A Comparison of a Generic MCMC-Based Algorithm for Bayesian Estimation in C++, R and Julia. Application to Plant Growth Modeling

Gautier Viaud, Yuting Chen, Benoît Bayol, Paul-Henry Cournède

A Comparison of a Generic MCMC-Based Algorithm for Bayesian Estimation in C++, R and Julia. Application to Plant Growth Modeling.

Gautier Viaud[†] Yuting Chen[†] Benoît Bayol[†] Paul-Henry Cournède[†]

[†] CentraleSupélec, Laboratory MAS, 92290 Châtenay–Malabry, France
Corresponding author: gautier.viaud@ecp.fr

January 6, 2016

Abstract

Plant growth is understood through the use of dynamical systems involving many interacting processes and model parameters whose estimation is therefore a crucial issue, all the more so since experimental data obtained from agronomical systems are most of the time characterized by their scarcity and their heterogeneity owing to the complex underlying acquisition. One of the approaches used to solve this kind of problem within a Bayesian paradigm involves Markov Chain Monte Carlo (MCMC) algorithms, one of the drawbacks of the latter being that they lead to some intensive computation because of the statistical framework employed, which is why the efficiency of the implemented computing methods is of particular importance. In this paper, we compare three implementations of a generic MCMC-based algorithm for Bayesian estimation in C++, R and Julia so as to compare the performance and precision of these languages. Here, genericness means that the estimation algorithm can be used for any dynamic model provided that it is implemented in a given modeling template. Such genericness is of crucial importance in a scientific field such as plant growth modeling for which no reference model exists and new models are constantly developed and evaluated. The tests are conducted for the particular cases of Lotka–Volterra model and the Log-Normal Allocation and Senescence model for sugar beet.

Introduction

Plant growth modeling has a wide range of potential applications for decision aid in farm management, the most common one being predicting yields in fields at a global scale. Lots of models, whether they be designed at the individual level, see [13] for a review, or the field level (such as the LNAS [11] or STICS [8] models), have been developed over the past decades. They can make use of a few or several dozens of parameters, and accurate predictions require correct parametrization of the models – which, because of the complexity and the interactions of the underlying biological processes, can be a difficult task.

Two approaches can be chosen for parameter estimation, the frequentist one (with the classical maximum likelihood estimator) and the Bayesian one. In Bayesian estimation, parameters are considered as a random vector whose posterior distribution is deduced from the knowledge of system observations and a prior distribution for the parameters. It offers some interesting perspectives in mechanistic plant growth modeling since some reasonable a priori distributions on the parameters can be deduced from the literature or from the well-known biological processes. The computation of the posterior distribution is generally untractable analytically, and two main families of algorithms can be used for its approximation, Sequential Monte-Carlo (SMC) methods and Markov Chain Monte Carlo (MCMC) methods. In this article, we concentrate on an MCMC-based algorithm for Bayesian estimation and address two main issues often encountered in plant growth models parameter estimation: (i) design both plant growth models and parameter estimation algorithms in a generic form so that each algorithm can be used in concordance with each model and (ii) use the programming language that is the most apropriate for these algorithms that is, ideally, the least time- and memory-consuming and the

easiest for writing code.

So far, parameter estimation algorithms were written for a specific model, whence the proliferation of algorithms identical in nature and the rewriting of code. To avoid this issue, we reformulate the theoretical framework of models so as to be able to apply a generic algorithm for different template models. As far as the second issue is concerned, plant growth models describe complex biological processes, generally on a large number of time steps, and a single simulation of the model is therefore by itself already time-consuming. Consequently, it is not surprising within plant growth modeling communities to have, when resorting to analysis or estimation algorithms that require a large number of simulations, computation times up to several hours, sometimes several tens of hours, and code optimization is thus a critical point. This is why we compared the implementation of a generic MCMC algorithm, of primary importance for parameter estimation in plant growth models, in different programming languages: C++ [1], a common choice for numerical methods when it comes to efficiency, R [3], a long-standing and very used language within the statistics community and with a wide set of libraries, and Julia [2], a rather young language developed only a few years ago but whose community is growing rapidly.

We begin by defining in Section 1 the theoretical framework for the genericness of models that will allow parameter estimation algorithms to be applied to different models easily. In Section 2, we recall the basic principles of MCMC algorithms for parameter estimation within a Bayesian perspective and the slight adaptations necessary to deal with the estimation of both parameters and hidden states. A predator-prey model and a plant growth model that serve as test models for MCMC simulations are then presented in Section 3 and the issues related to the programming side and the specificities of the different languages considered are discussed in Section 4. Finally, in Section 5, we present the results of the efficiency for each language in terms of process time and memory.

