Fast simulation of pharmacokinetics

Ylva Wahlquist * **Fredrik Bagge Carlson** ** **Kristian Soltesz** *

* *Lund University, Dept. Automatic Control, Sweden*
(e-mail: {first.last}@control.lth.se)
** *JuliaHub*
(e-mail: fredrik.carlson@juliacomputing.com)

Abstract: Fast simulation of linear time-invariant (LTI) pharmacokinetic (PK) models is crucial to mixed-effect modeling techniques, used extensively in pharmacological research and development. The by far most common LTI PK models are particularly structured compartmental systems with one, two or three compartments. Here we develop and demonstrate very efficient, and down to machine precision exact, simulators for those structures. Our proposed method is benchmarked against state-of-the art software for simulation of linear systems, using a clinically relevant data set.

Keywords: Pharmacokinetics and drug delivery, physiological modeling, biomedical system simulation

1. INTRODUCTION

Simulation of linear pharmacological (PK) models lies at the core of modern pharmacometric methods, widely used in pharmacological research and response prediction. In particular, pharmacometric mixed-effect modeling relies on a large number (thousands or more) of such simulations, to approximate integrals through sampling from patient-individual parameter distributions. These simulations are normally conducted within custom software.

In concurrent work, we are developing neural-network-based symbolic regression for modeling the relation between known patient covariates such as age or gender, and parameters of the PK model, as outlined in Wahlquist et al. (2022). Training these networks requires a several-fold increase in PK model simulations. Even with modern computing power at hand, the large number of simulations constitutes a bottle neck.

The most widely known tool/software that provides simulation of such systems is NONMEM introduced by Sheiner and Beal (1980) in the 1980s. Lately, the software Pumas AI has received attention as a faster alternative to NONMEM, as described in Rackauckas et al. (2020). Much of the speedup is attributed to Pumas AI being implemented in the the Julia language, with native support for differential programming, see Bezanson et al. (2017).

There are also numerous softwares that can simulate linear PK models through numeric integration of ODEs. Such methods are accessible through for example the `lsim` function in Matlab and DifferentialEquations.jl (Rackauckas and Nie (2017)) in Julia. Such numeric ODE integration methods approximate the solution using either fixed or adaptive time step lengths. They do not exploit the fact that the dynamics are LTI, and they do not exploit the fact that the considered input is a sequence of steady infusion levels (steps) and boluses (impulses), administered at known, but typically unevenly spaced, time instances.

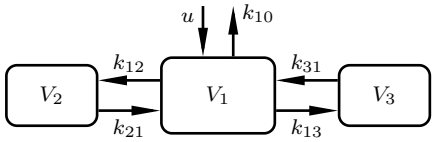


Fig. 1. Mammillary three-compartment PK model. The drug is administered to the central compartment at infusion rate u, from where the drug is also eliminated with elimination rate k_{10}. Two peripheral compartments are connected to the central compartment where transfer rates constants k_{ij} governs the drug transfer from compartment i to j. The compartmental volumes are V_1, V_2 and V_3, respectively.

Since the dynamics are LTI, a faster alternative is to compute the exact solution of the ODE only at the dose change instances, and observation instances (that may differ from them). Doing so naively involved the evaluation of an exponential matrix.

We tailor a method for exactly solving the ODE at instances of interest. By exploiting the structure of one, two, and three compartment LTI PK models, we avoid the need of evaluating exponential matrices, without introducing the need of solving a linear equation system in each time step, otherwise associated with diagonalizing the system matrix.

The efficiency of the proposed method is demonstrated through benchmarking it against commonly used solvers, using a large model set for the anesthetic drug propofol.

⋆ This work was partially funded by the Swedish government through the Swedish Research Council (grant 2017-04989), The Royal Physiographic Society in Lund, the Hans-Gabriel and Alice Trolle-Wachtmeister foundation and by the Swedish foundation for Strategic Research. The authors are members of the Excellence Center at Linköping-Lund in Information Technology (ELLIIT).

2. PHARMACOKINETIC MODELING

Pharmacokinetics describe the absorption and distribution of a drug within the body. To be able to simulate pharmacokinetics and predict physiological parameters, such as the compartment volumes, it is common to use a compartment model. The idea of the compartment model is to collect organs or tissue of similar properties together in compartments and model the drug transfer between them.

