ESAIM: M2AN 48 (2014) 1303–1329
DOI: 10.1051/m2an/2013140

ESAIM: Mathematical Modelling and Numerical Analysis
www.esaim-m2an.org

PARAMETER ESTIMATION IN NON-LINEAR MIXED EFFECTS MODELS WITH SAEM ALGORITHM: EXTENSION FROM ODE TO PDE

E. Grenier[1], V. Louvet[2] and P. Vigneaux[1]

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 population 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 population parametrization of complex models, including partial differential equations models. We illustrate our method on the classical KPP equation.

Mathematics Subject Classification. 35Q62, 35Q92, 35R30, 65C40, 35K57.

Received February 11, 2013. Revised October 13, 2013.
Published online July 28, 2014.

1. Introduction

The context of this paper is the following: we assume that we have a series of individuals forming a population and that we study the evolution of a given characteristic in time, of each individual (*e.g.* the size, the weight, a drug concentration, *etc.*). We further assume that we have a model (an ordinary differential equation (ODE), a system of ODE, a partial differential equation (PDE), or any other mathematical model) which depends on parameters and allows, providing the good parameters associated to an individual, to compute a curve which fit the time evolution of such individual. In this paper, the time evolving quantity to be fitted is assumed to be scalar. The considered problem is thus, given the discrete time evolution of each individual of the population, to recover the parameters of the model verifying the best fit (in a sense to be defined). It is a well-known kind of problem which enters in the so called "inverse problem" category.

To do so we can distinguish two general kinds of strategies. The first one is to consider each individual separately and to apply any inverse problem methodology to compute the "best" parameters of one individual (without using the information known on the other individuals). There is an abundant literature for this "individual" inverse problem and we refer to the following books for an overview [1, 9, 20]. Let us mention in

Keywords and phrases. Parameter estimation, SAEM algorithm, partial differential equations, KPP equation.

[1] U.M.P.A., Ecole Normale Supérieure de Lyon, CNRS UMR 5669 & INRIA, Project-team NUMED. 46 Allée d'Italie, 69364 Lyon Cedex 07, France. emmanuel.grenier@ens-lyon.fr

[2] Institut Camille Jordan, CNRS UMR 5208 & Université Lyon 1 & INRIA, Project-team NUMED. 43 Boulevard du 11 novembre 1918, 69622 Villeurbanne Cedex, France.

particular the so called "adjoint state" method which belongs to this category and has a long history [7, 16]. Such approaches are generally efficient but if the population has a lot of individuals, the computational time to obtain the parameters for all individuals can be really long.

The second strategy is to use so called "population" approaches which turn the parameters estimation in a "statistical" inverse problem where all the population data are used together to estimate, first, the average population parameters and, second, the inter individual variability (in other words, the deviation from the mean) to recover the parameters of each individual. For an introduction to statistical and computational inverse problems, we refer to the book [10].

This paper is mainly concerned with population approach to make the parameters estimation. More precisely, we concentrate on the use of the SAEM-MCMC algorithm [3,12] which has proved to be very efficient to estimate population parameters of complex *mixed effects models* defined by ODE [15,19] or SDE [4,5]. Still, this algorithm was never extended to the use with PDEs and this is the goal of present article.

The essential reason for which PDEs were not used in SAEM algorithm is it involves many calls to the PDE solver, and thus this leads to prohibitive computational times in practice. To alleviate this problem, we propose to couple the SAEM algorithm with an interpolation of pre-computed solutions of the PDE model.

Of note, this PDE pre-computation strategy can also be used in the context of "individual" approaches since they also involve a lot of calls to the PDE solver. In the context of population data, the combination of traditional SAEM algorithm and pre-computed databases leads to a new computational method.

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. Of course, the sole idea of pre-computing solutions of a complex problem, storing them and using them in conjunction with interpolation to obtain fast approximations of solution is an old and simple idea. There must have a lot of unpublished works using this idea but, to our knowledge, it is almost essentially found in the electronic engineering literature (see *e.g.* [2,8,22]). In this paper, we describe this interpolation approach both taking into account the qualitative behavior of the function to be stored as well as implementation choices in a multiprocessing context.

This general framework coupling SAEM and PDEs is then illustrated with the KPP equation, a classical model used for reaction-diffusion processes and for associated propagation phenomena (traveling waves). Applications we have in mind are the following. 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 a 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 KPP equation (with tumoral concentration as the unknown), and the tumour volume is the space 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 of the evaluation of the scalar time series associated to the solution of the full PDE. This is where the pre-computation philosophy mentioned above comes into play. We couple a SAEM algorithm with evaluation of the model through interpolation on a pre-computed mesh of the parameters domain.

This appears to be already very efficient for 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. This is another interesting result of present paper since it brings new insights on identifiability questions for this PDE which are unveiled thanks to population approach.

This paper is organised as follows. In Section 2, we present in more details so called "mixed effects models" in the context of population approach for parameters estimation. We recall essential ideas of the SAEM algorithm

and we present how we couple it with a pre-compuputation strategy to allow its use with PDEs. We then apply this new algorithm to the KPP equation and present the results in Section 3. Finally, we draw some further remarks and perspectives of improvements for this SAEM-PDE approach in Section 4, before concluding the article.

2. SAEM coupled with pre-computation, for expansive models

2.1. Principle of population approaches

Let us first recall the principles underlying population approaches and mixed effects models (we refer to [14] for a more detailed description). 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 population level. More precisely, let us consider a model

$$y = f(t, Z)$$

where y is the observable, t the time, and Z the individual parameters. The model f may be algebraic, 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 are stochastically independent and follow a normal distribution law, with mean 0 and standard deviation ε_σ.

Let us consider N individuals and for each individual, 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 $\#i$). We get measures y_{ij} such that

$$y_{ij} = f(t_{ij}, Z_i) + \varepsilon_{ij}, \tag{2.1}$$

where

$$\varepsilon_{ij} \sim \mathcal{N}\left(0, \varepsilon_\sigma^2\right).$$

Model (2.1) will lead to so called "mixed effects model" since we will take into account so called "fixed" effects (mean population parameters) and "random" effects (to describe the variability between individuals of the population).

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

$$p\Big((y_{ij})|Z_i\Big) = \frac{1}{(\varepsilon_\sigma \sqrt{2\pi})^{N_i}} \prod_{j=1}^{N_i} \mathrm{e}^{-\frac{1}{2}\frac{(y_{ij} - f(t_{ij}, Z_i))^2}{\varepsilon_\sigma^2}}.$$

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

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

is then solution of the classical nonlinear regression problem

$$Z_i^b = \operatorname{argmax} \prod_{j=1}^{N_i} \mathrm{e}^{-(y_{ij} - f(t_{ij}, Z_i))^2}.$$

However in many cases, 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 population 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

$$Z_i \sim \mathcal{N}(\theta_m, \theta_\sigma^2)$$

where θ_m is the mean of the individual parameters θ_i in the population, and θ_σ^2 their variance. Let

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

be the set of all population characteristics.

