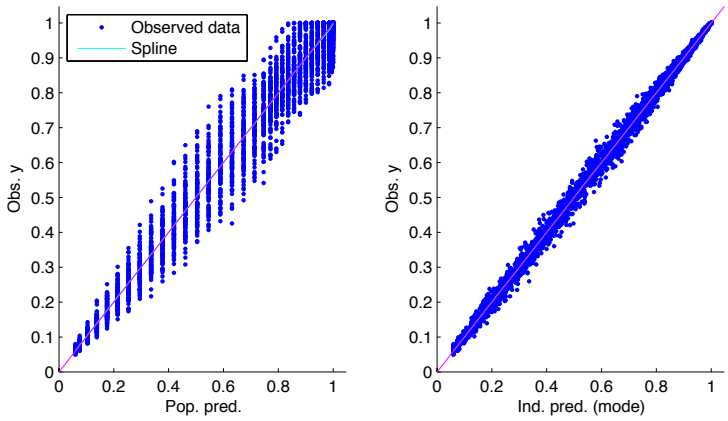

Figure 10: Test 1: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with 10% noise.

	Theor	E1		E2		E3	
			error		error		error
x_0	0.410	0.412	0.5%	0.418	2%	0.418	2%
ω_{x_0}	0.287	0.282	-1.7%	0.296	3.1%	0.296	3.1%

Table 2: Test 2: Results (from Monolix) and errors for the mean parameters of the population. Column E1 refers to a population without noise (see text). Column E2 (resp. E3) refers to a population with a 5% (resp. 10%) noise.

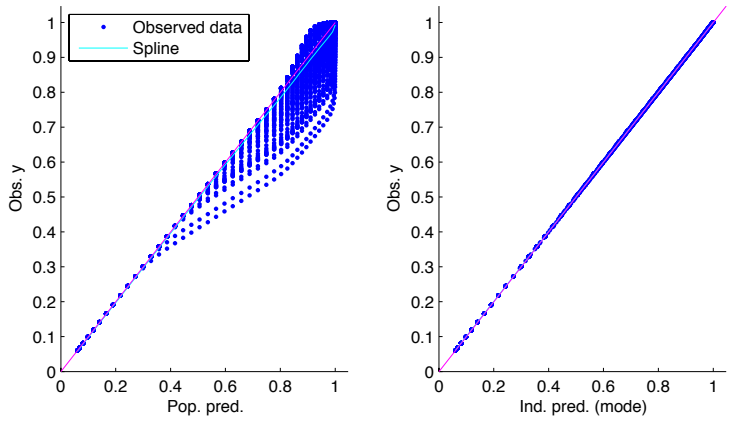

Figure 11: Test 2: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with no noise.

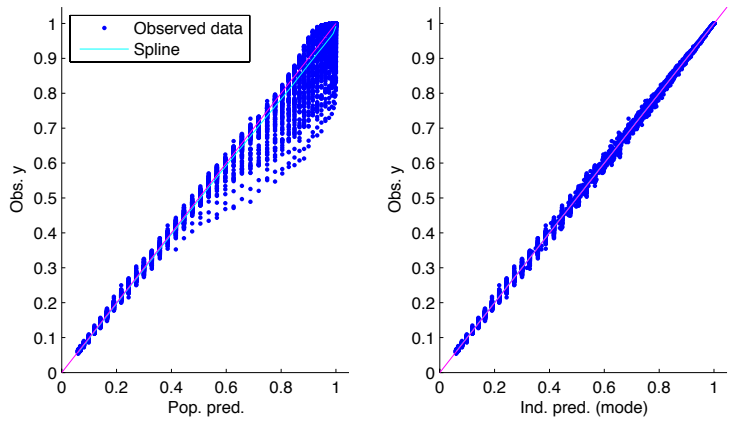

Figure 12: Test 2: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with 5% noise.