Commonly used compartment models are structures of one, two, and three compartments. A one-compartment model approximates the body as a single compartment where the drug concentration is assumed to be uniformly distributed and eliminated through a first-order process. Extending this model results in the two- and three-compartment models, where the different compartments relate to different types of tissue.

The compartment models can have different topologies, due to how the compartments are arranged, and where sources and sinks enters and exits the system. For example, a mammillary model consists of a central compartment with peripheral compartments connected to it, with no interconnections among other compartments. In the context of intravenously administered drugs, the central compartment models the blood plasma.

Figure 1 provides a schematic illustration of the three-compartment mammillary model, used, for example, to model the PK of the anesthetic drug propofol, as explained in Sahinovic et al. (2018). We can also note from the figure that as long as addition and elimination are associated with the central compartment, as is the case for most intravenously administered drugs, the two models of two or one compartment are special cases of the three compartment model.

Diffusion processes transferring drug between communicating compartments are governed by non-negative transfer rate constants k_{ij}, as shown in Figure 1.

A state-space representation of the one-compartment model relating drug infusion rate $u(t)$ to drug concentration $x(t)$ is given by the first-order LTI model

$$\dot{x} = -k_{10}x + \frac{1}{V_1}u, \tag{1}$$

where k_{10} is the elimination-rate constant (1/time), and V_1 is the volume of distribution of the single compartment. We have left out the time dependence on the state variables $x(t)$ and input $u(t)$ for readability.

Similarly, the second-order compartment model is given by

$$\dot{x}_1 = -(k_{10} + k_{12})x_1 + k_{21}x_2 + \frac{1}{V_1}u, \tag{2a}$$

$$\dot{x}_2 = k_{12}x_1 - k_{21}x_2, \tag{2b}$$

where x_1 denotes drug concentration within the central compartment, while x_2 is the peripheral compartment concentration.

The third-order mammillary compartment model, shown in Figure 1, corresponds to

$$\dot{x}_1 = -(k_{10} + k_{12} + k_{13})x_1 + k_{21}x_2 + k_{31}x_3 + \frac{1}{V_1}u, \tag{3a}$$

$$\dot{x}_2 = k_{12}x_1 - k_{21}x_2 \tag{3b}$$

$$\dot{x}_3 = k_{13}x_1 - k_{31}x_3, \tag{3c}$$

where $x_k \, \forall \, k \in \{1, 2, 3\}$, is the drug concentration within compartment k.

3. SIMULATION OF LTI SYSTEMS

For a pharmacological relevant scenario, the input u changes only at discrete time instances during a treatment. At these instances, there is either a step change in the infusion rate, a drug bolus (modeled by an impulse), or both.

The peripheral compartment concentrations are most commonly not directly measurable, as the peripheral compartments are modeling constructs, rather than actual tissues. However, the central compartment concentration can be measured by drawing blood samples at discrete time instances.

Motivated by the circumstances detailed above, we consider methods for simulation of LTI systems when the input is a linear combination of time-shifted impulses and steps. This means that we are dealing with events from the list below, occurring in a sequence of known time instances:

- An impulse of fixed magnitude added to the central compartment;
- Addition of drug mass at a constant rate to the central compartment;
- Observation of the central compartment concentration.

3.1 General solution of LTI systems

Let $\boldsymbol{x}$ be a state vector where x_k represents the drug concentration of compartment $k = 1, ..., n$. The state dynamics of a scalar-input LTI system can be written as

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + Bu(t) \tag{4}$$

where A is an $n \times n$ matrix, B is an $n \times 1$ column vector and u is the input. For the PK models introduced in Section 2, A and B are uniquely determined by transfer-rate constants and the central compartment volume V_1.

The exact solution of (4), derived in for example Åström and Wittenmark (2011), is

$$\boldsymbol{x}(t) = e^{At}\boldsymbol{x}(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau, \tag{5}$$

We assign $t = 0$ to the instance where drug is first administered, corresponding to an initial drug concentration $\boldsymbol{x}(0) = \boldsymbol{0}$. If that is not the case, the zero time can be shifted so that $\boldsymbol{x}(0) = 0$ holds.

The solution (5) can be evaluated through numerical approximation of the integral, and there exist numerous numerical integration methods to this end. Examples include `ode45` and other related methods accessible from the `lsim` wrapper in Matlab or DifferentialEquations.jl in Julia.