In a population 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 population approach. The probability that the individual i has characteristics Z_i is

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

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

$$p\Big((y_{ij})_j, Z_i|\theta_{\text{pop}}\Big) = \frac{1}{\theta_\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}\frac{(Z_i-\theta_m)^2}{\theta_\sigma^2}} \frac{1}{(\varepsilon_\sigma\sqrt{2\pi})^{N_i}} \prod_{j=1}^{N_i} \mathrm{e}^{-\frac{1}{2}\frac{(y_{ij}-f(t_{ij},Z_i))^2}{\varepsilon_\sigma^2}}.$$

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

$$\tilde{p}\Big((y_{ij})_{ij}, (Z_i)_i|\theta_{\text{pop}}\Big) = \frac{1}{(\theta_\sigma\sqrt{2\pi})^N} \prod_{i=1}^{N} \left[\mathrm{e}^{-\frac{1}{2}\frac{(Z_i-\theta_m)^2}{\theta_\sigma^2}} \frac{1}{(\varepsilon_\sigma\sqrt{2\pi})^{N_i}} \prod_{j=1}^{N_i} \mathrm{e}^{-\frac{1}{2}\frac{(y_{ij}-f(t_{ij},Z_i))^2}{\varepsilon_\sigma^2}} \right].$$

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

$$g\Big((y_{ij})_{ij}|\theta_{\text{pop}}\Big) = \int \tilde{p}\Big((y_{ij})_{ij}, (Z_i)_i|\theta_{\text{pop}}\Big)\mathrm{d}Z_1 \ldots \mathrm{d}Z_n.$$

The log likelihood of θ_{pop} is then

$$l(\theta_{\text{pop}}) = \log g\Big((y_{ij})_{ij}|\theta_{\text{pop}}\Big).$$

The usual approach leads to the search of $\theta_{\text{pop},N}$ which is an estimator of θ_{pop} and maximizes l:

$$\theta_{\text{pop},N} = \text{argmax}_\theta l(\theta). \tag{2.2}$$

2.2. SAEM algorithm

To solve (2.2) 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. In this paragraph we follow the approach of [3].

Namely instead of trying to maximize l we focus on

$$Q(\theta|\theta') = \int \log \tilde{p}\Big((y_{ij})_{ij}, (Z_i)_i|\theta\Big) p\Big((Z_i)_i|(y_{ij})_{ij}, \theta'\Big)\mathrm{d}Z_1 \ldots \mathrm{d}Z_N \tag{2.3}$$

where,

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

g being the normalization function

$$g((y_{ij})_{ij}|\theta) = \int \tilde{p}\Big((y_{ij})_{ij}, (Z_i)_i|\theta\Big)\mathrm{d}Z_1 \dots \mathrm{d}Z_N.$$

EM algorithm consists in building the following sequence

$$\theta_{k+1} = \mathrm{argmax}_\theta \; Q(\theta|\theta_k). \tag{2.4}$$

As we will see below, this problem is much easier to solver numerically.

It remains to compute $Q(\theta|\theta_k)$. For this we use a stochastic approximation (SA) of the integral, namely a Monte Carlo Markov Chain (MCMC) approach in order to get a sequence $(Z^{kl})_l$ of points, with distribution law $p((Z_i)_i|(y_{ij})_{ij}, \theta_k)$. This is easily done since the computation of $\tilde{p}((y_{ij})_{ij}, (Z_i)_i|\theta_k)$ is explicit. Note that $(Z^{kl})_l$ is a population and is composed of N individuals Z_i^{kl}. We then approximate $Q(\theta|\theta_k)$ by (we note $\theta = (\theta_m, \theta_\sigma)$)

$$\begin{aligned}\tilde{Q}(\theta|\theta_k) = &-\frac{N}{2}\log(\pi\theta_\sigma) - \frac{\sum_{1\le i\le N} N_i}{2}\log(\pi\varepsilon_\sigma)\\ &-\sum_l \sum_{1\le i\le N} \frac{(Z_i^{kl}-\theta_m)^2}{2\theta_\sigma^2} - \sum_l \sum_{1\le i\le N}\sum_{1\le j\le N_i} \frac{\left(y_{ij} - f\left(t_{ij}, Z_i^{kl}\right)\right)^2}{2\varepsilon_\sigma^2}.\end{aligned}$$

Note that the optimization step (2.4) is explicit: θ_{k+1} is explicit using the expression of $\tilde{Q}$, which is quadratic in θ.

Moreover, Theorem 1 of [3] shows that SAEM algorithm converges in the following sense

$$\lim_{k\to+\infty} \mathrm{d}(\theta_k, \mathcal{L}) = 0$$

where

$$\mathcal{L} = \{\theta|\partial_\theta l(\theta) = 0\}$$

is the set of critical points of l.

SAEM appears to be very efficient and is widely used in applied and industrial problems, in particular in pharmacokinetics-pharmacodynamics (PKPD) problems (see [5, 15, 19] or [4], for a few examples). However it is not possible to design fully parallel versions of SAEM, and SAEM is very long if the evaluation of the model f is time consuming. The aim of this article is to couple SAEM with a parallel pre-computation step.

2.3. Pre-computation revisited

To speed up the evaluation of the model f, a classic and easy approach is to use already computed values of the model, stored in a grid or a mesh (see *e.g.* [2, 8, 22]). Then, using interpolation methods, which are very fast, approximate values of the model can be inferred quickly, without new time consuming evaluations. If the approximation is good enough, then an evaluation of the model through interpolation may be sufficient. And only when needed, *i.e.* when the approximation is poor, one can then improve it by adding more pre-computed values (and pay the price of time costly evaluations).

In this paper, we describe this interpolation approach both taking into account the qualitative behavior of the function to be stored as well as implementation choices in a multiprocessing context, issues not presented in the references mentioned just above. The pre-computation can be done on arbitrary meshes, or on structured meshes of the bounded parameters space. 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.

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

- *once for all pre-computation:* the model is computed on a mesh before parameter identification procekdure. 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 with the values at points P_i, which is very fast. The identification procedure can be done very quickly.
  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 pre-computation. Mesh refinement will be done according to some criterion or score, which must be carefully chosen. This will be described in the following of this section.
- *interactive refinement:* during the identification procedure, we interpolate the model at (evolving) estimated set of parameters. If the interpolation quality is too bad, we need to compute the precise value of the model at this set of parameters (which is time consuming but more precise) and we can add it to the pre-computed values of the model, leading to a better data-base for the interpolation. A natural idea is then to run population algorithm for a while, then switch to pre-computation refinement in the areas which are explored by the population algorithm, and iterate. We will go back to this approach in Section 4.

The main issue is then 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).

2.3.1. Pre-computation algorithm

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)

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

to prescribe the area of search.

The algorithm is iterative. At step n, we have $1 + 2^N n$ cubes C_i with $1 \leq i \leq 1 + 2^N n$, organized in a tree. To each cube we attach J different weights ω_i^j (where $1 \leq j \leq J$, 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.

2.3.2. Parallelisation of the pre-computation algorithm

The above algorithm may be parallelized in various ways. 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 previous section.

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, correct load balancing 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 $(N_{\text{proc}} - 1)$ "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).

2.3.3. 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

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

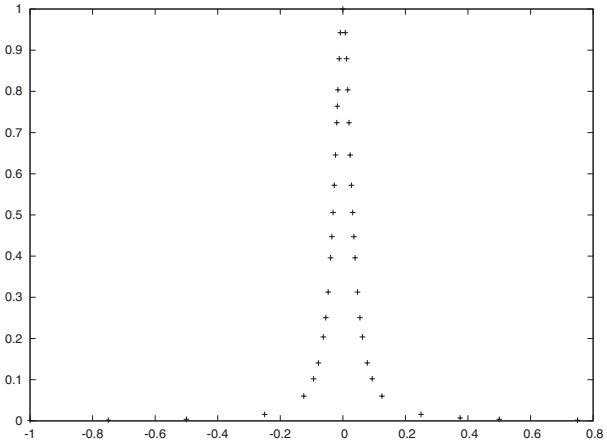

FIGURE 1. Example in one space dimension.

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 used for the required precisions. In some applications it is important to evaluate accurately f in some areas of C_{init}, whereas crude approximations are sufficient in other areas of C_{init}. Let us give an example. Let $\phi(x)$ be some positive function (weight), with $\int \phi(x)\mathrm{d}x = 1$. To evaluate $\int_{C_{\text{init}}} \phi(x)f(x)\mathrm{d}x$ with a precision δ it is sufficient to evaluate $f(x)$ with a precision $\delta/|C_{\text{init}}|\phi(x)$. In this case, we define

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

and consider $\omega_i^{\text{vol}} = \text{vol}(C_i)$.

2.3.4. 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

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

With the weight ω_i^{∞}, the mesh (see Fig. 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 Fig. 2) is refined near these two axes with weight ω_i^{∞}.