1 Generic statistical model

1.1 General state space models

Plant growth models are often formulated in terms of discrete nonlinear state space models [14]. The plant characteristics are updated at each time step $n \in [\![1, N]\!]$ with N the last step of the simulation, where it is made use of a transition function f_n and an observation function g_n as follows:

$$\begin{cases} X_{n+1} &= f_n(X_n, E_n, \Theta, \eta_n), \\ Y_{n+1} &= g_n(X_{n+1}, \Theta, \xi_n), \end{cases}$$

and where at time n, $X_n \in \mathbb{R}^{d_X}$ represents the state variables and $Y_n \in \mathbb{R}^{d_Y}$ the observations that are available. The environment influence is taken into account through the variables $E_n \in \mathbb{R}^{d_E}$, and $\Theta \in \mathbb{R}^{d_\Theta}$ denotes the functional parameters of the model. Two types of noises are considered, the modeling noises $\eta_n \in \mathbb{R}^{d_\eta}$ and the observation noises $\xi_n \in \mathbb{R}^{d_\xi}$. The former are introduced so as to take into consideration possible model imperfections or environmental stochasticity whereas the latter are because experimental data result from field measurements, which are always a source of errors. The experimental data Y_n comprise state variables that are linked to the hidden variables X_n via the observation function g_n but can differ from them. Owing to practical difficulties to obtain experimental data, these may not be available at all times, especially when dealing with biological systems. The set of the O measurement times is denoted by $\mathcal{O} = \{t_i, i \in [\![1, O]\!]\}$ and the following notations are introduced:

$$\begin{cases} X_{1:N} &= \{X_i, i \in [\![1, N]\!]\}, \\ Y_{1\to N} &= \{Y_{t_i}, i \in [\![1, O]\!]\}, \end{cases}$$

where $\forall i \in [\![1, O]\!], 1 \leq t_i \leq t_{i+1} \leq N$. Our aim is to estimate the values of certain functional parameters $\Theta_e \subset \Theta$ jointly to the values of the hidden states $X_{1:N}$. It is also possible to estimate some parameters related to the modeling and observation noises, e.g. the standard deviations of normal distributions although this is not discussed in the current article, for more details, see [10].

1.2 Generic probability distributions

Parameter estimation algorithms should be designed so as to be easily used with different models, which is why some probability density functions (pdf), necessary for this type of algorithms, need to be expressed in a generic form. In

the case of the MCMC algorithm used to run the simulations, these are the pdf of the observations conditional to the parameters and the hidden states $p(Y_{1\to N}|\Theta, X_{1:N})$, the transition pdf $p(X_{n+1}|\Theta, X_n)$ and the observation pdf $p(Y_{n+1}|\Theta, X_{n+1})$. For the n-th step of the simulation, the hidden state variables are written as $X_n = \{X_n^i, i \in [\![1, d_X]\!]\}$, the modeling noises as $\eta_n = \{\eta_n^i, i \in [\![1, d_\eta]\!]\}$ and the observation noises $\xi_n = \{\xi_n^i, i \in [\![1, d_\xi]\!]\}$. Let $m_\eta : [\![1, d_\eta]\!] \to [\![1, d_X]\!]$ and $m_\xi : [\![1, d_\xi]\!] \to [\![1, d_X]\!]$ be applications such that $m_\eta([\![1, d_\eta]\!])$ and $m_\xi([\![1, d_\xi]\!])$ represent the sets of indexes of the state variables on which are set the modeling and observation noises respectively.

$$\begin{cases} X_n^{m_\eta(i)+1} &= f(X_n^{m_\eta(i)}, \eta_n^i), \quad i \in [\![1, d_\eta]\!], \\ Y_n^i &= f(X_n^{m_\xi(i)}, \xi_n^i), \quad i \in [\![1, d_\xi]\!]. \end{cases}$$

In the case of multiplicative noises, this would translate into:

$$\begin{cases} X_n^{m_\eta(i)+1} &= X_n^{m_\eta(i)}(1 + \eta_n^i), \quad i \in [\![1, d_\eta]\!], \\ Y_n^i &= X_n^{m_\xi(i)}(1 + \xi_n^i), \quad i \in [\![1, d_\xi]\!]. \end{cases}$$

It is possible to express the transition pdf by choosing only d_η variables from the state variables whose indices are represented by $q : [\![1, d_\eta]\!] \to [\![1, d_X]\!]$ as:

$$p(X_{n+1}|\Theta, X_n) = \prod_{i=1}^{d_\eta} p(X_{n+1}^{q(i)}|\Theta, X_n^{1:d_X}, X_{n+1}^{1:q(i)-1}) \times \prod_{j \notin q([\![1,d_\eta]\!])} \delta(X^j, m^j(X_n^{1:N}, X_{n+1}^{1:j-1}))$$

where $m^j(X_n^{1:N}, X_{n+1}^{1:j-1})$ is the value computed for the state variable X^j within the model considered. In what follows, since the variables that are deterministic functions of the stochastic variables are computed directly by closure relationships in the simulation program, the Dirac distributions $\delta(\cdot, \cdot)$ take values 1, and hence will be omitted in the following. In particular, we can restrain ourselves to the "noised" variables and rewrite the transition pdf as:

$$p(X_{n+1}|X_n, \Theta) = p(\eta_n|\Theta) = \prod_{i=1}^{d_\eta} p(X_{n+1}^{m_\eta(i)+1}|\Theta, X_{n+1}^{m_\eta(i)}). \tag{1}$$

The observation pdf can be expressed in the same way as:

$$p(Y_{n+1}|\Theta, X_{n+1}) = p(\xi_n|\Theta) = \prod_{i=1}^{d_\xi} p(Y_{n+1}^i|X_{n+1}^{m_\xi(i)}). \tag{2}$$

The generic expression of the pdf of the observations conditional to the parameters and the hidden states $p(Y_{1\to N}|\Theta, X_{1:N})$ naturally follows from that of the observation pdf since:

$$p(Y_{1\to N}|\Theta, X_{1:N}) = \prod_{k=1}^{O} p(Y_{t_k}|\Theta, X_{t_k}) = \prod_{k=1}^{O} \prod_{i=1}^{d_\xi} p(Y_{t_k}^i|\Theta, X_{t_k}^{m_\xi(i)}).$$

