Numerical solvers for large systems of ordinary differential equations based on the stochastic direct simulation method improved by the Picard and Runge–Kutta principles

Flavius Guiaş

***Abstract*—** We present a numerical scheme for approximating solutions of large systems of ordinary differential equations which at its core employs a stochastic component. The approach used for this level is called *stochastic direct simulation method* and is based on path simulations of suitable Markov jump processes. It is efficient especially for large systems with a sparse incidence matrix, the most typical application being spatially discretized partial differential equations, for example by finite differences. Due to its explicit character, this method is easy to implement and can serve as a predictor for improved approximations. One possibility to reach this is by the Picard principle. Since we have simulated a full path of the corresponding Markov jump process, we can obtain more precise values by using Picard–iterations over small time intervals. This requires the computation of integrals of step–functions, which can be performed explicitly. A further way to increase the precision of the direct simulation method is to use a Runge–Kutta principle. In contrast to the Picard–scheme, here one integrates a polynomial function which interpolates either the values of the original jump process, or the values improved by the Picard iterations, at some equidistant points in the time discretization interval. These integrals can be computed by a proper quadrature formula from the Newton–Cotes family, which is also used in the standard deterministic Runge–Kutta schemes. However, the intermediate values which are plugged into the quadrature formulae are computed in our method by stochastic simulation, possibly followed by a Picard iteration, while in the usual Runge–Kutta methods one uses the well-known Butcher–tableau. We also introduce a time–adaptive version of the stochastic Runge–Kutta scheme. Here we do not take fixed time intervals, but a fixed number of jumps of the underlying process. Depending on the scheme, we may consider intermediate points after the half of this number of jumps. Since in this case the points are not necessarily equidistant in time, we have to compute the corresponding interpolation polynomial and its integral exactly. If high precision is required, this adaptive variant of the stochastic Runge–Kutta method combined with Picard–iterations turns out to be the most effective if compared to the other methods from this family. We illustrate the features of all considered schemes at a standard benchmark problem, a reaction–diffusion equation modeling a combustion process in one space dimension (1D) and two space dimensions (2D).

***Keywords*—**numerical approximation of ordinary differential equations, stochastic direct simulation method, Picard iterations, Runge–Kutta principle.

Flavius Guiaş is with the Dortmund University of Applied Sciences and Arts, Sonnenstr. 96, 44139 Dortmund, Germany; phone: +49-231-9112260; e-mail: flavius.guias@fh-dortmund.de

I. Introduction

Consider n-dimensional autonomous systems of ordinary differential equations (ODEs) $\dot{X} = F(X),\ F = (F_i)_{i=1}^n$ written in an integral form on a small time interval:

$$X(t+h) = X(t) + \int_t^{t+h} F(X(s))\, ds.$$

The value $\bar{X}(t) \approx X(t)$ is assumed to be computed by a certain numerical scheme and the goal is now to determine a value $\bar{X}(t+h)$ which approximates the solution at the next point of the time discretization grid. If we have computed a predictor $\tilde{X}(s)$ on the whole interval $[t, t, +h]$, then one can in principle obtain a better approximation result by performing a Picard–iteration:

$$\bar{X}(t+h) = \bar{X}(t) + \int_t^{t+h} F(\tilde{X}(s))\, ds. \qquad (1)$$

In our approach we compute the predictor $\tilde{X}(s)$ by the *stochastic direct simulation method* (see also [2], [3], [4], [5]) as a path of an appropriate Markov jump process, in which at every jump only one component of the process is changed with a fixed increment ($\pm 1/N$). The steps of this method are the following:

1) Given the state vector $\tilde{X}(s)$ of the Markov process at time s:
2) Choose a component i with probability proportional to $|F_i(\tilde{X})|$.
3) The random waiting time δt is exponentially distributed with parameter $\lambda = N\sum_{j=1}^n |F_j(\tilde{X})|$: $\delta t = -\log U/\lambda$, where U is a uniformly distributed RV on $(0,1)$.
4) Update the value of the time variable: $s = s + \delta t$.
5) Update the value of the sampled component: $\tilde{X}_i \mapsto \tilde{X}_i + \frac{1}{N} sign(F_i(\tilde{X}))$.
6) Update the values $F_j(\tilde{X})$ for all j for which $F_j(\tilde{X})$ depends on the sampled component i.
7) GOTO 1.

We note that the larger N or the larger the absolute values of the r.h.s. of the equations, the smaller the random time step between two jumps. This implies an automatic time–adaption of the scheme which computes the predictor $\tilde{X}(s)$

by direct simulation. The result of this simple algorithm is a vector–valued step–function which, if we let the magnitude of the jumps being sufficiently small, is a decent approximation for the exact solution. Rigorous results concerning the stochastic convergence of Markov jump processes towards solutions of ODEs were proved in [1], and [7], while [4] formulates the convergence result for this particular scheme.