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

2.3.5. 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,

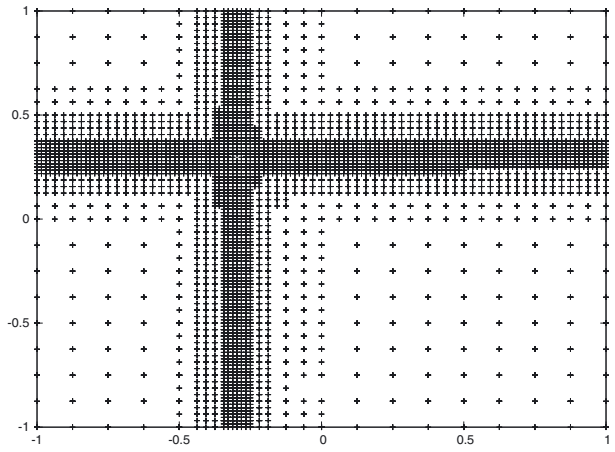

FIGURE 2. Product of tangents.

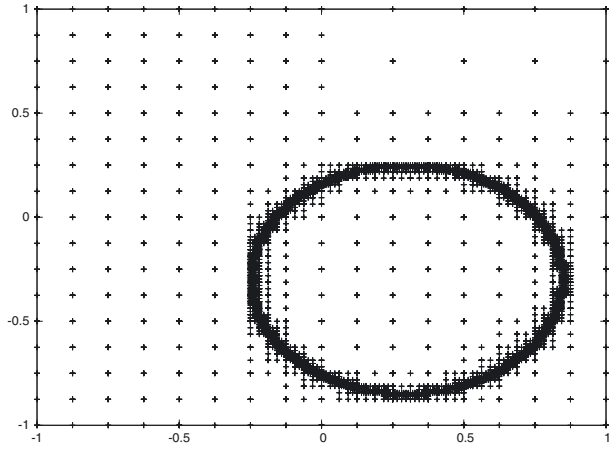

FIGURE 3. Characteristic function of a disk.

up to a numerical error δ_{num}. In our method we approximate f by interpolations of approximate values of f. The global error δ of our method is therefore the sum of two terms

$$\delta = \delta_{\text{num}} + \delta_{\text{interp}}.$$

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

  $$\delta_{\text{num}} = C_1 h^\beta$$

  for some constant β. The number of cells in the mesh is of order h^{-d} where $d = 1, 2, 3$ is the physical dimension. The number of time steps is of order $h^{-\alpha}$, hence the computational cost of an evaluation is

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

The constant C_2 depends on the parameters 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 δ_{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 δ, what is the best decomposition of δ into δ_{interp} and δ_{num}?

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

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

Let us define δ by $\delta_{\text{num}} = \eta\delta$. Then

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

This time is minimal provided

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

which gives

$$\delta_{\text{num}} = \frac{d+\alpha}{d+\alpha+\beta N}\delta$$

as an optimal choice.

2.3.6. 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 δ we need to refine the mesh until the size of the sub-cubes is less than $\delta/\|\nabla f\|_{L^\infty}$, which leads to approximately

$$N(\delta) = \left(\frac{\|\nabla f\|_{L^\infty}}{\delta}\right)^N$$

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

$$T(\delta) = \frac{T_m}{N_{\text{proc}}} \left(\frac{\|\nabla f\|_{L^\infty}}{\delta}\right)^N.$$

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

	$N_{\text{proc}} = 1$	$N_{\text{proc}} = 8$	$N_{\text{proc}} = 128$	$N_{\text{proc}} = 1024$	$N_{\text{proc}} = 10\,000$
$N = 1$	16 mn	2 mn	8 s	1 s	–
$N = 2$	4h30	32 mn	2 mn	15 s	2 s
$N = 3$	3 days	8.5 h	32 mn	4 mn	25 s
$N = 4$	1.5 month	5.6 d	8.5 h	1 h	6 mn
$N = 5$	–	3 m	5.6 d	17 h	1.7 h
$N = 6$	–	–	3 m	11 d	1.1 d
$N = 7$	–	–	–	6 m	18 d

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

	$N_{\text{proc}} = 1$	$N_{\text{proc}} = 8$	$N_{\text{proc}} = 128$	$N_{\text{proc}} = 1024$	$N_{\text{proc}} = 10\,000$
$N = 1$	32 mn	4 mn	16 s	2 s	–
$N = 2$	17 h	2 h	8 mn	1 mn	6 s
$N = 3$	22 d	3 d	4h30	32 mn	3.3 min
$N = 4$	–	3 m	5.6 d	17 h	1.7 h
$N = 5$	–	–	6 m	22 d	2.3 d

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 Section 4.

2.4. The direct coupling between pre-computation and SAEM

This is the easiest and natural way. First, one runs the pre-computation step, in order to get a mesh with a given interpolation precision. Its output is an approximation of the model f_{app} through grid interpolation. The SAEM step is then done with the approximate model f_{app}.

Let us consider a population with N individuals and parameters θ_{pop}. If we apply SAEM algorithm on this sample, we get an estimator $\theta_{\text{pop},N}$ of the population parameters. Moreover, when $N \to +\infty$, we expect $\theta_{\text{pop},N} \to \theta_{\text{pop}}$ (see [17] for a result in this direction; see also [18]).

If we apply our algorithm, we get population parameters $\theta^\star_{\text{pop},N}$ which optimize the likelihood for the approximate model f_{app}. As the approximation error goes to 0 (*i.e.* $f_{\text{app}} \to f$), we expect that $\theta^\star_{\text{pop},N}$ converges to $\theta_{\text{pop},N}$ [13]. Therefore the difference between $\theta^\star_{\text{pop},N}$ and θ_{pop} comes from the combination of an interpolation error and the fact that the sample is finite. The convergence of this modified SAEM algorithm would deserve further theoretical studies, which are out of scope of this paper.

Let us conclude this first Section with a few remarks on the SAEM-PDE algorithm. The pre-computation step is long but can be parallelized very efficiently. Note that the pre-computations 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. With such coupling that allows a feasible computation of parameters estimation *via* mixed effects models with PDEs, it opens interesting applications with real data. As a matter of fact, for one individual, we often only have a few data in time. For these sparse data, an individual inverse problem approach will potentially give poor results or no result at all. On the contrary, SAEM allows to take into account all the population data to make the estimation. This gives more opportunity to increase the accuracy of estimated parameters for each individual of the population.

3. Application: parametrization of a KPP model with monolix

We want to illustrate the previous methodology in the context of the estimation of the parameters 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 [11]. Such kind of equation is also sometimes referred to as the Fisher equation, introduced in the context of the theory of evolution [6]. 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 log-normal distributions on its parameters, and adding noise. We then try to recover the distributions of the parameters by a SAEM approach (using Monolix software [21]). 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.

3.1. Presentation of the problem

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

$$\partial_t u - \nabla.(D\nabla u) = Ru(1-u), \tag{3.1}$$

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

$$u(T_0, x) = \alpha 1_{|x-x_0|\leq\varepsilon}, \tag{3.2}$$

for some time T_0 (in the past). It is well-known that for (3.1) and (3.2), there exists 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

$$S(t) = \int_\Delta u(t,x)\mathrm{d}x$$

or by

$$S(t) = \mathrm{vol}\,(u \geq \sigma)\,,$$

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

We assume that we have data from a population of individuals at various times $t_1, \ldots, t_{N_t}$. For an individual, let $S_1, \ldots, S_{N_t}$ be these data. If we want to compare aforementioned KPP model with data, we have to look for solutions of (3.1) with initial data (3.2) such that $S(t_i)$ is close to S_i for $1 \leq i \leq N_t$.

As a first approximation, α and ε may be fixed to given values (*e.g.* $\alpha \leq 1$, $\varepsilon \ll 1$)[3]. It remains to find x_0, D and R such that

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

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 are interested in a population approach to parametrize the model. Namely, we are concerned in the distribution of the model parameters in the population which maximize the likelihood of the observations and want to look for the mean and standard deviation of each parameter in the population, *e.g.*

$$D \sim \mathcal{N}\left(\bar{D}, D_\sigma^2\right)$$

(normal distribution with mean $\bar{D}$ and variance D_σ^2) and similarly for R and x_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.