Equation (1) makes it possible to compute the transition pdf as long as are specified the nature of the noises (e.g. additive/multiplicative normal, uniform) and lists of labels corresponding to the parameters necessary to compute the values of these pdfs, such as, for the case of multiplicative normal noises, the modeling noise parameters $\{\sigma_{\eta_i}, i \in [\![1, d_\eta]\!]\}$, the deterministic states $\{X_{n+1}^{m_\eta(i)}, i \in [\![1, d_\eta]\!]\}$ and the corresponding stochastic states $\{X_{n+1}^{m_\eta(i)+1}, i \in [\![1, d_\eta]\!]\}$. Likewise, Equation (2) makes it possible to compute the observation probability density where this time lists of labels must include the observation noise parameters $\{\sigma_{\xi_i}, i \in [\![1, d_\xi]\!]\}$, that of the observation states $\{Y_n^{(i)}, i \in [\![1, d_\xi]\!]\}$ and that of the corresponding hidden states $\{X_n^{m_\xi(i)}, i \in [\![1, d_\xi]\!]\}$. In Section 3, these functions are explicitly written for the case of the LNAS model.

2 MCMC algorithms for Bayesian estimation

MCMC algorithms have attracted a lot of attention [16] [5] for the analysis of complex statistical models in a Bayesian perspective since they allow for the evaluation of complex and high-dimensional integrals, whence for the estimation of parameters and hidden states. The general principle of MCMC algorithms is to generate a sequence of samples from a probability distribution from which it is not possible to sample directly. This sequence represents an approximation of the target distribution and can be used for the computation of integrals, for instance to estimate the expectation or the standard deviation of parameters. The procedure goes as follows: at each iteration of the algorithm, a sample from a so-called proposal distribution is generated; this sample is either accepted or discarded based on an acceptance ratio which depends on both the actual candidate and the last accepted one.

In plant growth models, we are not only interested in estimating some of the parameters of the model Θ but also the hidden states $X_{1:N}$. A joint estimation of $(\Theta, X_{1:N})$ is therefore required, according to an update scheme that can be iteratively repeated:

$$\begin{cases} \text{Step 1: Update } \Theta \sim p(\Theta|X_{1:N}, Y_{1\to N}) \\ \text{Step 2: Update } X_{1:N} \sim p(X_{1:N}|\Theta, Y_{1\to N}) \end{cases}$$

since there is a high correlation between the parameters and the hidden states. The motivation for the second step is to improve the estimation of the hidden states once the estimation of the parameters have converged. In what follows, the Adaptive Metropolis-Within-Gibbs (AMWG) algorithm [10] is presented for parameter estimation (Step 1) only.

2.1 Step 1: estimating functional parameters

The target distribution is defined as $p(\Theta, X_{1:N}|Y_{1\to N})$. The principle of AMWG is to generate an ergodic Markov chain $\{(\Theta^{(i)}, X_{1:N}^{(i)}), i \in [\![1, N_p]\!]\}$ whose stationary distribution is precisely the target distribution $p(\Theta, X_{1:N}|Y_{1\to N})$, where N_p is the maximum number of iterations or chain length. At the i-th iteration, a candidate $(\Theta^{(\star)}, X_{1:N}^{(\star)})$ is generated by sampling from a proposal distribution $q(\Theta^{(\star)}, X_{1:N}^{(\star)}|\Theta^{(i)}, X_{1:N}^{(i)})$. As proposed by [15], we decomposed this proposal distribution into two parts,

$$q(\Theta^{(\star)}, X_{1:N}^{(\star)}|\Theta^{(i)}, X_{1:N}^{(i)}) = q(\Theta^{(\star)}|\Theta^{(i)})q(X_{1:N}^{(\star)}|\Theta^{(\star)}).$$

This means that a new candidate for the parameters has first to be sampled, and then the state list $X_{1:N}$ must be generated through the model considered with this set of parameters, i.e. we take

$$q(X_{1:N}^{(\star)}|\Theta^{(\star)}) = p(X_{1:N}^{(\star)}|\Theta^{(\star)})$$

where $p(X_{1:N}^{(\star)}|\Theta^{(\star)})$ is the pdf corresponding to the model simulation function. The candidate $(\Theta^{(\star)}, X_{1:N}^{(\star)})$ is then accepted with probability $\min(1, \alpha)$ where

$$\alpha = \frac{p(\Theta^{(\star)}, X_{1:N}^{(\star)}|Y_{1\to N})}{p(\Theta^{(i)}, X_{1:N}^{(i)}|Y_{1\to N})} \times \frac{q(\Theta^{(i)}, X_{1:N}^{(i)}|\Theta^{(\star)}, X_{1:N}^{(\star)})}{q(\Theta^{(\star)}, X_{1:N}^{(\star)}|\Theta^{(i)}, X_{1:N}^{(i)})}.$$