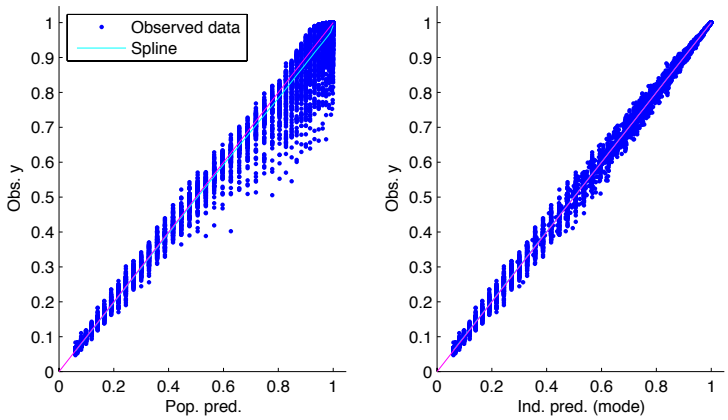

Figure 13: Test 2: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with 10% noise.

4.3.3 Test 3: estimation of x_0, R and D

Here, all the parameters of the population are random and we tune Monolix to perform an estimation of x_0, R and D. Again, we proceed as for the Test 1, on 3 populations with respect to the noise on the data (none, 5% and 10%).

This test with 3 parameters to estimate is rather challenging. We see in Table 3 that the results are a bit less accurate than those in Tests 1 and 2, but they are still of good quality. Adding some noise again deteriorate the accuracy but the results are reasonable for practical applications.

Dispersion associated to the noise can be seen by comparing Figures 14, 15 and 16.

	Theor	E1		E2		E3	
			error		error		error
R	0.0245	0.237	-3.3%	0.0234	-4.5%	0.0231	-5.7%
D	$8.64e^{-7}$	$8.67e^{-7}$	0.3%	$8.79e^{-7}$	1.7%	$9.62e^{-7}$	11%
x_0	0.415	0.399	-3.9%	0.393	-5.3%	0.37	-11%
ω_R	0.201	0.196	-2.5%	0.263	31%	0.253	26%
ω_D	0.205	0.188	-8.3%	0.247	20%	0.395	93%
ω_{x_0}	0.254	0.244	-3.9%	0.241	-5%	0.616	143%

Table 3: Test 3: Results (from Monolix) and errors for the mean parameters of the population. Column E1 refers to a population without noise (see text). Column E2 (resp. E3) refers to a population with a 5% (resp. 10%) noise.

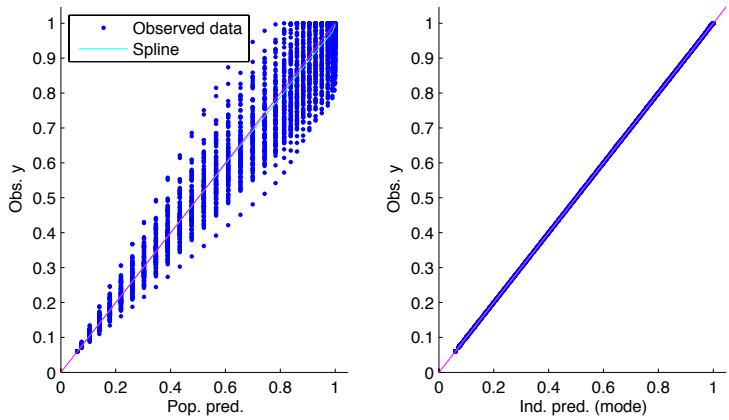

Figure 14: Test 3: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with no noise.

4.3.4 Test 4: estimation of x_0, R and D with inhomogeneous grid

Here, as in Test 3, all the parameters of the population are random and we tune Monolix to perform an estimation of x_0, R and D. The difference is that we use an heterogeneous mesh for

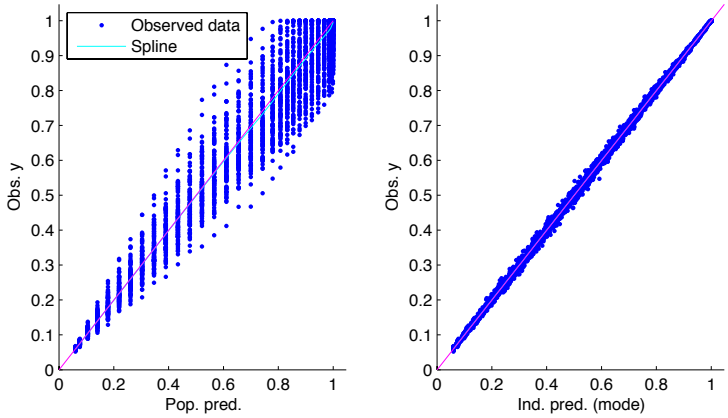

Figure 15: Test 3: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with 5% noise.