Having computed this jump process, the integral in the Picard–iteration step (1) can be computed exactly (to be precise, it can be updated at every jump of $\tilde{X}(s)$) and so we obtain an improved approximation $\bar{X}(t+h) \approx X(t+h)$. For each component j of the system we define the quantities: $I_j(s) = \int_t^s F_j(\tilde{X}(u))du$ and τ_j as the last time moment when I_j was changed due to a jump of a component which influences the term $F_j(\tilde{X}(s))$. The algorithm based on Markov jump processes using Picard iterations can be then described as follows:

1) Given the state vector $\tilde{X}(s)$ of the Markov process at time s:
2) Choose a component i with probability proportional to $|F_i(\tilde{X})|$.
3) The random waiting time δt is exponentially distributed with parameter $\lambda = N\sum_{j=1}^{n}|F_j(\tilde{X})|$: $\delta t = -\log U/\lambda$, where U is a uniformly distributed RV on $(0,1)$.
4) Update the value of the time variable: $s = s + \delta t$.
5) Update the Picard-integrals: $I_j \mapsto I_j + (s-\tau_j)F_j(\tilde{X})$ and set $\tau_j = s$, for all j for which $F_j(\tilde{X})$ depends on the sampled component i.
6) Update the value of the sampled component: $\tilde{X}_i \mapsto \tilde{X}_i + \frac{1}{N}sign(F_i(\tilde{X}))$.
7) Update the values of $F_j(\tilde{X})$ for all j for which $F_j(\tilde{X})$ depends on the sampled component i.
8) Counter=Counter+1;
9) if (Counter=M) Picard-Iteration();
10) GOTO 1.

The above loop changes only the values of the process $\tilde{X}(s)$. In the procedure Picard-Iteration() we perform the following steps by which we obtain the improved approximation $\bar{X}(t+h)$ given the known value $\bar{X}(t)$:

1) $I_j := I_j + (s-\tau_j)F_j(\tilde{X})$ for all j.
2) Update all components: $\bar{X}_j \mapsto \bar{X}_j + I_j$ for all j.
3) Update all sampling probabilities, setting them as $F_j(\bar{X})$ for all j.
4) Reset: $\tilde{X}_j = \bar{X}_j$, Counter=0, $I_j = 0, \tau_j = s$, for all j.

A more detailed discussion of this issue can be found in [5], where also the sampling and implementation methods are explained.

Based on the predictor $\tilde{X}(s)$, by performing a Picard–iteration we thus obtain improved approximations $\bar{X}(t)$ and $\bar{X}(t+h)$ (possibly also at several intermediate time steps $\bar{X}(t+hc_i)$, $0 \le c_i \le 1$). Having done this, we discuss next the possibility of further improving our approximation of the exact solution, by computing a new quantity following the integral scheme:

$$X^*(t+h) = X^*(t) + \int_t^{t+h} Q(s)\,ds. \tag{2}$$

As integrand $Q(s)$ we take the polynomial which interpolates some intermediate values of $F(\tilde{X}(\cdot))$ or $F(\bar{X}(\cdot))$. The directly simulated process $\tilde{X}$ or the Picard–approximations $\bar{X}$ are evaluated here at some few equidistant points between t and $t+h$. By using an exact quadrature formula in order to compute the integral in (2), we can employ the principle of the Runge–Kutta method in order to further improve our approximation.

The purpose of this paper is to present a general framework for this family of schemes and to illustrate this principle with several concrete examples. We mention that one scheme belonging to this family was previously introduced in our paper [5].

II. The framework of the schemes based on the Runge–Kutta principle

The family of Runge–Kutta (RK) methods is based on the scheme

$$X^*(t+h) = X^*(t) + h\sum_{i=1}^{m} b_i k_i, \tag{3}$$

where k_i are suitable approximations for $F(X(t+hc_i)), 0 \le c_i \le 1$ (we consider here only the case of autonomous systems, to which the stochastic simulation method can be efficiently applied).

For different values of m, the sum in the r.h.s. of (3) is taken as an exact quadrature formula for the polynomial $Q(s)$ which interpolates the nodes $(t+hc_i, k_i)$, $i = 1 \dots m$, on the interval $[t, t+h]$.

For $m = 2$ and $c_1 = 0, c_2 = 1$ we obtain the trapezoidal rule:

$$h\left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right) \tag{4}$$

which integrates exactly the linear interpolant for the two nodes.