If the input changes only at discrete time instances t_k, a more efficient alternative to fixed (or adaptively variable)

step-length numeric integration is to perform a exact zero-order-hold (ZOH) discretization of (5):

$$\boldsymbol{x}(t_{k+1}) = e^{A(t_{k+1}-t_k)}\boldsymbol{x}(t_k) + \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}d\tau B u(t_k), \tag{6}$$

which can be rewritten on the compact form

$$\boldsymbol{x}_{k+1} = \Phi_k \boldsymbol{x}_k + \Gamma_k u_k, \tag{7a}$$

so that

$$\begin{aligned} \Phi_k &= e^{Ah_k}, \\ \Gamma_k &= \int_0^{h_k} e^{A\tau}d\tau B, \end{aligned} \tag{7b}$$

where the sampling period is

$$h_k = t_{k+1} - t_k, \tag{7c}$$

and subscript k denotes time dependence so that for example $\boldsymbol{x}_k = \boldsymbol{x}(t_k)$.

3.2 Simulation with impulses and piece-wise constant inputs

If we consider the input being an impulse (bolus) of magnitude v_k, administered at time t_k, we have

$$u_k = \delta_k v_k, \tag{8}$$

where $\delta_k = \delta(t_k)$ is the Dirac distribution (unit impulse) centered at $t = t_k$. For this case the state update of (7a) becomes

$$\begin{aligned} \boldsymbol{x}_{k+1} &= e^{Ah_k}\boldsymbol{x}_k + \int_0^{h_k} e^{A\tau}B v_k \delta_0(\tau)d\tau \\ &= e^{Ah_k}\boldsymbol{x}_k + e^{Ah_k}B v_k = \Phi_k(\boldsymbol{x}_k + B v_k). \end{aligned} \tag{9}$$

If we instead consider a piece-wise constant input $u_k = w_k$, then (5) takes on the form

$$\begin{aligned} \boldsymbol{x}_{k+1} &= e^{Ah_k}\boldsymbol{x}_k + \int_0^{h_k} e^{A\tau}d\tau B w_k \\ &= \Phi_k \boldsymbol{x}_k + \Gamma_k w_k. \end{aligned} \tag{10}$$

Combining (9) and (10), we get the updated state vector under both an impulse and a piece-wise constant input

$$\boldsymbol{x}_{k+1} = \Phi_k\left(\boldsymbol{x}_k + B v_k\right) + \Gamma_k w_k. \tag{11}$$

As a consequence of the identity

$$\underbrace{\begin{bmatrix} \Phi_k & \Gamma_k \\ 0 & I \end{bmatrix}}_{M} = \exp\left(\underbrace{\begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix}}_{E} h_k\right), \tag{12}$$

which can be easily verified through $dM/dh_k = ME$, it is possible to evaluate Φ_k and Γ_K in (7b) by computing an exponential matrix of dimension $n+1$. This makes the approach less computationally expensive than numeric integration schemes for small tolerances.

4. FAST SIMULATION OF PK MODELS

Computing the matrix exponential needed to evaluate A_d at each dosing instance is the most time-consuming task associated with simulating the system using the exact solution for piece-wise constant inputs. We next show how we can get rid of this step.

Using the Laplace transform, the state dynamics (4) can be written

$$X(s) = (sI - A)^{-1}BU(s). \tag{13}$$

For the considered PK models, this is equivalent to

$$\frac{X(s)}{U(s)} = \frac{\mathrm{adj}(sI-A)B}{\det(sI-A)} = \frac{1}{V_1}\frac{\mathrm{adj}(sI-A)\boldsymbol{e}_1}{\det(sI-A)}, \tag{14}$$

where the last equivalence comes from the fact that $B = (1/V_1)\boldsymbol{e}_1$, where $\boldsymbol{e}_1$ is the first unit vector in Euclidean $\mathbb{R}^n$. The numerator $\mathrm{adj}(sI-A)$ is completely determined by the rate constants. For the i^{th} state in the state vector $X(s)$, with $i = 1, \ldots, n$, we can evaluate (14) so that for each state $X_i(s)$

$$\frac{X_i(s)}{U(s)} = \frac{1}{V_1}\frac{\sum_{j=1}^{n} p_{ji}s^{n-j}}{(s-\lambda_1)...(s-\lambda_n)}, \tag{15}$$