Accepting the samples with such a ratio ensures that the stationary distribution of the Markov chain will be the target distribution. A common strategy for the sampling of the parameters is random walk: the proposal distribution is therefore taken to be such that $q(\Theta^{\star}|\Theta) = q_{RW}(\Theta^{\star} - \Theta)$ where q_{RW} is symmetric. A classical choice when it comes to multidimensional parameter estimation is to choose a multivariate normal distribution with zero mean and a covariance matrix $\Sigma^{(i)}$ for q_{RW} so that

$$\Theta^{(\star)} \sim \mathcal{N}(\mu^{(i)}, \Sigma^{(i)})$$

where $\mu^{(i)}$ is set to the last accepted candidate $\Theta^{(i)}$. With such a choice and after some rearrangements, the acceptance ratio can therefore be rewritten in a more compact form

$$\alpha = \frac{p(Y_{1\to N}|\Theta^{(\star)}, X_{1:N}^{(\star)})}{p(Y_{1\to N}|\Theta^{(i)}, X_{1:N}^{(i)})} \times \frac{p(\Theta^{(\star)})}{p(\Theta^{(i)})}$$

where $p(\Theta)$ is the prior distribution chosen for the estimated parameters.

The choice of the covariance matrix is of primary importance: if the variance of the proposal distribution is too small, most of the proposed values will be accepted and the difference between two accepted values tend to be rather small, resulting in a slow exploration of the state space. On the other hand, if the variance of the proposal distribution is too large, the candidates will often be rejected and the chain can be stationary for a long time. Adaptive algorithms [17] have been proposed to avoid having to manually change the proposal distribution and employ the following update scheme

$$\begin{cases} \Delta^{(i)} &= \Theta^{(i+1)} - \mu^{(i)} \\ \mu^{(i+1)} &= \mu^{(i)} + \gamma^{(i)}\Delta^{(i)} \\ \Sigma^{(i+1)} &= \Sigma^{(i)} + \gamma^{(i)}[\Delta^{(i)}\Delta^{(i)T} - \Sigma^{(i)}] \end{cases}$$

where $\gamma^{(i)} = 1/i$ is defined so as to make the variations introduced by this adaptive scheme vanish and ensure the ergodicity of the chain. Finally, the covariance matrix is scaled by a factor $\lambda^{(i)}$ that assures a proper exploration of the target distribution even when the covariance matrix is not well initialized

$$\begin{cases} \lambda^{(i+1)} &= \lambda^{(i)}\exp(\gamma^{(i)}[\alpha^{(i)} - \alpha_\star]) \\ \Sigma^{(i+1)} &= \lambda^{(i+1)}\Sigma^{(i+1)} \end{cases}$$

where $\alpha_\star$ is the target acceptance rate. The algorithm is stopped either when the maximum number of iterations is reached or when a specific stopping criterion is verified. Several stopping criteria can be used, they can be based on mean estimates or Monte–Carlo errors, see [10] for more details. Once the algorithm has stopped, the estimated value of the parameters are computed by averaging. However, it is probable that the stationary distribution of the chain is not reached during the first iterations of the algorithm, which is why a burn-in parameter N_{b_p} corresponding to the number of iterations discarded is introduced. Finally, the estimated parameters is computed as

$$\hat{\Theta} = \frac{1}{N_p - N_{b_p}} \sum_{i=N_{b_p}+1}^{N_p} \Theta^{(i)}.$$

All this algorithm needs in order to work in a generic way is the computation of the three pdfs presented in Section 1 and the function that generates the states from time step 1 to time step N for a given model M. The different pdfs can be easily computed by specifying the nature of the modeling and observation noises introduced and the names of the variables affected in model M. It is potentially available for the user to specify its own pdf if the standard ones (normal and uniform as of now) do not suffice. Plant growth models are designed outside the algorithm and any can be called to generate a list of states $X_{1:N}$ with a given set of parameters.

3 On two models

To illustrate the genericness of our approach, we present two examples of the class of models considered in Section 1.

3.1 Model 1: Lotka–Volterra (LV)

The Lotka–Volterra model deals with the evolution of the population of two species, one being a predator and the other a prey. The discrete version we considered reads

$$X_{n+1} = \begin{pmatrix} N_{n+1}^1 \\ N_{n+1}^2 \end{pmatrix} = \begin{pmatrix} (1+a)N_n^1 - bN_n^1N_n^2 \\ (1-c)N_n^2 + dN_n^1N_n^2 \end{pmatrix}$$

where N_n^1 and N_n^2 are the number of preys and the number of predators at time n respectively, and $\Theta = (a, b, c, d) \in \mathbb{R}_+^4$ represents the parameters of the model describing the interaction between the two species. The populations N_n^1 and N_n^2 are assumed to be observed at time n with measurement noises $\epsilon^1 \sim \mathcal{N}(0, (\sigma^1)^2)$ and $\epsilon^2 \sim \mathcal{N}(0, (\sigma^2)^2)$, taken to be additive normal noises,

$$Y_n = \begin{pmatrix} Y_n^1 \\ Y_n^2 \end{pmatrix} = \begin{pmatrix} N_n^1 + \epsilon_n^1 \\ N_n^2 + \epsilon_n^2 \end{pmatrix}$$

Algorithm 1 Adaptive Metropolis-Within-Gibbs

Initialization

Initialize $\Theta^{(0)}$, $X_{1:N}^{(0)}$.

Initialize $\mu^{(0)}$, $\Sigma^{(0)}$, $\lambda^{(0)}$.

Parameters estimation

for $i = 0 : N_p - 1$ **do**

  Sample $\Theta^{(\star)} \sim \mathcal{N}(\Theta^{(i)}, \lambda^{(i)}\Sigma^{(i)})$.