[3]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.

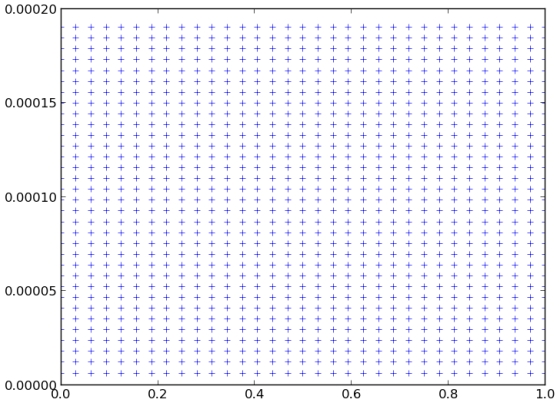

FIGURE 4. An example of a uniform mesh of the space of parameters (fifth uniform splitting). The two parameters are $w = D/R$ and x_0.

3.2. 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 to fully describe the initial position of the invaded zone, namely (x_0, y_0, z_0) [with a small abuse of notation here, since y_0 and z_0 are spatial coordinates and not observations, for instance].

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 simulations, 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 cost 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 Fig. 4) on two cores (one master and one worker) of a quad-core AMD Opteron (2.7 GHz) and 40 h 31 mn 10 s for the eighth splitting (4^8 cubes). Of course, explained previously, 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 h 3 mn 18 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.

Approximate evaluation of the model through interpolation is very fast (far below 1 s). As a consequence, a Monolix run lasts typically 10 min 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 a classical inverse problem approach. We detail a few specific examples in the next section.

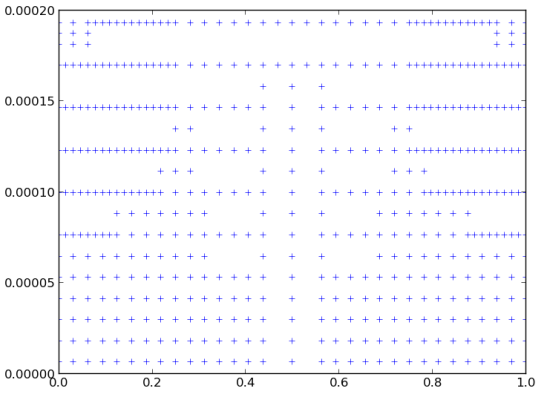

FIGURE 5. An example of an inhomogeneous mesh of the space of parameters (with 500 points). The two parameters are $w = D/R$ and x_0. We can see that the finer zones have a smaller size than the size the homogeneous mesh of Figure 4 (which contains 1089 points): the mesh is here more refined in zones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

3.3. 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 (3.1) and (3.2) 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 $\{t_{i,j}, S_{i,j}\}$, $i = 1 \dots 101$, $j = 1 \dots 100$. These series will be given to Monolix as data to perform the parameters 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 Fig. 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 parameters estimation (Test 4, below) with a 500 points grid (see Fig. 5) built with the "BV" weight (see Sect. 2.3.3).

3.3.1. 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 [21] 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 Tab. 1, column "E1") and the mean "theoretical" parameters used to build the population (see Tab. 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 Fig. 7). In

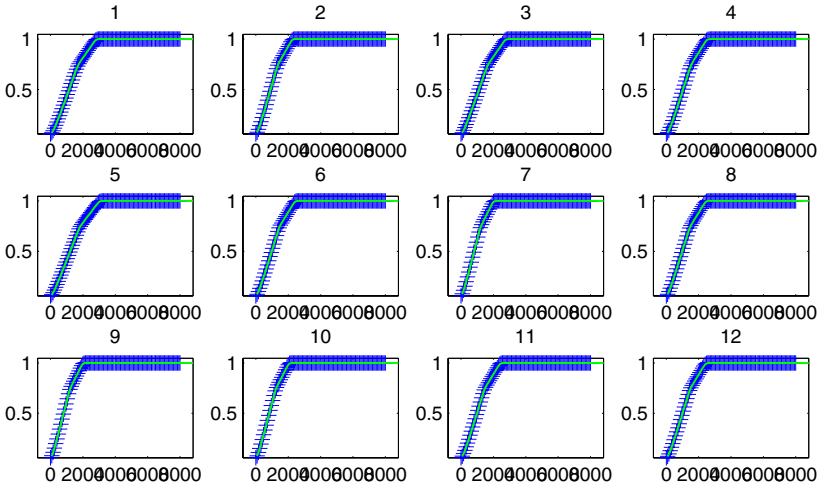

FIGURE 6. Test 1: 12 individuals of the population (in blue) and the corresponding estimated curves (in green) computed by Monolix using the KPP model.

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.

	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%

addition the results of Table 1 can be illustrated by the Monolix output of observed *vs.* predicted data, both for the population and individual data (see Fig. 8, left and right respectively).

Let us note that the convergence graphs of the SAEM (Fig. 7) and the comparison of population and fitted data (Fig. 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 *vs.* predicted" graphs.

Then, we performed the same run but with a population of individuals who are perturbed with a random noise of amplitude 5% (*i.e.* since the data are of order 1, the noise is of order 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 of order 1, the noise is of order 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 *vs.* predicted data are shown on Figure 10.

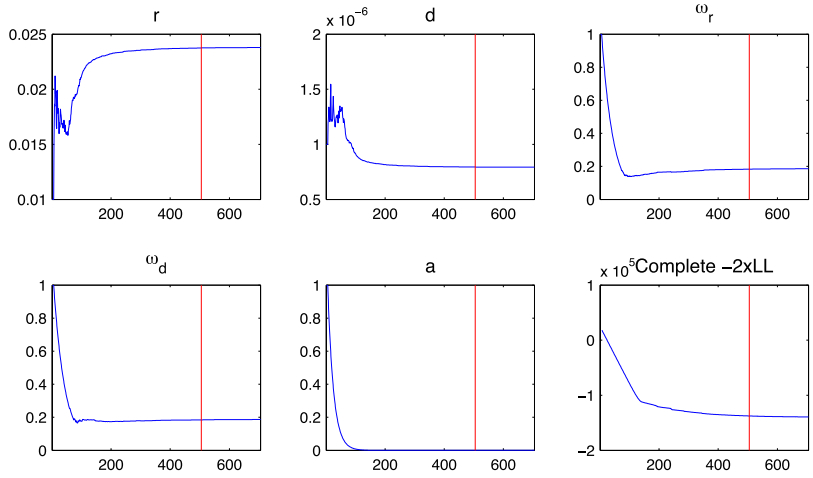

FIGURE 7. Test 1: Graphs showing the convergence of the SAEM algorithm in Monolix. Case with no noise.

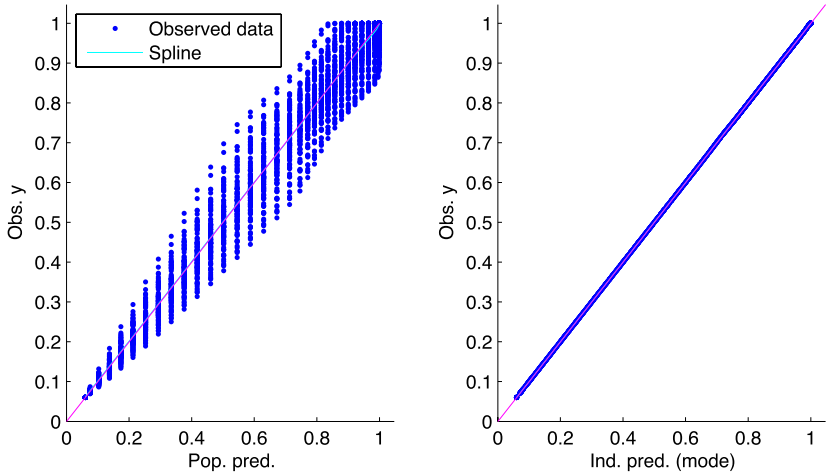

FIGURE 8. Test 1: monolix output of observed *vs.* predicted data both for the population and individual data. Case with no noise.