where p_{ij} are the numerator polynomial coefficients and λ_k is the k^{th} eigenvalue of the A-matrix. For the considered models, all eigenvalues are unique, real and strictly negative. Each of these eigenvalues can be explicitly expressed as a function of the rate constants of the system. We used a computer algebra system to solve the eigenvalue problem $A - \lambda I = 0$ for the three compartment case, to obtain the form

$$\lambda = \begin{bmatrix} -c_1 - c_7 \\ -c_9 - c_{10} \\ c_9 - c_{10} \end{bmatrix} \tag{16a}$$

where

$$\begin{aligned} b_1 &= k_{10} + k_{12} + k_{13} + k_{21} + k_{31} \\ b_2 &= k_{21}(k_{10} + k_{13} + k_{31}) + k_{31}(k_{10} + k_{12}) \\ b_3 &= k_{10}k_{21}k_{31} \\ c_1 &= b_1/3 \\ c_2 &= c_1^3 \\ c_3 &= b_2/3 \\ c_4 &= c_3 - c_1^2 \\ c_5 &= (b_1c_3 - b_3)/2 \\ c_6 &= 2\left(c_5 + \sqrt{c_4^3 + (c_2 - c_5)^2} - c_2\right)^{1/3} \\ c_7 &= -\mathrm{Re}(c_6) \\ c_8 &= \mathrm{Im}(c_6) \\ c_9 &= c_8\sqrt{3}/2 \\ c_{10} &= c_1 - c_7/2. \end{aligned} \tag{16b}$$

was obtained by manual substitutions within the corresponding expression tree.

Performing a partial-fraction decomposition of (15), we obtain

$$\frac{X_i(s)}{U(s)} = \frac{1}{V_1}\sum_{j=1}^{n}\frac{r_{ji}}{s-\lambda_j}. \tag{17}$$

In the case of the three-compartment model and for state x_1—corresponding to the measurable central compartment concentration—the partial fraction decomposition is given by

$$\frac{p_{11}s^2 + p_{21}s + p_{31}}{(s-\lambda_1)(s-\lambda_2)(s-\lambda_3)} = \frac{r_{11}}{s-\lambda_1} + \frac{r_{21}}{s-\lambda_2} + \frac{r_{31}}{s-\lambda_3}. \tag{18}$$

By multiplying both sides to get a common denominator, we obtain

$$\begin{aligned} p_{11}s^2 + p_{21}s + p_{31} = r_{31}(s-\lambda_2)(s-\lambda_3) + \\ r_{21}(s-\lambda_1)(s-\lambda_3) + r_{31}(s-\lambda_1)(s-\lambda_2). \end{aligned} \tag{19}$$

Matching powers of s results in the linear system

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 \\ -(\lambda_2+\lambda_3) & -(\lambda_1+\lambda_3) & -(\lambda_1+\lambda_2) \\ \lambda_2\lambda_3 & \lambda_1\lambda_3 & \lambda_1\lambda_2 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} r_{11} \\ r_{21} \\ r_{31} \end{bmatrix}}_{\boldsymbol{r}_1} = \underbrace{\begin{bmatrix} p_{11} \\ p_{21} \\ p_{31} \end{bmatrix}}_{\boldsymbol{p}_1}, \tag{20}$$

where the elements of P are completely determined by the transfer-rate constants.

Performing partial-fraction expansions of (15) also for $k = 1$ and $k = 3$, we have that

$$QR = P, \tag{21a}$$

where

$$P = \begin{bmatrix} \boldsymbol{p}_1 & \boldsymbol{p}_2 & \boldsymbol{p}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ k_{21}+k_{31} & k_{12} & k_{13} \\ k_{21}k_{31} & k_{12}k_{31} & k_{13}k_{21} \end{bmatrix}, \tag{21b}$$

$$R = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \tag{21c}$$

where P is expressed in terms of the rate constants of the mammillary three compartment model (3). (Note that despite the variable naming, Q and R are in general not the QR factors of P.)