For $m = 3$ and $c_1 = 0, c_2 = 1/2, c_3 = 1$ we obtain Simpson's 1/3 (or Kepler's) rule:

$$h\left(\frac{1}{6}k_1 + \frac{4}{6}k_2 + \frac{1}{6}k_3\right) \tag{5}$$

which integrates exactly the quadratic interpolation polynomial for the three nodes, but in this particular case all cubic polynomials are also integrated exactly.

For $m = 4$ and $c_1 = 0, c_2 = 1/3, c_3 = 2/3, c_4 = 1$ we obtain Simpson's 3/8 rule:

$$h\left(\frac{1}{8}k_1 + \frac{3}{8}k_2 + \frac{3}{8}k_3 + \frac{1}{8}k_4\right) \tag{6}$$

which integrates exactly the cubic interpolation polynomial for the four nodes.

By considering for the k_i the values of $F(\tilde{X}(t+hc_i))$ or $F(\bar{X}(t+hc_i))$ based on the simulated Markov process or on

its Picard iterate, we can obtain several stochastic Runge–Kutta or Picard–Runge–Kutta schemes. The scheme (5) was already introduced in [5] (in a slightly modified version), while the other ones are new.

To make a comparison: for the well-known classical Runge–Kutta methods the k's can be computed recursively by

$$k_i = F(X^*(t) + h\sum_{j=1}^{i-1} a_{ij}k_j).$$

The values of b_i, c_i and a_{ij} have to be properly chosen, in order to obtain consistency and convergence. The most typical examples are enumerated in the following.

For $m = 3$ the scheme corresponding to (5) leads to the RK method of order 3:

$$k_1 = F(X^*(t)), k_2 = F(X^*(t) + hk_1/2),$$

$$k_3 = F(X^*(t) - hk_1 + 2hk_2).$$

But, since the used quadrature formula integrates exactly also cubic polynomials, in the deterministic setting one can take $c_1 = 0, c_2 = c_3 = 1/2, c_4 = 1$ in order to obtain the standard RK scheme of order 4:

$$k_1 = F(X^*(t)), k_2 = F(X^*(t) + hk_1/2),$$

$$k_3 = F(X^*(t) + hk_2/2), k_4 = F(X^*(t) + hk_3)$$

based on the integration formula

$$h\left(\frac{1}{6}k_1 + \frac{2}{6}k_2 + \frac{2}{6}k_3 + \frac{1}{6}k_4\right) \tag{7}$$

which is fact derived from (5) but with two approximating values k_2 and k_3 at the middle of the interval.

For $m = 4$, the scheme corresponding to (6) leads to the 3/8 RK method of order 4:

$$k_1 = F(X^*(t)), k_2 = F(X^*(t) + hk_1/3),$$

$$k_3 = F(X^*(t) - hk_1/3 + hk_2),$$

$$k_4 = F(X^*(t) + hk_1 - hk_2 + hk_3).$$

We note that the classical scheme (7) has no natural stochastic counterpart, since in the middle of the interval we would need two approximating values k_2 and k_3. Nevertheless, the stochastic scheme based on (6) is formally similar to the 3/8 RK method of order 4 presented above. As we will see in the numerical examples, although the intermediate values k_i computed by the stochastic method demand more computational effort, they are more precise than those used by the classical deterministic RK schemes and the overall efficiency of the new solver can be superior, especially in the case of equations which are difficult to integrate.

III. Stochastic Runge–Kutta schemes

In this paper we consider three versions of stochastic Runge–Kutta schemes, based on the quadrature formulae (4), (5), (6). We will denote them by RK2, RK3, RK4 respectively, the number suggesting the convergence order of the corresponding deterministic method. To be precise, we consider either $k_i = F(\tilde{X}(t + hc_i))$, where $\tilde{X}(\cdot)$ is the path simulated by (dODE), or $k_i = F(\bar{X}(t + hc_i))$, where $\bar{X}(t + hc_i)$ is the result of the Picard iteration (1) at the corresponding time moment, taken on the interval elapsed since the previous iteration.

We will illustrate the features of the method on a very simple example. Consider the ODE $x' = x,\ x(0) = 1$ with exact solution $x(t) = e^t$. We chose for example $N = 50$ and $t_{max} = 0.3$. We simulate the stochastic process $\tilde{X}(\cdot)$ by the algorithm: $\tilde{X}(0) = 1$, $\tilde{X} \to \tilde{X} + 1/N$ after a random waiting time $\tau = -log(U)/(N \cdot \tilde{X})$, where U is a uniformly distributed RV on $(0, 1)$. This means that the waiting time τ is exponentially distributed with parameter $\lambda = N \cdot \tilde{X}$. Moreover, the time steps τ are automatically adapted, since this approach imposes that the current value of $\tilde{X}$ changes always by the fixed quantity $1/N$. We note that the size of the time steps is inverse proportional to the value of $\tilde{X}$ and also to the value of N, which can be increased if we want to improve the time resolution and by this the overall precision of our approximation.