3.3.2. 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%).

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 population parameters, but the individual parameters are not the same. Dispersion associated to the noise can be seen by comparing Figures 11, 12 and 13.

3.3.3. 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%).

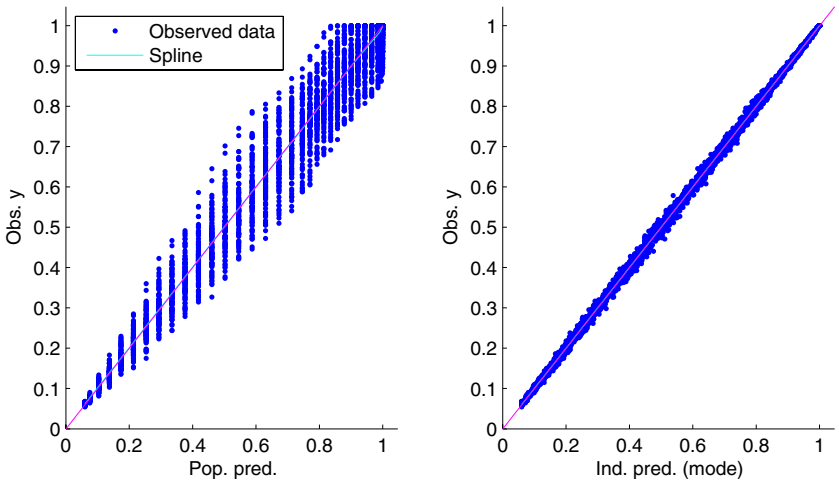

FIGURE 9. Test 1: monolix output of observed *vs.* predicted data both for the population and individual data. Case with 5% noise.

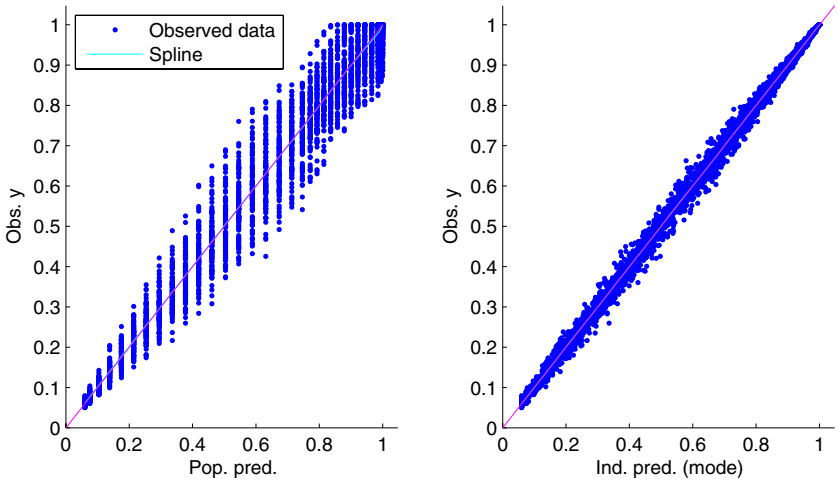

FIGURE 10. Test 1: monolix output of observed *vs.* predicted data both for the population and individual data. Case with 10% noise.

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.

	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%

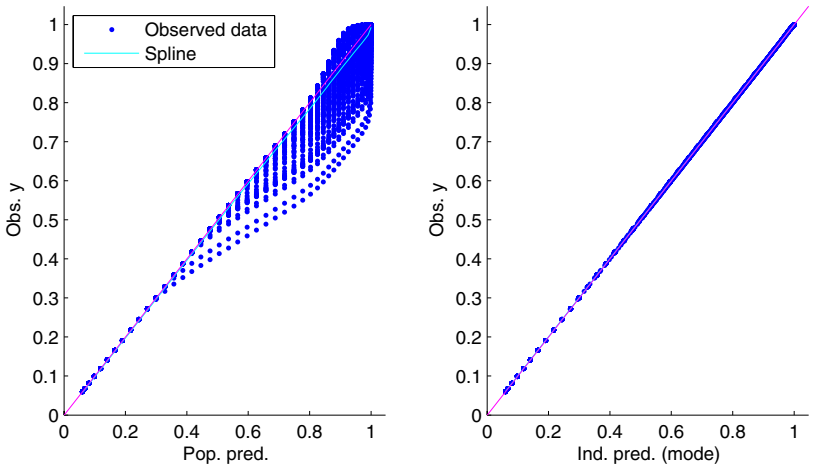

FIGURE 11. Test 2: monolix output of observed *vs.* predicted data both for the population and individual data. Case with no noise.

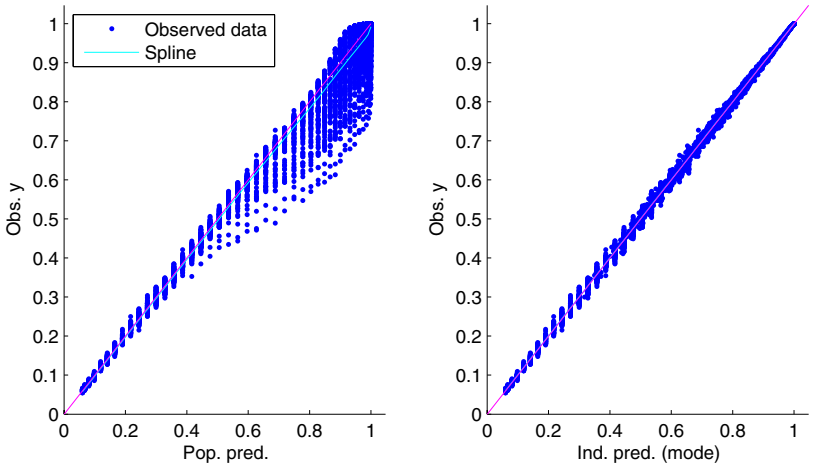

FIGURE 12. Test 2: monolix output of observed *vs.* predicted data both for the population and individual data. Case with 5% noise.

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.

3.3.4. 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 the interpolation on the

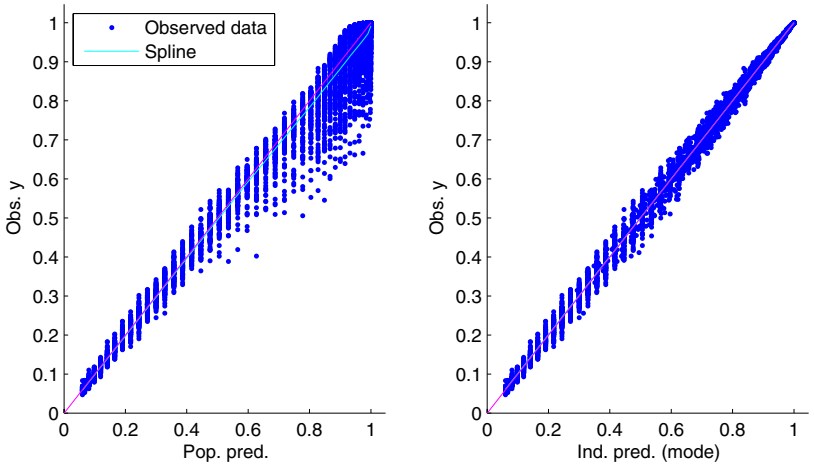

FIGURE 13. Test 2: monolix output of observed *vs.* predicted data both for the population and individual data. Case with 10% noise.

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.

	Theor.	E1		E2		E3	
			error		error		error
R	0.0245	0.0237	–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%

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.

3.4. Computational cost comparison

The main interest of this methodology is to tackle problem of parameters identification in complex PDE systems. The gain in terms of computational cost can be easily evaluated and illustrates the feasibility of the method for a large range of problems. The computational cost of the whole algorithm (*i.e.* generation of the mesh + SAEM computation) can be divided in two distinct parts: an offline time corresponding to the computation of the mesh (which can be done once for all) and an online time corresponding to the estimation of the parameters for a given population.