Solving (20) for $\boldsymbol{r}_1$, or more generally (21a) for R, can be done by first explicitly computing the inverse of Q, since $R = Q^{-1}P$. For the three-compartment case, the inverse of Q can be expressed in terms of the eigenvalues as

$$Q^{-1} = \begin{bmatrix} \lambda_1^2/d_1 & \lambda_1/d_1 & 1/d_1 \\ \lambda_2^2/d_2 & \lambda_2/d_2 & 1/d_2 \\ \lambda_3^2/d_3 & \lambda_3/d_3 & 1/d_3 \end{bmatrix}, \tag{22a}$$

where

$$\begin{aligned} d_1 &= (\lambda_1-\lambda_3)(\lambda_1-\lambda_2), \\ d_2 &= (\lambda_2-\lambda_3)(\lambda_2-\lambda_1), \\ d_3 &= (\lambda_3-\lambda_2)(\lambda_3-\lambda_1). \end{aligned} \tag{22b}$$

The two-compartment model is a special case, obtained by removing the third row and column of the involved matrices, and equating any entry coefficient with subscript 3 to zero.

For a one-compartment model, there will be no need for a partial-fraction decomposition as the model is already of first order, resulting in $Q = 1$. Therefore, the one-compartment model can be simulated directly as explained below, with $R = 1$.

The same methodology can be used also for models of $n > 3$ compartments. However, for such models, the matrices P and Q do generally not have closed-form entries. However, they can still be numerically pre-computed. Doing so, and performing a LU factorization, enables replacement of the exponential matrix associated with each time step with one forward and one backward substitution needed to obtain R from P and the factors of Q.

Having computed R, we have access to the right-hand side of (17). For each eigenvalue $j = 1, \ldots, n$, and state $i = 1, \ldots, n$, the right-hand side comprises of a parallel interconnection of n first-order systems, as a result of the partial-fraction expansion (17). Each of these are on the form

$$\frac{r_{ij}}{V_1}\frac{1}{s-\lambda_j}. \tag{23}$$

A possible state-space realization of (23) is

$$\begin{aligned} \dot{z}_j &= \lambda_j z_j + u, \\ x_j &= \frac{r_{ij}}{V_1} z_j. \end{aligned} \tag{24}$$

Zero-order-hold sampling of (24) with sampling period h_k results in

$$\begin{aligned} z_{k+1} &= \varphi_j z_k + \gamma_j u_k, \\ x_k &= \frac{r_{ij}}{V_1} z_k, \end{aligned} \tag{25a}$$

where

$$\begin{aligned} \varphi_j &= e^{\lambda_j h_k}, \\ \gamma_j &= \frac{1}{\lambda_j}\left(\varphi_j - 1\right). \end{aligned} \tag{25b}$$

We will now return to study the full system. For all eigenvalues $j = 1, \ldots, n$, we put the corresponding φ_j, γ_j and z_j from (25) into arrays of length j. This allows us to perform parallel computations of each subsystem and therefore speeding up the simulation of the full system. The resulting updating scheme is

$$\boldsymbol{z}_{k+1} = \boldsymbol{\varphi}_k \odot \boldsymbol{z}_k \oplus \boldsymbol{\gamma}_k u_k, \tag{26a}$$

$$\boldsymbol{x}_k = \frac{1}{V_1} R^\top \boldsymbol{z}_k, \tag{26b}$$

where

$$\boldsymbol{z} = \begin{bmatrix} z_1 & \ldots & z_n \end{bmatrix}^\top, \tag{27a}$$

$$\boldsymbol{\varphi} = \begin{bmatrix} \varphi_1 & \ldots & \varphi_n \end{bmatrix}^\top, \tag{27b}$$

$$\boldsymbol{\gamma} = \begin{bmatrix} \gamma_1 & \ldots & \gamma_n \end{bmatrix}^\top, \tag{27c}$$

and $\odot$ denotes the element-wise product (sometimes also called the Hadamard or Schur product) and $\oplus$ denotes the vector addition so that $. \oplus u = . + \mathbf{1}u$.

We have now demonstrated how an n-compartment model can be simulated as a system of n first-order systems, which increases the computational efficiency compared to traditional methods. Specifically, for the case of $n \leq 3$ compartment models with unique eigenvalues, the approximation-free simulation of the state evolution between consecutive (bolus or infusion change) dosing events comes down to computing R of (17). The inverse Q^{-1} is pre-computed using the closed-form expression (22), relying on the also pre-computed eigenvalues of the system matrix A. Due to the structure of the system, the eigenvalues can be evaluated using the algorithm presented in (16), that is computationally much cheaper than solving a general eigenvalue problem of dimension $n = 3$. Finally,

the simulation only results in the computation of three *scalar* exponentials and simple arithmetic operations.