We compute successive jumps, until the simulation time t_* of the process exceeds t_{max}, so $\tilde{X}(t_*)$ will be the first approximation for e^{t_*} (predictor). Since we have computed a full path, we gained information not only by the final value of the simulation, which may be biased by random fluctuations, but also by the whole behavior of the process $\tilde{X}$ on the current time interval. We can therefore exploit this global information by performing a Picard iteration at $t = t_*$ in order to improve the approximation: $\bar{X} = x(0) + \int_0^{t_*} \tilde{X}(s)\, ds$. This is the integral of a step function which can be computed explicitly as $\sum \tilde{X}(t_i)(t_{i+1} - t_i)$, where t_i are the jump times of the step function $\tilde{X}(\cdot)$ (more precisely, the value of the Picard integral cand be updated after every jump). We can also compute an improved approximation using the RK2-step (4), where $k_1 = x(0) = 1$ and either $k_2 = \tilde{X}(t_*)$ (RK2-method) or $k_2 = \bar{X}$ (Picard-RK2-method). The results are plotted in Fig. 1. We illustrate here a situation where the error of the computed path is very large compared to the exact value e^{t_*} and show how the Picard iteration, the RK2-step or the combination of both improves the approximation. In the same figure is plotted the error curve from 48 simulations (magnified by the factor 5) of the four stochastic approximations of the exact value of the exponential function (absolute value of the differences at t_*). For this error curve the tics on the x-axis have no relevance. We note that the fluctuations of (dODE) are very large, while Picard or RK2 have the effect to significantly

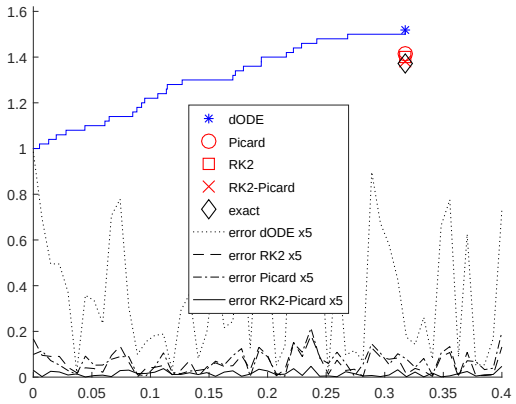

Fig. 1. A sample path of $\tilde{X}(\cdot)$ and the improved approximations at $t = t_*$ with error curves from 48 simulations

reduce the error. Moreover, the combination of both is even more precise.

Turning back to the general problem, we note that since we consider stochastic processes with small random jump times, we cannot hit exactly the prescribed time steps required by the Runge–Kutta scheme. Instead, we take the first time when the prescribed time step is exceeded during the simulation. If the jump time intervals τ of the stochastic process are small compared to the time step h considered in the Runge–Kutta scheme (a typical example: $h = 10^{-7}, \tau = O(10^{-11})$), the error is expected also to be sufficiently small. Alternatively, one can work with the exact times given by the stochastic process, but then the simple formulae (4)-(6) have to be replaced with the exact computation of the linear, quadratic or cubic interpolation polynomial and of its integral. This eliminates the error induced by (5)-(6) when using slightly different time moments, but at a higher computational cost. This version was used in [5] in order to correct the formula (5).

For this paper we performed several experiments, which showed the following facts. First, if the time steps h are not extremely small, the overall efficiency does not change, regardless which alternative we will choose: the cheap approximate computation of the integral with inexact time moments, or the more expensive interpolation-based exact computation. Moreover, the stochastic RK2-method for fixed time steps h is too imprecise in problems which require a high time resolution as in our test case, where the other two are preferable. Nevertheless, we will see later that it is a good choice if we consider a stochastic Runge–Kutta scheme with adapted time steps h. Second, if the time steps h are taken very small, then the computation of the interpolation polynomials (either by the Lagrange or by the Newton formula) will involve difference quotients with very small terms and the output given by the computer is biased by rounding errors. This phenomenon can be observed in a sudden lost of precision if we use quadratic polynomials in the RK3-method or even in chaotic results if we use cubic interpolation polynomials in the RK4-method.