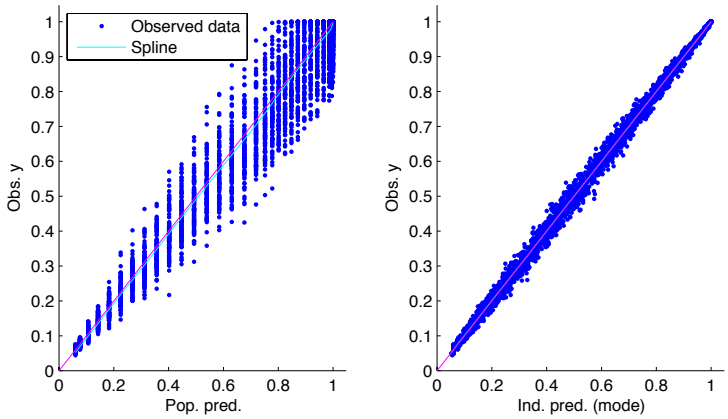

Figure 16: Test 3: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with 10% noise.

the interpolation on the parameter space. It is important to notice that the non-uniform grid contains half of the points of the uniform grid.

We see in table 4 that the results are as good as in Test 3. The grid with only 500 points gives the same accuracy on the evaluation of the solution of the KPP model. This is a good achievement since we have the same precision with a smaller computational cost.

Dispersion associated to the noise is also shown on Figures 17, 18 and 19.

	Theor	E1		E2		E3	
			error		error		error
R	0.0245	0.0245	0%	0.0241	-1.6%	0.0239	-2.4%
D	$8.64e^{-7}$	$8.31e^{-7}$	-3.8%	$8.47e^{-7}$	-1.9%	$8.66e^{-7}$	0.2%
x_0	0.415	0.414	-0.2%	0.406	-2.1%	0.436	5%
ω_R	0.201	0.197	-1.9%	0.238	18.4%	0.257	27.8%
ω_D	0.205	0.191	-6.8%	0.238	16%	0.299	45.8%
ω_{x_0}	0.254	0.262	3.1%	0.247	-2.7%	0.290	14.1%

Table 4: Test 4: Results (from Monolix) and errors for the mean parameters of the population. Column E1 refers to a population without noise (see text). Column E2 (resp. E3) refers to a population with a 5% (resp. 10%) noise.

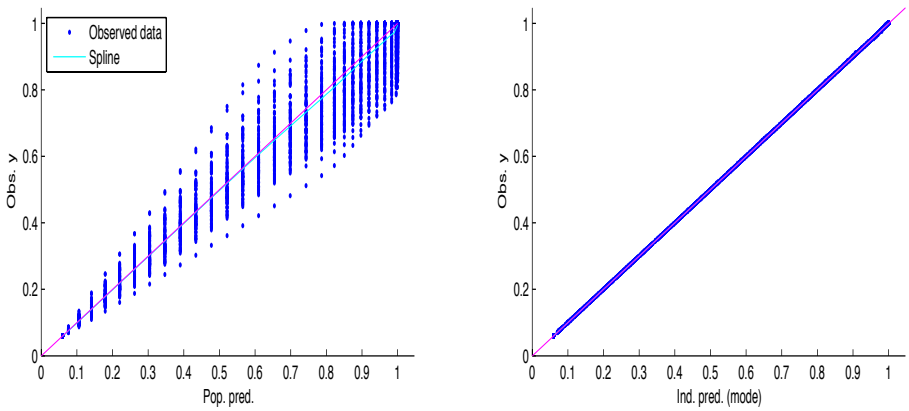

Figure 17: Test 4: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with no noise.