4.1 Simulation algorithm

Instead of simulating the system with $\boldsymbol{x}$ as a state, we can simulate the system in (26) with $\boldsymbol{z}$ as a state and convert back to $\boldsymbol{x}$ once we are interested in the physiological state. The simulation can now be divided into two parts, pre-processing and simulation between events. The pre-processing part includes a computation of R and the n eigenvalues of A. Once this has been done, cheap simulation between events can be performed in parallel.

If we use the state update for impulses and piece-wise constant inputs in Section 3 and (11) together with the state update in (27), we get

$$\boldsymbol{z}_{k+1} = \boldsymbol{\varphi}_k \odot (\boldsymbol{z}_k \oplus v_k) \oplus \boldsymbol{\gamma}_k w_k, \tag{28}$$

where $\boldsymbol{z}_0$ is some initial state.

The simulation between events now only has two vector-vector additions and two element-wise product evaluations, each involving n elements.

At those time instances when we are interested in the physiological state x_i, the output can be computed from (26b), with the state update given in (28).

5. SIMULATION EXAMPLE

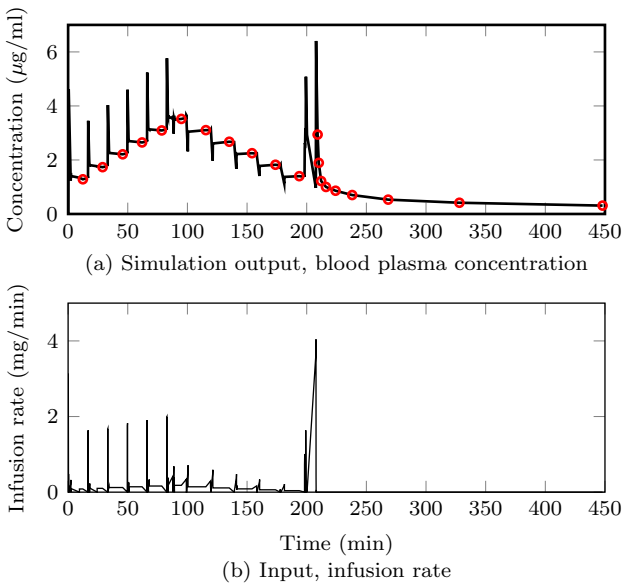
In this paper, we consider simulation of a model set for the anesthetic drug propofol. It has been published in full in in a supplement to Eleveld et al. (2018) and is a data collection from 30 previously published studies. The model set includes identified individual model parameters of the three-compartment model, input infusion data and blood plasma concentrations of 1033 patients as well as observation times.

One simulated example is shown in Figure 2 where the upper figure shows the simulated blood plasma concentration in black, and where the observation instances are marked in red. Note that we are typically only interested in the state at these instances. The lower part of Figure 2 illustrates the infusion rate.

The simulations algorithms compared in this paper are implemented in either Matlab or Julia. In Matlab, `lsim` has been used for simulation. Due to the fact that the time between events is not constant, `lsim` has to be called at each time instance. In Julia, the proposed simulation algorithm in Section 4.1, available through the package FastPKsim.jl Wahlquist (2022b) is compared to direct computation of the exact solution in (11) and simulation using DifferentialEquations.jl with callbacks. For the direct method in (11), computation of the exponential matrix is performed using Padé approximation.

6. RESULTS

Table 1 shows the wall-clock time for simulation of the full model set (1033 patients) published in Eleveld et al. (2018), with the simulations in Matlab and Julia, as explained in Section 5. Even though all simulations in Table 1



(a) Simulation output, blood plasma concentration

(b) Input, infusion rate

Fig. 2. Simulation of one patient in the model set published in Eleveld et al. (2018). The upper figure shows the simulation output, the blood plasma concentration of the central compartment. Red circles shows times for observation. The lower figure shows the infusion rate, administered to the central compartment.

are on the millisecond scale, faster simulations are of outmost importance if thousands (or more) simulations needs to be performed. All simulations were performed single-threaded. For each simulation method, we performed 10 simulations and in the table, we present the median simulation time of these simulations. Simulations in Julia also includes the total memory allocation, which is not available for the Matlab simulation.

The proposed simulator can be found in the Julia package FastPKSim.jl, see Wahlquist (2022b). Code and data used to generate the results in Table 1 are available in a GitHub repo, see Wahlquist (2022a). All computations were performed on a standard Linux PC (Intel i5-8265U, 8-core processor) in Julia 1.8.2 and Matlab 2022b.