IV. Adaptive stochastic Runge–Kutta schemes

Another possibility is to use a stochastic RK-scheme with *adapted time steps*. That is, we do not prescribe the value h of the time step, but we fix a number M of jumps of the process (typical example: $M = n$ in 1D and $M = 0.3n$ in 2D, where n is the number of equations) and the time step h is determined automatically as the sum of all lengths of jump intervals since the previous step (which are in turn automatically adapted, by the nature of the jump process). As in the situation with fixed time intervals, we can either use as predictor only the simulated Markov jump process $\tilde{X}(\cdot)$, or its Picard iterates $\bar{X}(\cdot)$. For the RK2-method we thus consider simply the process or its Picard iterate at the time moment when M jumps were performed. The formula (4) yields the exact integral of the corresponding linear interpolant. For the RK3 and RK4-methods we have to compute the intermediate k_i's, now related to the number M of jumps: after $M/2$ jumps in case of RK3 and after $M/3$ and $2M/3$ jumps in case of RK4. But now, since the time moments are not necessarily equidistant (even not approximately so), we are obliged to use an exact computation of the interpolation polynomial and of its integral. The drawbacks of this approach were presented above and the experience showed that the adapted RK4-method, although theoretically it should be the most precise, is impracticable due to the mentioned rounding errors. Nevertheless, the adapted RK3-method functions well and is in principle slightly better than the adapted RK2-method, except in the range where the time intervals h after M jumps become too small (due to large values of N considered in the stochastic scheme). With increasing values of N, the convergence of the RK2-method is improving, while the RK3-method, for N larger than a threshold value, remains at the same level of the error due to the rounding effects. Another aspect of the adapted RK3-method is the handling of the results when reaching the maximal computation time t_{max}. If this occurs after computing k_2 after $M/2$ jumps, then we simply compute k_3 at the stopping time of the process and take the usual approach. However, if we reach t_{max} with less than $M/2$ jumps, then we simply perform a RK2-step for this last time interval.

V. Numerical examples

We illustrate the application of several stochastic, but also deterministic methods at the test equation

$$\frac{\partial u}{\partial t} = \Delta u + \frac{5e^{\delta}}{\delta}(2-u)\exp\left(-\frac{\delta}{u}\right) \tag{8}$$

with initial condition $u_0 \equiv 1$, either on $(0,1)$ with boundary conditions $\partial_\nu u(0) = 0$ and $u(1) = 1$, or on the square

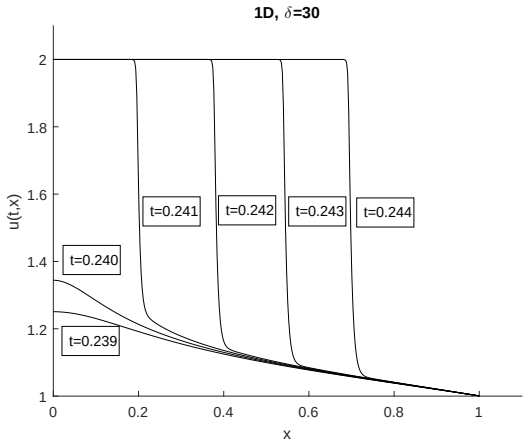

Fig. 2. The solution of (8) in one space dimension for $\delta = 30$.

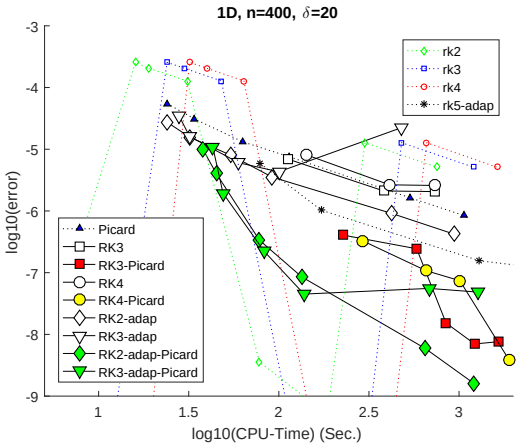

Fig. 3. Efficiency comparison in 1D $\delta = 20$, $n = 400$

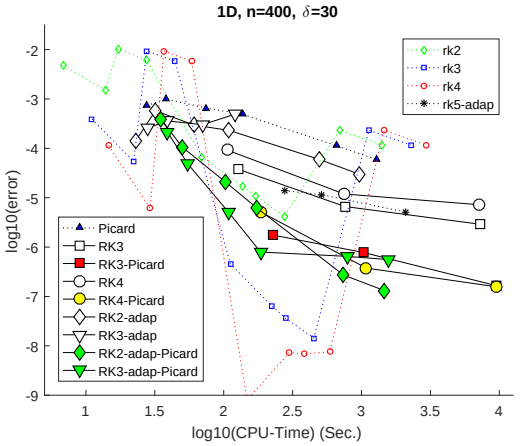

Fig. 4. Efficiency comparison in 1D for $\delta = 30$, $n = 400$

$(0,1)^2$, with boundary conditions $\partial_\nu u = 0$ if $x = 0$ or $y = 0$ and $u = 1$ if $x = 1$ or $y = 1$.