4.4 Computational cost comparison

The main interest of this methodology is to tackle problem of parameters identification in complex PDE systems. The gain in term of computational cost can be easily evaluated and illustrate the feasability of the method for a large range of problems. The computational cost of the whole algorithm that mean generation of the mesh and SAEM computation, can be divided in two distinct parts: an offline time corresponding to the computation of the mesh, which can be done once and for all, and an online time corresponding to the estimation of the parameters for a given population.

Table 5 illustrates this different times and shows the gain du to the method. In particular, the exact case refer to the SAEM algorithm solving the whole PDE.

The considered case is the one explained in 4.3.3: estimation of x_0, R and D for a population of 100 individuals.

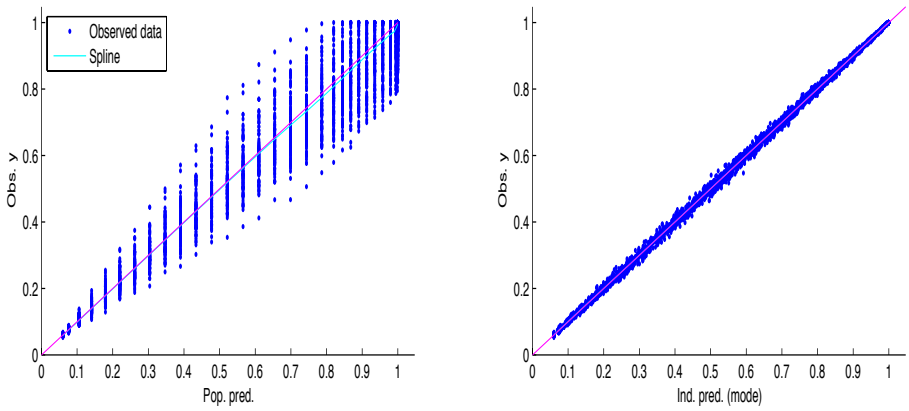

Figure 18: Test 4: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with 5% noise.

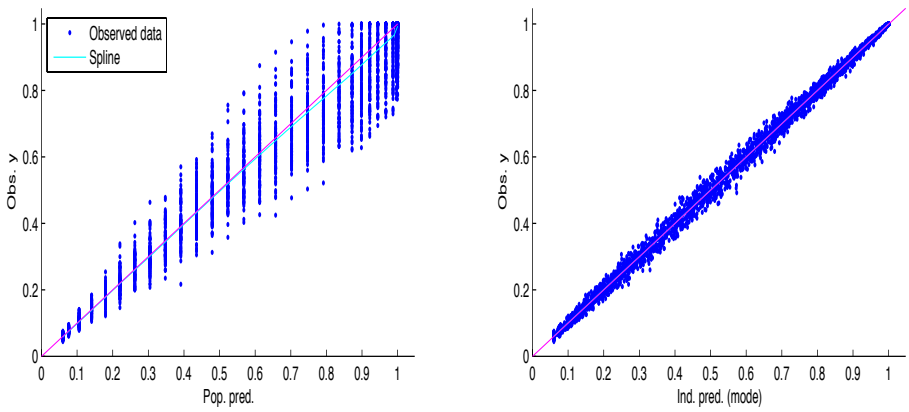

Figure 19: Test 4: Monolix output of observed *v.s.* predicted data both for the population and individual data. Case with 10% noise.

5 Conclusion

In this paper a new method combining SAEM algorithm and a precomputation step has been presented. This method could be helpful to reduce the overall computation time when the model is very long to compute, for instance when the model is based on partial differential equations. To our best knowledge, this is the first demonstration of parameter estimation of PDE thanks to a SAEM algorithm. In a future work we intend to study in details the method which performs simultaneously the precomputation of the parameter space and the SAEM algorithm.