  Generate $X_{1:N}^{(\star)}$ through model M.

  Compute the ratio

$$\alpha = \min\left(1, \frac{p(Y_{1\to N}|\Theta^{(\star)}, X_{1:N}^{(\star)})}{p(Y_{1\to N}|\Theta^{(i)}, X_{1:N}^{(i)})}\frac{p(\Theta^{(\star)})}{p(\Theta^{(i)})}\right).$$

  Sample $u \sim \mathcal{U}(0,1)$.

  if $\alpha < u$ **then**

    Set $(\Theta^{(i+1)}, X_{1:N}^{(i+1)}) = (\Theta^{(\star)}, X_{1:N}^{(\star)})$.

  else

    Set $(\Theta^{(i+1)}, X_{1:N}^{(i+1)}) = (\Theta^{(i)}, X_{1:N}^{(i)})$.

  Update RW mean vector and covariance matrix

$$\mu^{(i+1)} = \mu^{(i)} + \gamma^{(i)}(\Theta^{(i+1)} - \mu^{(i)},$$
$$\Sigma^{(i+1)} = \Sigma^{(i)} + \gamma^{(i)}[(\Theta^{(i+1)} - \mu^{(i)})(\Theta^{(i+1)} - \mu^{(i)})^T - \Sigma^{(i)}].$$

Set $\hat{\Theta} = \dfrac{1}{N_p - N_{b_p}} \displaystyle\sum_{j=N_{b_p}+1}^{N_p} \Theta^{(j)}$.

so that

$$\begin{aligned} p(Y_{n+1}|\Theta, X_{n+1}) &= \frac{1}{\sigma^1\sqrt{2\pi}} e^{\frac{(Y_{n+1}^1 - N_{n+1}^1)^2}{2(\sigma^1)^2}} \\ &\quad \times \frac{1}{\sigma^2\sqrt{2\pi}} e^{\frac{(Y_{n+1}^2 - N_{n+1}^2)^2}{2(\sigma^2)^2}}. \end{aligned}$$

3.2 Model 2: Log-Normal Allocation and Senescence (LNAS)

This model concerns the growth of sugar beet [11]. The latter is considered to be made of two main compartments, the root and the foliage. At day n, the root biomass is denoted Q_n^r and that of the foliage is $Q_n^f = Q_n^g + Q_n^s$, where Q_n^g is the biomass of the green leaves and Q_n^s is the biomass of the senescent leaves. The unit of all biomasses is $g.m^{-2}$. The model is discretized according to a daily time step and the environmental variables of day n such as the temperature T_n ($^\circ C$) and the photosynthetically active radiation PAR_n ($MJ.m^{-2}$) are daily averages. To emphasize the variables on which are set modeling noises, *deterministic* variables are denoted with a superscript *det* whereas their *stochastic* equivalent are denoted with a superscript *sto*. The growth of the plant is led by the thermal time, which represents the accumulated daily temperature above a certain threshold – taken as $T^b = 0\ ^\circ C$ in the case of sugar beet – since germination, corresponding to day $i = 0$, of the plant:

$$\tau_n = \sum_{i=0}^{n} (T_i - T^b).$$

The thermal time must be higher than a certain value τ^{init} for the plant to reach the emergence stage. When the latter is attained, the plant starts to intercept light and as a consequence to produce biomass through photosynthesis. The biomass produced at day n per unit surface area is denoted by Q_n^{det}. It is assumed to follow a Beer–Lambert law:

$$Q_n^{det} = PAR_n \times \mu \times (1 - \exp(-\lambda \times Q_n^g)),$$

where μ ($g.M\mathcal{J}^{-1}$) is an efficiency coefficient, λ ($g^{-1}.m^2$) a coefficient parameter and $(1 - \exp(-\lambda \times Q_n^g)$ represents the fraction of radiation intercepted by the foliage. To account for some inaccuracies of the Beer–Lambert law, the variable production of biomass is rendered stochastic by multiplying its deterministic value by a multiplicative normal noise such that

$$Q_n^{sto} = Q_n^{det} \times (1 + \eta_n^Q),$$

where $\eta_n^Q \sim \mathcal{N}(0, \sigma^Q)$. The biomass produced at day n is distributed between the foliage and root system compartments according to an empirical function γ whose deterministic value is given by

$$\gamma_n^{det} = \gamma^0 + (\gamma^f - \gamma^0) \times G^a(\tau_n),$$

where $\gamma^0 \geq 0$ and $\gamma^f \leq 1$ are respectively the initial value and the limit when the thermal time goes to infinity and G^a is the cumulative distribution of a log-normal law parameterized by its median μ^a and its standard deviation σ^a. As for the production of biomass, a modelling noise for the allocation is introduced since this strategy highly depends on environmental conditions. Once again, a multiplicative normal noise is chosen and

$$\gamma_n^{sto} = \gamma_n^{det} \times (1 + \eta_n^\gamma),$$

with $\eta_n^\gamma \sim \mathcal{N}(0, \sigma^\gamma)$. The biomass of the senescent foliage at day n is denoted by Q_n^s, it is a proportion of the accumulated foliage biomass making use of the cumulative distribution of a log-normal law parametrized by its median μ^s and its standard deviation s^s. This process begins once the thermal time has reached a certain threshold τ^s. The proportion of the foliage at day n becoming senescent is therefore $G^s(\tau_n - \tau^s)Q_n^f$. The biomass of the whole foliage increases every day: it receives the proportion $\gamma \in [0, 1]$ of the biomass produced on day n,