This is a standard benchmark problem which requires a high time resolution [6]. Its solution in 1D is plotted in Fig. 2. The variable u denotes a temperature which increases gradually up to a critical value when ignition occurs (in this example at time $t = 0.240$) resulting in a fast propagation of a reaction front towards the right end of the interval. Due to the very high speed and the steepness of the front, a very precise time resolution is crucial in any numerical approximation of this problem.

By a finite–difference discretization with spatial mesh size of $1/400$ we obtain a system of ODEs, for which we compute the solution up to $t = 0.244$ in the case $\delta = 30$ and up to $t = 0.27$ in the case $\delta = 20$. Note that the larger the value of δ, the larger the stiffness of the problem. As error measure we consider the supremum norm of the difference to a reference solution vector u_* at the end of the computational time interval. The reference solution is computed in MATLAB with the highest possible degree of precision by the standard solver *ode45*. As alternative we test also the stiff solver *ode15s*. For $\delta = 20$ the CPU-times were 18 seconds for *ode45* and less than 2 seconds for *ode15s*, the maximal difference between the two solutions being of about $1.4 \cdot 10^{-9}$. For $\delta = 30$ the CPU-times were 36 seconds and 14 seconds for *ode45* and *ode15s* respectively, the difference between the two solution being slightly less than 10^{-7}. Since the solver *ode15s* has a lower convergence order, we consider as reference solution the results of *ode45*, which are much closer ($O(10^{-13})$ for $\delta = 20$ and $O(10^{-10})$ for $\delta = 30$) to the results produced by the solver *ode113*. The high performance of all these solvers relies on the powerful facilities of MATLAB, which involve computations performed in a vectorial manner. The codes have only a few program lines, with few function calls and the pattern of the corresponding Jacobian is computed apriori, within a separate function.

Our algorithms were however programmed in an object-oriented manner in C++ and the computations involving vectors were performed sequentially, within usual loops. For this reason, in order to get a better comparison, we use also a self-programmed version of *ode45*, which is nothing but a time-adaptive Runge–Kutta method of order 5, called the *Dormand–Prince* method. In contrast to MATLAB, here the updates of the involved vectorial quantities occur also within normal loops. In the legend this method is referred to as 'rk5-adap'. At the same time we use also the results produced by three standard Runge–Kutta solvers with fixed time steps, called 'rk2', 'rk3' and 'rk4', which correspond to the schemes (4), (5) and (7).

Fig. 3 shows the results of the different methods for the case $\delta = 20$, which is the easier case, with a less steep front. We note that the standard solvers 'rk2', 'rk3' and 'rk4' exhibit an unstable behavior. For a certain range of the time steps the error is extremely small, but if we take larger or smaller values, the error will get larger. However, the time adapted solver 'rk5-adap' shows stability and a good

precision.

We discuss now the performance of the stochastic solvers. Due to the poor performance in problems like this, where an extremely precise time resolution is needed, we omit plotting the results of (dODE) (i.e. the direct simulation of the Markov jump process, which is at the core of all the other schemes) and of RK2. Nevertheless, the solver (dODE) combined with Picard iterations shows a similar performance to RK3 and RK4. The method 'RK2-adap' behaves slightly better, while 'RK3-adap', after showing for a while a similar efficiency to the latter solver, exhibits afterward the previously described phenomenon of rounding errors, which becomes visible by the loss of precision if we increase N, decreasing thus the length of the time intervals. The score obtained by the deterministic solver 'rk5-adap' is slightly better than that of the previous group of stochastic methods.

As next group of stochastic solvers we consider 'RK3-Picard' and 'RK4-Picard'. That is, at the time moments prescribed by the Runge–Kutta methods we do not use the values produced by (dODE), but we improve them by a Picard iteration. In this case, the precision is increased by taking larger N and smaller time steps h. Note that h cannot be arbitrarily increased (for deterministic methods this would yield instability, here larger errors), while for small h, we have to take a sufficiently large N in order to obtain good results. This implies smaller jump times and therefore longer computation times. Nevertheless, if N is too large, for fixed h we do not observe an increased precision. There is therefore an optimal dependency between h and N, which can be determined experimentally in order to obtain the best efficiency. We note that the RK-Picard solvers are not cheap ones, their employment makes therefore sense only if we seek a high degree of precision, even at a higher computational cost. Anyway, at the same (high) computational cost, they are definitely more precise than the previous groups of solvers.