Table 5 illustrates these different times and shows the gain induced by the method. In particular, the exact case refers to the SAEM algorithm solving the full PDE everytime it is required.

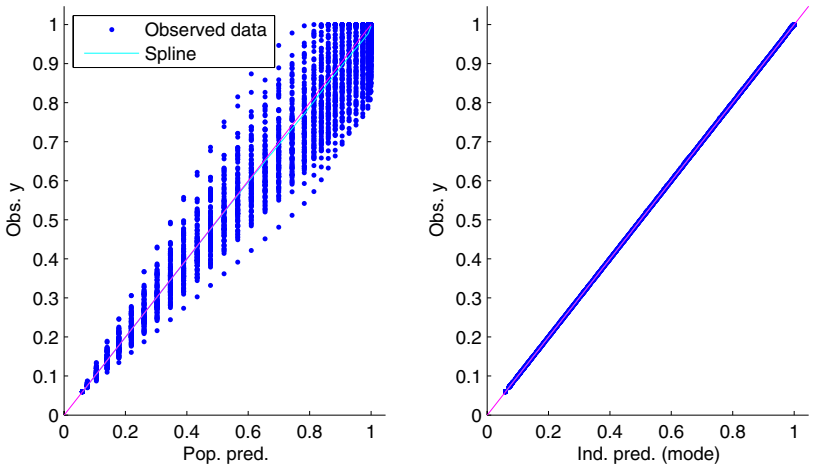

FIGURE 14. Test 3: monolix output of observed *vs.* predicted data both for the population and individual data. Case with no noise.

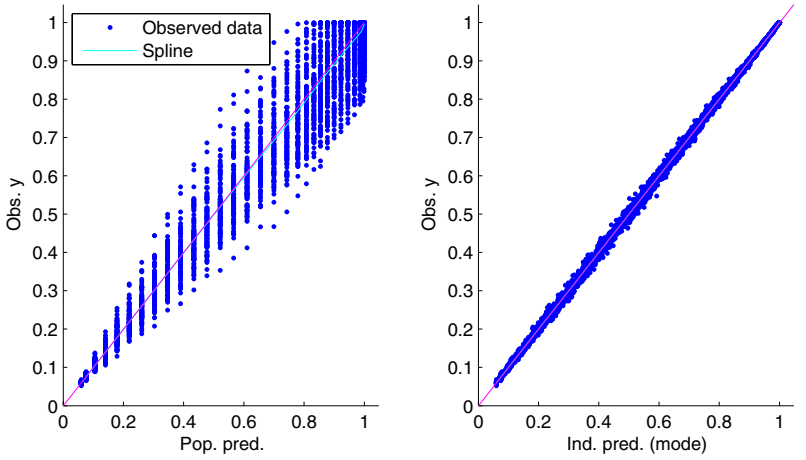

FIGURE 15. Test 3: monolix output of observed *vs.* predicted data both for the population and individual data. Case with 5% noise.

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.

	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%

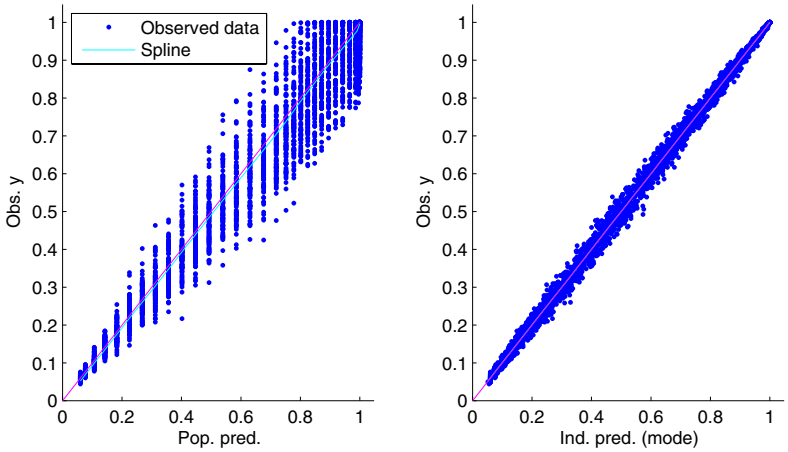

FIGURE 16. Test 3: monolix output of observed *vs.* predicted data both for the population and individual data. Case with 10% noise.

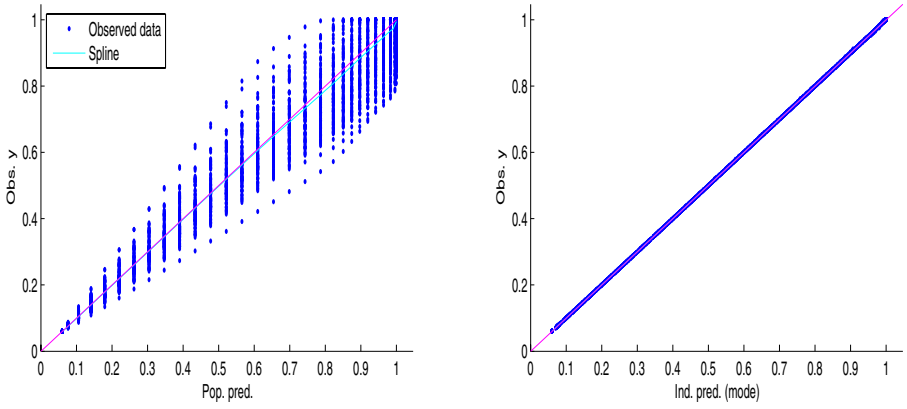

FIGURE 17. Test 4: monolix output of observed *vs.* predicted data both for the population and individual data. Case with no noise.

The considered case is the one explained in paragraphs 3.3.3 (and 3.3.3): estimation of x_0, R and D for a population of 100 individuals.

3.5. Statistical issues

All previous tests were done with only one population and all the errors are computed with this unique population. Of course from the statistical viewpoint, for each test (Tests 1–4), we should have repeated the same study for many randomly generated populations. In this section, we adress this issue.

It is important to note that when we choose randomly the population of 100 individuals for Tests (1–4), we repeated the random generation of parameters to obtain a distribution which follows as much as possible a log-normal distribution (recall that this law is assumed by the SAEM algorithm when it looks for the parameters). This is due to the fact that when there is only 100 individuals, a random generation of parameters can lead to a distribution which is not really close to a log-normal one, as illustrated in Figure 20. The fact that the

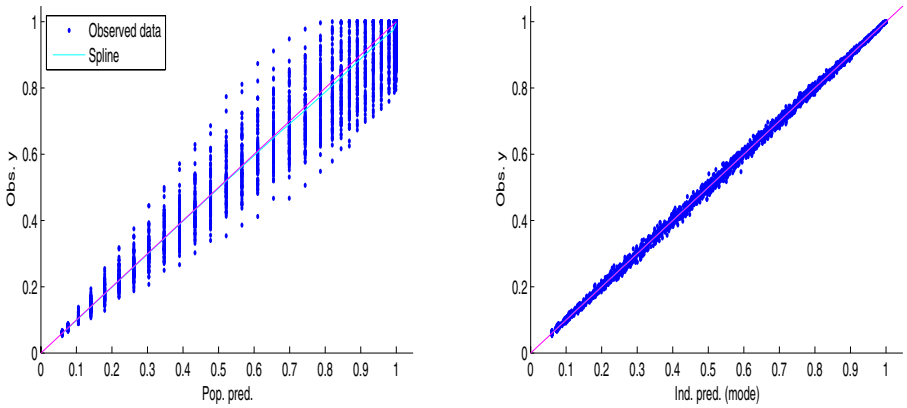

FIGURE 18. Test 4: monolix output of observed *vs.* predicted data both for the population and individual data. Case with 5% noise.