Acknowledgements. The authors would like to thank B. Ribba, M. Lavielle as well as all the Lixoft team and M. Herda for many interesting discussions and interactions. P. Vigneaux was supported by an INRIA "délégation" during this work. Numerical simulations were done

	"Exact" case	Interpolation with homogeneous mesh	Interpolation with heterogeneous mesh
Offline	No offline computation	Mesh with n segmentations, $(2^n + 1)^2$ points. For 5 segmentations, 1089 points	Mesh with n points. Example with 500 points
Unit average CPU cost	-	$2.12s$	$2.12s$
Offline total CPU cost	-	$38mn28s$	$17mn40s$
Online	SAEM, 10^6 KPP evaluations	SAEM, 10^6 interpolations	SAEM, 10^6 interpolations
Unit average CPU cost	$2s$	$4.5 \times 10^{-4}s$	$5.1 \times 10^{-4}s$
Online total Cost	$\sim 23\ days\ 3\ h$	$7mn30s$	$8mn30s$
Total cost	$\sim 23\ days\ 3\ h$	$45mn58s$	$26mn10s$

Table 5: Offline and online computational cost for the different approaches.The number of calls of the solver in SAEM is about 10^6 for this case. Note that this is sequential CPU time. The mesh generation can be easily parallelize on many cores with an excellent scalability.

using computer resources of the Pole Scientifique de Modelisation Numerique (PSMN), Lyon, France.

References

[1] B. Delyon, M. Lavielle, and E. Moulines. Convergence of a stochastic approximation version of the EM algorithm. *The Annals of Statistics*, 27(1):94–128, 1999.

[2] R.A. Fisher. *The genetical theory of natural selection.* Oxford University Press, 1930.

[3] Bernard Haasdonk and Boris Lohmann. Special issue on "model order reduction of parameterized problems". *Mathematical and Computer Modelling of Dynamical Systems*, 17(4):295–296, 2011.

[4] Jari Kaipio and Erkki Somersalo. *Statistical and computational inverse problems*, volume 160 of *Applied Mathematical Sciences.* Springer-Verlag, New York, 2005.

[5] A. Kolmogoroff, I. Petrovsky, and N. Piscounoff. Etude de l'equation de la diffusion avec croissance de la quantite de matiere et son application a un probleme biologique. *Bulletin de l'universite d'Etat a Moscou. Section A*, I(6):1–26, 1937.

[6] E. Kuhn and M. Lavielle. Maximum likelihood estimation in nonlinear mixed effects models. *Computational Statistics and Data Analysis*, 49(4):1020–1038, 2005.

[7] W. H. A. Schilders, H. A. van der Vorst, and J. Rommes (Eds.). *Model Order Reduction: Theory, Research Aspects and Applications.* Springer Berlin Heidelberg, 2008.

[8] Monolix Team. *The Monolix software, Version 4.1.2. Analysis of mixed effects models.* LIXOFT and INRIA, http://www.lixoft.com/, March 2012.

Contents

1 SAEM algorithm **5**
1.1 Principle of populational approaches . . . 5
1.2 SAEM algorithm . . . 6

2 Precomputation **7**
2.1 Principle of precomputation . . . 7
2.2 Precomputation algorithm (non parallel version) . . . 7
2.3 Precomputation algorithm (parallel version) . . . 8
2.4 Weights . . . 9
2.5 Numerical illustrations . . . 9
2.6 Errors . . . 10
2.7 Generic functions . . . 12
2.8 Remarks . . . 13
2.8.1 Functions with sharp transitions . . . 13
2.8.2 Monotonic functions . . . 13
2.8.3 Parameter sensitivity . . . 13
2.8.4 Iterative refinement . . . 14
2.8.5 Concluding remarks . . . 14

3 Two couplings between precomputation and SAEM **14**
3.1 Precomputation before SAEM . . . 14
3.2 Simultaneous precomputation and SAEM . . . 14

4 Application: parametrization of a KPP model with Monolix **16**
4.1 Presentation of the problem . . . 16
4.2 Technical details . . . 17
4.3 Results of the parameters estimation . . . 18
4.3.1 Test 1: x_0 fixed . . . 19
4.3.2 Test 2: R and D fixed . . . 20
4.3.3 Test 3: estimation of x_0, R and D . . . 25
4.3.4 Test 4: estimation of x_0, R and D with inhomogeneous grid . . . 25
4.4 Computational cost comparison . . . 27

5 Conclusion **28**

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

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

ISSN 0249-6399