$$Q_n^f = Q_{n-1}^f + \gamma_n^{sto} \times Q_n^{sto},$$

The biomass of senescent leaves is calculated as a proportion ρ^s of the foliage biomass,

$$Q_n^s = \rho_n^s \times Q_n^f,$$

from which the biomass of green leaves can be deduced,

$$Q_n^g = Q_n^f - Q_n^s = (1 - \rho_n^s) \times Q_n^f.$$

Finally, the biomass of the root is increased by what is not allocated to the foliage,

$$Q_n^r = Q_{n-1}^r + (1 - \gamma_n^{sto}) \times Q_n^{sto}.$$

For some days, it is assumed that the green leaves biomass Q^g and the root biomass Q^r are observed with respective measurement noises ϵ^g and ϵ^r that are assumed to be multiplicative normal noises. The experimental data potentially available at day n can therefore be written

$$Y_n = \begin{pmatrix} Y_n^g \\ Y_n^r \end{pmatrix} = \begin{pmatrix} Q_n^g \times (1 + \epsilon_n^g) \\ Q_n^r \times (1 + \epsilon_n^r) \end{pmatrix}.$$

Since

$$\begin{cases} Q_n^{sto} & \sim & \mathcal{N}(Q_n^{det}, (\sigma^Q Q_n^{det})^2) \\ \gamma_n^{sto} & \sim & \mathcal{N}(\gamma_n^{det}, (\sigma^\gamma \gamma_n^{det})^2) \end{cases}$$

the transition pdf can be written as

$$\begin{aligned} p(X_{n+1}|\Theta, X_n) &= \frac{1}{\sigma^Q Q_{n+1}^{det}\sqrt{2\pi}} e^{\frac{(Q_{n+1}^{sto} - Q_{n+1}^{det})^2}{2(\sigma^Q Q_{n+1}^{det})^2}} \\ &\quad \times \frac{1}{\sigma^\gamma \gamma_{n+1}^{det}\sqrt{2\pi}} e^{\frac{(\gamma_{n+1}^{sto} - \gamma_{n+1}^{det})^2}{2(\sigma^\gamma \gamma_{n+1}^{det})^2}}. \end{aligned}$$

In concrete terms, we have to specify into the configuration of the MCMC algorithm before it be run that modeling noises will be multiplicative normal noises as well as the names of the variables affected, i.e. Q^{det} and Q^{sto} with noise standard deviation σ^Q on the one hand and γ^{det} and γ^{sto} with noise standard deviation σ^γ on the other. A generic function then takes care of computing the value of $p(X_{n+1}|\Theta, X_n)$. Since $Y_n^g \sim \mathcal{N}(Q_n^g, \sigma^g Q_n^g)$ and $Y_n^r \sim \mathcal{N}(Q_n^r, \sigma^r Q_n^r)$, the observation pdf can be written as

$$\begin{aligned} p(Y_{n+1}|\Theta, X_{n+1}) &= \frac{1}{\sigma^g Q_{n+1}^g \sqrt{2\pi}} e^{\frac{(Y_{n+1}^g - Q_{n+1}^g)^2}{2(\sigma^g Q_{n+1}^g)^2}} \\ &\quad \times \frac{1}{\sigma^r Q_{n+1}^r \sqrt{2\pi}} e^{\frac{(Y_{n+1}^r - Q_{n+1}^r)^2}{2(\sigma^r Q_{n+1}^r)^2}} \end{aligned}$$

and a similar procedure is used to automatically compute it.

4 Three different languages

C++ is a procedural, object-oriented, generic programming language. It is general purpose and is used intensively in the industrial and academic fields when it comes to numerical programming since it can deliver high performance in terms of speed and memory consumption. However, the language can be quite complex for writing scientific code; in particular, the standard library does not provide any regular function used in modeling and statistics.

R is a dynamic, lazy, functional, object-oriented language. It is probably the most comprehensive domain specific language for statistical analysis, developed by senior statisticians and researchers and used extensively within the statistics community, which provides an undoubted support. It comes with a platform that allows the loading of many packages, more than 6000 as of October 2014, which can be standard statistical tests, models or data sets. It has immense graphical capabilities, which outperform those of most other statistical languages. For all these reasons, it is very convenient for developing ideas and rapid testing. However, the consumption of time and memory cannot compare to that of C++ or other mainstream languages.

Julia is a fairly new language since it appeared in 2012, [7] [6]. R and Julia are both dynamic languages, which are often considered as highly productive but, as is the case of R, not very efficient. Julia was designed with this idea in mind [2] and it was taken advantage of modern techniques such as a rich type information, aggressive code specialization against run-time types and Just-In-Time (JIT) compilation using the LLVM compiler which allows to create functions on the fly. As a result, Julia's performance is claimed to compare to that of statically compiled languages such as C++ while being an interactive dynamic language just like R, providing flexibility and coding productivity.

Possessing this unusual combination of aspects, Julia is a very attractive programming language for plant growth models communities, whence the desire to test its performance compared to those of C++ and R. We implemented the generic MCMC algorithm presented in Section 2 in C++, R and Julia. For researchers unfamiliar with numerical methods, the implementation in R and Julia is much more intuitive than in C++. It must be noted that, since by default R passes all the arguments to functions by copy, we had to embed most of the objects (states, parameters, etc.) inside environment objects to avoid unnecessary copies. On the contrary, variables are always passed by reference in Julia and easily can be in C++, which make them much simpler to design efficient algorithms.