Table 1. Wall-clock time for simulation of the model set (1033 patients) published in Eleveld et al. (2018). The simulations has been performed in Matlab and Julia, using the simulation methods described in Section 5. Memory allocation count is not accessible for the Matlab simulation.

Software	Method	Time (ms)	Allocations
Matlab	`lsim`	64585	-
Julia	$x_{k+1} = \Phi_k x_k + \Gamma_k u_k$	548	$2.03 \cdot 10^6$
Julia	DifferentialEquations.jl	230	$2.79 \cdot 10^7$
Julia	FastPKSim.jl	1.58	0

7. DISCUSSION

In this paper, we have demonstrated a fast simulator for LTI PK compartment models of $n = 1$, 2 or 3 compartments. Instead of simulating an n-compartment model, we simulate n first-order models, resulting in a significant speedup compared to traditional methods. By exploiting model structure, we circumvent the need to solve a linear equation system at each time step to obtain a diagonalizing transform needed to achieve this. For models with $n > 3$ compartments (being far less common in pharmacological applications), we would additionally rely on one forward and one backward substitution using precomputed LU factors of Q to solve the equation system associated with diagonalization.

Taking care to avoid dynamic memory allocation further contributes to high efficiency of our implementation in FastPKSim.jl.

In comparison to existing ODE-solvers where the solution is approximate to some degree, the proposed simulator is exact down to the precision of the scalar exponential function evaluation.

When using the simulator as part of an inference engine or to train a symbolic regression network as in Wahlquist et al. (2022), differentiation through the simulator (with respect to model parameters) is necessary. Since Julia has native support for automatic differentiation, this can be performed very cheaply with $\mathcal{O}(1)$ complexity, and with exactness limited only by machine precision. In contrast, ODE solvers called through the `lsim` wrapper in Matlab would need to rely on finite-difference approximations to compute gradients.

In near-future work, we aim to setup FastPKSim.jl to run fully parallelized on a cloud architecture, and use automatic differentiation to train symbolic regression networks, to demonstrate viability of the methodology introduced in Wahlquist et al. (2022) in realistic data-driven modeling scenarios—something that would require far too long computational times using conventional ODE solvers.

REFERENCES

Åström, K. and Wittenmark, B. (2011). *Computer-Controlled Systems: Theory and Design, Third Edition*. Dover Books on Electrical Engineering. Dover Publications.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V.B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1), 65–98. doi:10.1137/141000671.

Eleveld, D.J., Colin, P., Absalom, A.R., and Struys, M.M.R.F. (2018). Pharmacokinetic–pharmacodynamic model for propofol for broad application in anaesthesia and sedation. *British Journal of Anaesthesia*, 120(5), 942–959. doi:10.1016/j.bja.2018.01.018.

Rackauckas, C., Ma, Y., Noack, A., Dixit, V., Mogensen, P.K., Byrne, S., Maddhashiya, S., Santiago Calderón, J.B., Nyberg, J., Gobburu, J.V., et al. (2020). Accelerated predictive healthcare analytics with pumas, a high performance pharmaceutical modeling and simulation platform. *bioRxiv*. doi:10.1101/2020.11.28.402297.

Rackauckas, C. and Nie, Q. (2017). DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software*, 5(1).

Sahinovic, M.M., Struys, M.M.R.F., and Absalom, A.R. (2018). Clinical Pharmacokinetics and Pharmacodynamics of Propofol. *Clinical Pharmacokinetics*, 57(12), 1539–1558. doi:10.1007/s40262-018-0672-3.

Sheiner, L.B. and Beal, S.L. (1980). Evaluation of methods for estimating population pharmacokinetics parameters. I. Michaelis-Menten model: routine clinical pharmacokinetic data. *Journal of Pharmacokinetics and Pharmacodynamics*, 8, 553–571. doi:10.1007/BF01060053.

Wahlquist, Y. (2022a). Fast simulation of pharmacokinetics. URL `https://github.com/wahlquisty/fast-simulation-of-pharmacokinetics`. Commit: 5246112.

Wahlquist, Y. (2022b). FastPKSim.jl. URL `https://github.com/wahlquisty/FastPKSim.jl`. Commit: 55e878d.

Wahlquist, Y., Morin, M., and Soltesz, K. (2022). Pharmacometric covariate modeling using symbolic regression networks.