The last group of solvers consists of the adaptive Runge–Kutta-Picard solvers 'RK2-adap-Picard' and 'RK3-adap-Picard' in its first variant (where we stop after $M/2$ and M iterations). In a certain range they seem to exhibit a similar performance, clearly better than all the other solvers, while for large values of N the already discussed imprecision of 'RK3-adap-Picard' shows up again. Nevertheless, since in the solver 'RK2-adap-Picard' we don't use interpolation polynomials of high degree, this type of errors does not appear and overall this solver may be considered as the 'test-winner'.

Fig. 4 shows the results for $\delta = 30$, which is a much more demanding problem than the previous one. We will mention only some differences compared to the former situation. The solvers 'RK3' and 'RK4' perform now similarly to 'rk5-adap' and better than the (dODE)-Picard solver or 'RK2-adap' or 'RK3-adap'. Within a certain range, the best efficiency is provided this time by 'RK3-adap', and 'RK2-adap' surpasses it only for larger values of N. Within this range, even if 'RK2-adap' is again the best one, its efficiency is comparable to that of 'RK3-adap', 'RK3-Picard' and 'RK4-Picard'.

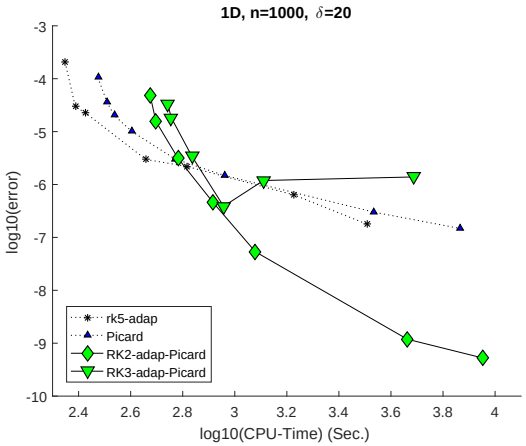

Fig. 5. 1D for $\delta = 20$, $n = 1000$

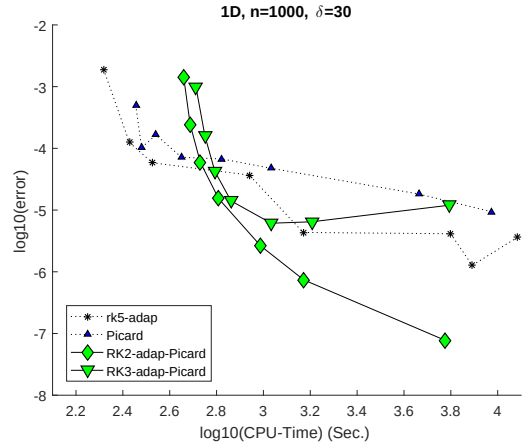

Fig. 6. 1D for $\delta = 30$, $n = 1000$

We perform next computations for the smaller spatial discretization step of 1/1000. In order to obtain a good reference solution we use the solver 'RK2-adap-Picard' with a large value of N. The reason for this choice is that in this case the MATLAB-solvers reach their limits. The solver *ode15s* with the minimal value of the parameter 'RelTol'=$2.2 \cdot 10^{-14}$ is still fast but, by its construction, it is not a high precision solver. For $\delta = 20$ we can use the solver *ode113* with the smallest possible value 'RelTol'=10^{-10} (due to memory reasons), the difference between the two computed solutions being of 10^{-8}. So, this is virtually the best precision which can be reached by the MATLAB-solvers. For $\delta = 30$ we can use *ode113* only with 'RelTol' larger than 10^{-5}, the difference to the solution delivered by *ode15s* being of 10^{-4}. The results of the numerical experiments are plotted in Fig. 5 and 6. We compare here