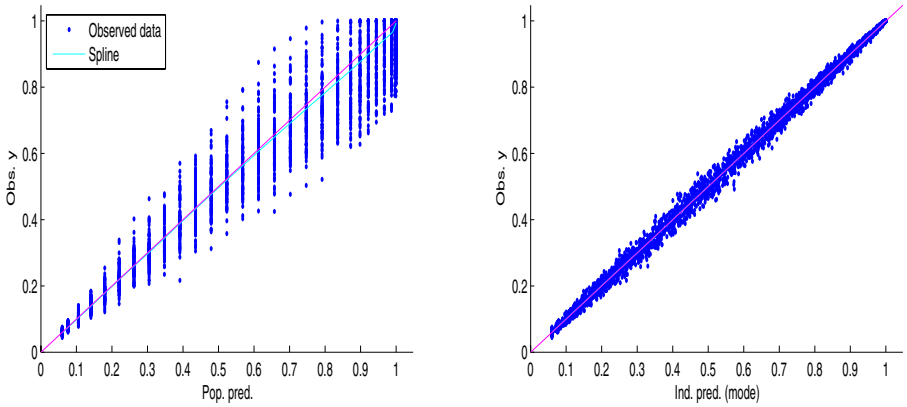

FIGURE 19. Test 4: monolix output of observed *vs.* predicted data both for the population and individual data. Case with 10% noise.

distribution is closer to a log-normal one improves the results of the SAEM algorithm, as we will see in the following.

Here, we perform two studies, each of them with repeated simulations to obtain averaged errors which are more statistically sound.

- The first study has exactly the same characteristics as in Test 3 (Sect. 3.3.3), except that instead of using one population, we use repeated simulations with 120 populations. The presented errors are thus averaged with these 120 realizations. For this study, since the populations are generated automatically, we have various types of generated distributions: some are close to a log-normal one, some are not (as in Fig. 20).
- The second study is like the first one except that, instead of taking 100 individuals in a population, we build a population with 1000 individuals. The goal is here to ensure that distributions of the parameters are all much more close to a log-normal distribution than in the first study. Of note, since with 1000 individuals per population, the CPU time for simulations is longer, we made 92 realizations instead of 120.

Let us describe in more details these two studies.

The first approach consists in generating 120 populations of 100 individuals in order to evaluate the behavior of the methodology from a better statistical viewpoint. The parameters for each of these populations are drawn

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 parallelized on many cores with an excellent scalability.

	"Exact" case	Interpolation with homogeneous mesh	Interpolation with heterogeneous mesh
Offline	No offline computation	Mesh with n levels of refinement (l.o.r),$(2^n+1)^2$ points. For 5 l.o.r, 1089 points	Mesh with n points. Example with 500.
Unit average CPU cost	–	2.12 s	2.12 s
Offline total CPU cost	–	38 mn28 s	17 mn40 s
Online	SAEM, 10^6 KPP evaluations	SAEM, 10^6 interpolations	SAEM, 10^6 interpolations
Unit average CPU cost	2 s	4.5×10^{-4} s	5.1×10^{-4} s
Online total Cost	∼23 days 3 h	7 mn30 s	8 mn30 s
Total cost	∼23 days 3 h	45 mn58 s	26 mn10 s

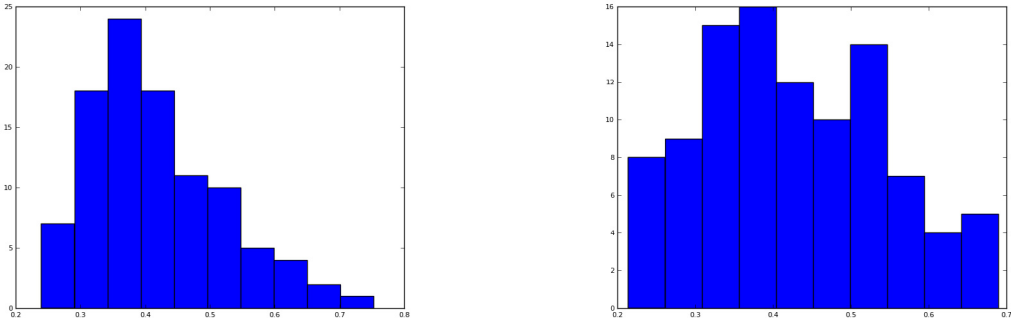

FIGURE 20. Distributions for parameter x_0 which can be obtained with a population with 100 individuals. (Left) a "good" one (Right) a "bad" one.

TABLE 6. Test for 120 populations of 100 individuals: results (from Monolix) and errors for the *mean parameters of the population*. Column E1 refers to a population without noise. Column E2 (resp. E3) refers to a population with a 5% (resp. 10%) noise.

	E1 error	E2 error	E3 error
R	58.6%	46.1%	47.5%
D	26.5%	22.5%	23.5%
x_0	20.9%	19.2%	17.0%

from a random log-normal distribution. None of these parameters are fixed as in Section 3.3.3. We perform the same run with populations perturbed with a random noise of amplitude 5% and 10%. The previously described interpolation method on a homogeneous grid is used to evaluate population and individual parameters with the Monolix software.

The averaged errors of the *mean parameters of the population* are computed for each population and we indicate in Table 6 the mean of those errors.

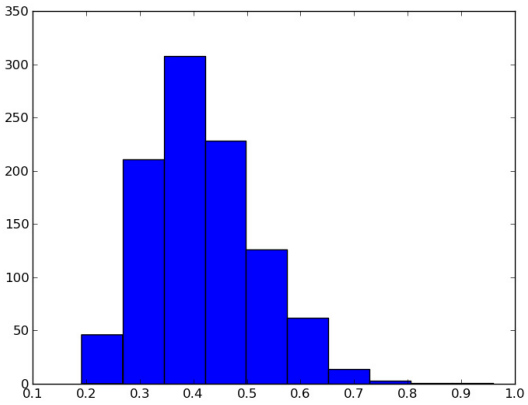

FIGURE 21. Distributions for parameter x_0 for a 1000 individual population.

TABLE 7. Test for 92 populations of 1000 individuals: results (from Monolix) and errors for the *mean parameters of the population*. Column E1 refers to a population without noise. Column E2 (resp. E3) refers to a population with a 5% (resp. 10%) noise.

	E1 error	E2 error	E3 error
R	10.0%	9.44%	8.96 %
D	12.0%	11.1%	10.7%
x_0	6.03%	5.52%	4.44%

We see that the quality of the results is not as good as in Test 3. This is due to the quality of the distribution of the generated parameters. As a matter of fact, when looking closely to each population, we see that with "good" log-normal distribution of the parameters (in the sense given above), estimation performed by Monolix has a quality which is of the same order as in Test 3. Whereas for a population with a "bad" distribution of generated parameters, the estimation is of a poorer quality.

To go further into this analysis, we performed the second study. We designed another numerical experiment with 92 virtual populations of 1000 individuals (instead of 100), in order to obtain a better drawn log-normal distribution for the parameters, as confirmed by Figure 21. Let us note that all of 92 populations have distributions of parameters which are close to log-normal one. The conditions of the study are exactly the same as for the first study. Table 7 summarized the averaged errors of the mean parameters of population.

We now see that the results are of significantly improved quality, compared to the first study. Furthermore they are very close to the quality obtained in Test 3.

Before closing this section, it is worth noting that parameters of the KPP model, as treated in present framework are fully identifiable. As a matter of fact, we consider the problem of the full invasion of a spatial domain Δ: in terms of the evolution of the volume of the invaded zone (*i.e.* the observed data), the typical curve is growing from 0 to 1 and then saturates at 1 (when no noise is added to the solution of KPP). This type of curve allows to identify all the parameters: x_0 is recovered by looking at the time of transition between the linear growth regimes. The slope of the growth regimes gives the speed of the travelling wave, while the duration of the transition between these regimes gives the width of the front: as a consequence one can recover R and D since the speed (resp. the width) is proportional to $\sqrt{RD}$ (resp. $\sqrt{D/R}$).