5 Results

We compared the performance of the different languages on two test cases using the LV and LNAS models.

5.1 Test case 1: LV model

We chose a set of parameters

$$(a, b, c, d) = (0.1, 0.02, 0.15, 0.03) \tag{3}$$

and we generated experimental data $Y^1_{1:50}$ and $Y^2_{1:50}$ from these values. We then ran the AMWG algorithm with priors

$$\begin{cases} a & \sim & \mathcal{N}(0.12, 0.012), \\ b & \sim & \mathcal{N}(0.018, 0.0018), \\ c & \sim & \mathcal{N}(0.17, 0.017) \\ d & \sim & \mathcal{N}(0.028, 0.0028). \end{cases}$$

As can be seen from Table 1, the values of the estimated parameters agree with the ones used for simulating the experimental data set.

5.2 Test case 2: LNAS model

The environment variables available were the daily average temperature and the photosynthetically active radiation over 160 days. The experimental data was made of 14 observations at days

$$\mathcal{O} = \{54, \quad 68, \quad 76, \quad 83, \quad 90, \quad 98, \quad 104, \quad 110, \quad 118, 125, \quad 132, \quad 139, \quad 145, \quad 160\}$$

at which were given the values of the biomasses of the green foliage Q^g and of the root Q^r. We estimated three parameters of the model: the radiation use efficiency μ, the median of the allocation log-normal distribution μ^γ and the initial value of the allocation strategy γ^0. All the other parameters were set to a constant value throughout the simulations. We used as prior distributions for the estimated parameters

$$\begin{cases} \mu & \sim & \mathcal{N}(3.6, 0.1), \\ \mu^\gamma & \sim & \mathcal{N}(0.75, 0.08), \\ \gamma^0 & \sim & \mathcal{N}(600.0, 20.0). \end{cases}$$

We chose this particular test case to ensure that the algorithms were correctly written since the results are known for a non-generic MCMC algorithm specifically designed for the LNAS model within the PyGMAlion platform of the Digiplante team.

5.3 General remarks

Because the main concern of these simulations was to compare the efficiency of different programming languages, the same fixed number of iterations was set for each language. We ran $N_p = 350000$ iterations with a burn-in period of $N_{b_p} = 50000$ iterations. The estimated parameters for these test cases are shown in Tables 1 and 2. For each language,

	a	b	c	d
Means	9.99 e-02	1.99 e-02	1.50 e-01	3.00 e-02
Vars	3.41 e-09	2.09 e-09	1.97 e-08	2.54 e-09

Table 1: Final values of the estimated parameters (means and variances) for test case 1.

	μ	μ^γ	γ^0
Means	3.75 e-00	7.52 e-01	5.73 e+02
Vars	3.54 e-03	3.10 e-04	3.08 e+02

Table 2: Final values of the estimated parameters (means and variances) for test case 2.

we measured the time spent with the Unix command `time`, which gives three different values: `real` is the wall clock

elapsed time, `user` is the amount of CPU time spent executing the program, `sys` is the CPU time spent executing operating system services on behalf of the program. The tests were conducted 100 times and the mean total time as well as its standard deviation were computed. It has to be noted that the results of the mean and of the variance indicated for 100 runs in R are based on linear interpolation of the time needed for 1000 iterations, which is still consistent with the time obtained when the full algorithm was run for time issues. We performed these tests both on a personal machine with Intel Core i7-3687U CPU 2.10GHz $\times$ 4, 8 GiB of memory on a 64-bit Linux operating system, denoted Machine 1, and on the computing cluster of CentraleSupélec, with CPU Intel Xeon E5-2695 V2 (2.4Ghz), denoted Machine 2. The results for test case 2 (by far the most time-consuming) are presented in Tables 3 and 4.

	C++	R	Julia
Test case 1			
Machine 1			
`real`	2.87 *e*+00	3.37 *e*+04	5.30 *e*+01
`usr`	2.74 *e*+00	3.76 *e*+04	5.29 *e*+01
`sys`	1.02 *e*-02	1.52 *e*+01	4.36 *e*-01
Machine 2			
`real`	2.38 *e*+00	3.54 *e*+04	5.92 *e*+01
`usr`	2.24 *e*+00	3.95 *e*+04	5.78 *e*+01
`sys`	8.20 *e*-03	2.53 *e*+01	1.50 *e*+00
Test case 2			
Machine 1			
`real`	2.73 *e*+01	1.02 *e*+05	6.48 *e*+01
`usr`	2.73 *e*+01	1.02 *e*+05	6.45 *e*+01
`sys`	1.28 *e*-02	4.62 *e*+01	2.12 *e*-01
Machine 2			
`real`	2.51 *e*+01	1.10 *e*+05	8.34 *e*+01
`usr`	2.51 *e*+01	1.10 *e*+05	8.33 *e*+01
`sys`	1.01 *e*-02	7.05 *e*+01	1.56 *e*+00

Table 3: Times (in seconds) needed for the algorithm in the different languages to end (average based on 100 runs).

	C++	R	Julia
Test case 2			
Machine 2			
`mem`	1.99 *e*+01	5.32 *e*+01	1.68 *e*+02

Table 4: Memory (in MB) used by the algorithms in the different languages (average based on 100 runs).