To conclude, provided that the parameters to be estimated follow closely a log-normal distribution, the previous studies tend to show that the coupled SAEM-Precomputation algorithm leads to a rather good level of accuracy to estimate the parameters *via* a population approach.

4. Perspectives

In this section, we sketch some perspectives of improvement for the coupled algorithm. We begin by some remarks on the design of the pre-computed grid. We then present another version of the coupling between pre-computation and SAEM. These points are under developement and will be presented in a forthcoming work.

4.1. Remarks on the pre-computed grid

We saw in previous sections that the size of the grid quickly increases, in association with the canonical curse of dimensionality. It is thus useful to use as much as possible any information that can be obtained on the function f to be pre-computed, to decrease the size of the grid. In the following, we give some sources of improvements in this direction.

4.1.1. 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 $D \subset C_{\text{init}}$, 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 δ we will need

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

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.

4.1.2. 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 do not need to compute all the summits of the 2^N sub-cubes. This is equivalent to the gain of one dimension.

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

4.1.4. Summary

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

To go above these dimensions, another approach can be investigated: iterative refinements of the grid *via* the knowledge of the parameters to be explored, as described in the following section.

4.2. Iterations between pre-computation and SAEM

The main drawback of the direct coupling presented in Section 2.4 is that the pre-computation step will mesh the whole parameters domain, whereas SAEM algorithm will concentrate on particular areas of the individual parameters. Most of the pre-computations will therefore be useless, and it is more efficient to concentrate the pre-computation where SAEM requires them. In this paragraph, we sketch a possible interaction of pre-computation and SAEM.

First, we choose some initial precision δ_{initial} and run the pre-computation step with this initial precision. We get a first model approximation f_1, such that $|f - f_1|$ is of order δ_{initial}. We then run SAEM algorithm, which converges to some approximation θ_1 of the population parameters θ_{pop}. SAEM algorithm also gives the conditional probabilities for each individual parameter. Hopefully, the individual parameters only explore a reduced part of the complete parameters space. The idea is to improve the accuracy of the model evaluation in this particular area.

For this we choose $\delta_2 < \delta_{\text{initial}}$ and run again the pre-computation step with this precision in the desired area. This gives a second model approximation f_2, which is better than f_1 in the area of interest.

We then run SAEM again, using f_2 and starting from θ_1. This converges to new population parameters θ_2, and new individual conditional probabilities.

We can then iterate this process. Doing this, we decrease the model interpolation error. This algorithm deserves further studies and implementation to explore its convergence properties.

5. Conclusion

In this paper a new method combining SAEM algorithm and a pre-computation step has been presented. In the context of parameters estimation, this new algorithm is helpful to reduce the overall computation time when the model is very long to compute. This is the case when the model is based on a huge number of ordinary differential equations, on partial differential equations, or on multi-physics systems of PDEs.

The algorithm was applied for the estimation of parameters in the framework of a population approach with mixed effects model. The SAEM algorithm from Monolix was coupled with a pre-computed grid for the classical KPP model. To our best knowledge, this is the first demonstration of parameters estimation of PDE thanks to a SAEM algorithm. It was shown that, provided that parameters distribution to be estimated is of the same structure as the one assumed by the SAEM algorithm (say log-normal), the quality of estimated parameters is good. Even if simple and most effective (in its current version) for a number of parameters below five or six, this method now allows to use SAEM population approach for parameters estimation with PDEs. The range of possible practical applications entering in this category still remains significant.

Perspectives of improvements of the method, to be able to reach higher numbers of parameters to be estimated, can be done in several directions. On the one hand, there are the improvements on the grid, independantly of the SAEM algorithm: all the tools developed in the field of sensitivity analysis may be used to optimise the design of the grid, as well as tools like kriging. On the other hand, the interactive coupling between grid design and SAEM seems to be also an attractive route to be explored.

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 a so called "*délégation*" at INRIA, during this work in 2012-2013. The support of INRIA's Project Lab "MONICA" is also gratefully acknowledged. Numerical simulations were done using computer resources of the *Pôle Scientifique de Modélisation Numérique* (PSMN), Lyon, France.

References

[1] H.T. Banks and K. Kunisch, Estimation techniques for distributed parameter systems, vol. 1. *Systems & Control: Foundations & Appl.* Birkhäuser Boston Inc., Boston, MA (1989).

[2] G.T. Brauns, R.J. Bishop, M.B. Steer, J.J. Paulos and S.H. Ardalan, Table-based modeling of delta-sigma modulators using Zsim. *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.* **9** (1990) 142–150.

[3] B. Delyon, M. Lavielle and E. Moulines, Convergence of a stochastic approximation version of the EM algorithm. *Ann. Statis.* **27** 94–128 (1999).

[4] S. Donnet, J.-L. Foulley and A. Samson, Bayesian Analysis of Growth Curves Using Mixed Models Defined by Stochastic Differential Equations. *Biometrics* **66** (2010) 733–741.

[5] S. Donnet and A. Samson, Parametric inference for mixed models defined by stochastic differential equations. *ESAIM: PS* **12** (2008) 196–218.

[6] R.A. Fisher, The Genetical Theory of Natural Selection. Oxford University Press (1930).

[7] M.B. Giles and N.A. Pierce, An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion* **65** (2000) 393–415.

[8] W.M. Goughran, E. Grosse and D.J. Rose, CAzM: A circuit analyzer with macromodeling. *IEEE Trans. Electron. Devices* **30** (1983) 1207–1213.

[9] V. Isakov, Inverse problems for partial differential equations, vol. 127. *Appl. Math. Sci.*, 2nd edition. Springer, New York (2006).

[10] J. Kaipio and E. Somersalo, Statistical and computational inverse problems, vol. 160. *Appl. Math. Sci.* Springer-Verlag, New York (2005).

[11] A. Kolmogoroff, I. Petrovsky and N. Piscounoff, Étude de l'equation de la diffusion avec croissance de la quantite de matiere et son application a un problème biologique. *Bulletin de l'université d'État à Moscou, Section A* **I** (1937) 1–26.

[12] E. Kuhn and M. Lavielle, Maximum likelihood estimation in nonlinear mixed effects models. *Comput. Statis. Data Anal.* **49** (2005) 1020–1038.

[13] M. Lavielle, Private Communication (2012).

[14] M. Lavielle and K. Bleakley, Population Approach & Mixed Effects Models – Models, Tasks, Tools & Methods. Avalaible at http://popix.lixoft.net/ INRIA (2013).

[15] M. Lavielle and F. Mentré, Estimation of population pharmacokinetic parameters of saquinavir in HIV patients with the monolix software. *J. Pharmacokinetics and Pharmacodynamics* **34** (2007) 229–249.

[16] J.-L. Lions, Optimal control of systems governed by partial differential equations. Translated from the French by S.K. Mitter. Springer-Verlag, New York (1971).

[17] L. Nie, Strong consistency of the maximum likelihood estimator in generalized linear and nonlinear mixed-effects models. *Metrika* **63** (2006) 123–143.

[18] L. Nie and M. Yang, Strong consistency of mle in nonlinear mixed-effects models with large cluster size. *Sankhya: Indian J. Statis.* **67** (2005) 736–763.

[19] E. Snoeck, P. Chanu, M. Lavielle, P. Jacqmin, E.N. Jonsson, K. Jorga, T. Goggin, J. Grippo, N.L. Jumbe and N. Frey, A Comprehensive Hepatitis C Viral Kinetic Model Explaining Cure. *Clinical Pharmacology & Therapeutics* **87** (2010) 706–713.

[20] A. Tarantola, Inverse problem theory and methods for model parameter estimation. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA (2005).

[21] M. Team, The Monolix software, Version 4.1.2. Analysis of mixed effects models. Available at http://www.lixoft.com/ LIXOFT and INRIA (2012).

[22] G. Yu and P. Li, Efficient look-up-table-based modeling for robust design of sigma-delta ADCs. *IEEE Trans. Circuits Syst. – I* **54** (2007) 1513–1528.