Obviously, there must still be room for improvement in each language as we designed these implementations as researchers who would have common knowledge of these languages. It was envisioned that R had little chance to compete with the other two languages in terms of time, but not with such high ratios: as far as test case 2 is concerned, on Machine 1, R takes roughly 3,700 more time than C++ whereas Julia takes a bit more than twice, which is a decent achievement for a dynamic language compared to a static one. It has to be noted that C++ is advantaged since the compile time is not taken into account. The results obtained on the computing cluster are somewhat identical, except for Julia's performance. A possible explanation for this is that the binary version of Julia was used here, since it was

not possible to compile it properly. Rather surprisingly, the ratio of times between test case 2 and test case 1 is much higher for Julia than for C++. This probably arises because of some operations that cannot be time-compressed, such as loading packages, including files, JIT compilation of functions, etc. In R's defence, MCMC-based algorithms are not the best-case scenario for this language since no matrix computation is performed. In terms of memory, once again, C++ is the more efficient, R achieves a decent score and Julia seems a bit more greedy. A last point, rather subjective, has to be mentioned: the time required for the development of the algorithms were much less in R and Julia than in C++. All things considered, whereas R is not a conceivable option because of the process time needed, Julia offers a respectable alternative to C++, admittedly a bit more time- and memory-consuming, but with a higher expressiveness.

6 Discussion and perspectives

In this paper, we presented an approach to defining general state space models in a generic form, allowing to design algorithms – such as parameter estimation algorithms – making use of this genericness and which would not require to be written for a specific model.

The Digiplante team built a C++ platform called PyGMAlion based on this principle of genericness [12]. It allows the user to easily define a dynamic model comprising structures such as state variables, environmental variables, parameters and observations. Some statistics tools for parameter estimation, sensitivity analysis, data assimilation and uncertainty analysis are available and the possibility to use generic complex algorithms such as MCMC or Convolution Particle Filters (CPF), see [9], [11], algorithms for different plant growth models is currently under work.

One of the main issues for parameter estimation within plant growth models being numerical computing performance, we compared the memory and time consumptions for different programming languages which highlighted a very promising future for this rather new dynamic language that Julia is: with performances comparable to that of C++, being probably easier for statisticians to code with and having rapidly growing community and package libraries, it may just be a matter of time before it gets more attention from the statistics researchers community. It would be interesting to conduct new tests on other parameter estimation algorithms such as CPF or Particle Markov Chain Monte–Carlo (PMCMC), see [4], all the more so since these can be parallelized, which would be a good opportunity to test the parallel computing capabilities of Julia.

References

[1] C++ website. `http://www.cplusplus.com/`.

[2] Julia website. `http://julialang.org/`.

[3] R website. `http://www.r-project.org/`.

[4] C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society*, 72:269–342, 2010.

[5] Bernd A. Berg and Alain Billoire. Markov chain monte carlo simulations. In *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley and Sons, Inc., 2007.

[6] J. Bezanson, A. Edelman, S. Karpinski, and V.B. Shah. Julia: A fresh approach to numerical computing. arXiv:1411.1607v1. 6 Nov 2014.

[7] J. Bezanson, S. Karpinski, V.B. Shah, and A. Edelman. Julia: A fast dynamic language for technical computing. arXiv:1209.5145v1. 24 Sep 2012.

[8] N. Brisson, M. Launay, B. Mary, and N. Beaudoin. *Conceptual basis, formalisations and parameterization of the STICS crop model*. Quae éditions, 2009.

[9] F. Campillo and V. Rossi. Convolution Particle Filter for Parameter Estimation in General State-Space Models. *IEEE Transactions in Aerospace and Electronics.*, 45(3):1063–1072, 2009.

[10] Y. Chen. *Bayesian Inference in Plant Growth Models for Prediction and Uncertainty Assessment.* PhD thesis, Ecole Centrale Paris, 2014.

[11] Y. Chen and P.-H. Cournède. Data assimilation to reduce uncertainty of crop model prediction with convolution particle filtering. *Ecological Modelling*, 290:165–177, 2014.

[12] P-H Cournede, Yuting Chen, Qiongli Wu, Charlotte Baey, and Benoît Bayol. Development and evaluation of plant growth models: methodology and implementation in the pygmalion platform. *Mathematical Modelling of Natural Phenomena*, 8(04):112–130, 2013.

[13] P. de Reffye, E. Heuvelink, D. Barthélémy, and P.-H. Cournède. Plant growth models. In S.E. Jorgensen and B. Fath, editors, *Ecological Models. Vol. 4 of Encyclopedia of Ecology (5 volumes)*, pages 2824–2837. Elsevier, Oxford, 2008.

[14] A. Doucet, N. De Freitas, and N. Gordon. *Sequential Monte Carlo methods in practice.* Springer, New York, 2001.

[15] P. Fearnhead. MCMC for state-space models. In S. Brooks et al., editors, *Handbook of Markov chain Monte Carlo*, pages 513–529. Chapman and Hall, London, 2011.

[16] W. R. Gilks. Markov chain monte carlo. In *Encyclopedia of Biostatistics*. John Wiley and Sons, Ltd, 2005.

[17] H. Haario and J. Tamminen. An adaptive Metropolis algorithm. *Bernoulli*, 7:223–242, 2001.