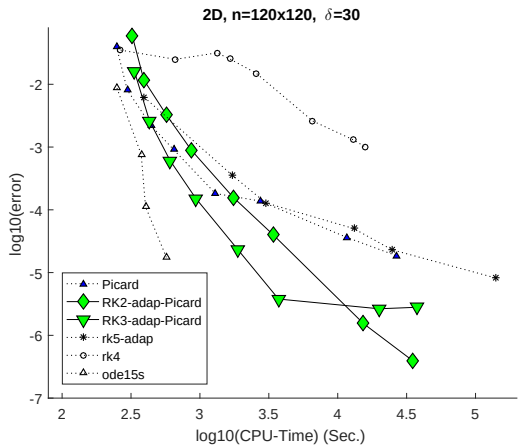

Fig. 7. Efficiency comparison in 2D for $\delta = 30$, $n = 120 \times 120$

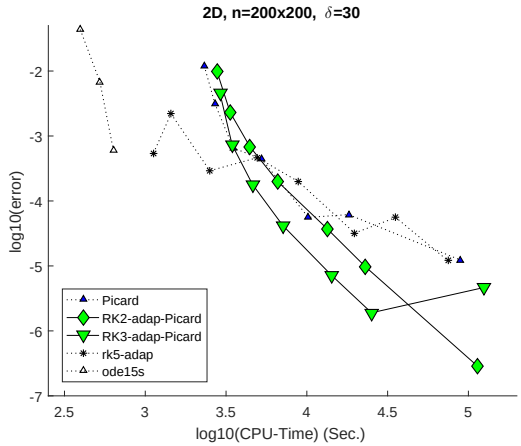

Fig. 8. Efficiency comparison in 2D for $\delta = 30$, $n = 200 \times 200$

only stochastic solvers, all based on Picard-iterations in different variants: 'Picard' and the adaptive schemes 'RK2-adap-Picard' and 'RK3-adap-Picard'. As deterministic solver in this test we use the standard one, 'rk5-adap'. We note that this scheme performs similarly to the stochastic Picard–solver and in the precision range up to $10^{-4} - 10^{-5}$ both of them are faster than the adaptive stochastic RK-solvers. Nevertheless, if higher precision is required (for example to compute a good reference solution), the latter solvers are more performant. 'RK2-adap-Picard' turns to be again a robust solver with a good precision. We observe the known problems caused by rounding errors in the case of the solver 'RK3-adap-Picard'. Altough in a certain precision range this solver performs slightly better than the other stochastic Runge–Kutta solver, by increasing the precision parameters above a certain level, the precision of the computed solutions does not improve. However, this is not the case for the solver 'RK2-adap-Picard'. We note also that by using this scheme, we can obtain a higher precision than using the solvers available in MATLAB.

The next figures depict the results in case of the 2D-problem for $\delta = 30$, computed up to $t_{max} = 0.268$. Fig. 7 corresponds to the spatial discretization step of 1/120 (reference solution computed by *ode113*), while Fig. 8 to a step of 1/200 (reference solution computed by 'RK2-adap-Picard'). We compare the same solvers as in the previous example, together with the MATLAB-solver *ode15s* and the standard Runge–Kutta method 'rk4' (only for the spatial discretization step of 1/120). We note that in the case of this difficult problem, with a large number of equations, the performance of this solver is poor compared to the others. Concerning the stochastic solvers, the picture is almost the same as before. The 'Picard' solver performs similarly to the deterministic 'rk5'-method. If one needs a higher precision, the stochastic solvers 'RK2-adap-Picard' and 'RK3-adap-Picard' are more performant. In both cases we considered also the results of the MATLAB-solver *ode15s*, set up to the maximal possible precision. Due to the fast MATLAB-codes, based internally on strong parallelization, the ODE-solvers available in this software are generally very fast. Nevertheless, for difficult problems (as in the 1D case with a very small discretization step of 1/1000 or in the 2D case for steps 1/200 or smaller), the stochastic solvers presented in this paper perform better in the range of high precision.

VI. Conclusions

The conclusion of these experiments is that by using the Picard and/or Runge–Kutta principle, the performance of the stochastic direct simulation method can be enhanced considerably. The stochastic solvers presented here are comparable to deterministic ones, at least in a precision range relevant for PDEs. Within the framework of all Runge-Kutta-solvers of stochastic type, we distinguish the time adaptive solver 'RK2-adap-Picard', which shows the best allround performance, either if we need a good precision at a mostly cheap computational cost, or if we need a very high precision, even at the price of an expensive computational cost.

References

[1] S. Ethier, T.G. Kurtz, Markov Processes: Characterization and Convergence, Wiley, 1986

[2] D. Gillespie, *General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions*, J. Comp. Phys. 22 (1976) 403–434

[3] D. Gillespie, *Stochastic Simulation of Chemical Kinetics*, Annu. Rev. Phys. Chem. 58 (2007) 35–55

[4] F. Guiaş, *Direct simulation of the infinitesimal dynamics of semi-discrete approximations for convection–diffusion–reaction problems*, Math. Comput. Simulation 81 (2010) 820–836

[5] F.Guiaş, P. Eremeev, *Improving the stochastic direct simulation method with applications to evolution partial differential equations*, Applied Mathematics and Computation 289 (2016) 353–370

[6] W. Hundsdorfer, J.G. Verwer, Numerical Solution of Time-Dependent Advection–Diffusion–Reaction Equations, Springer–Verlag, Berlin, 2003

[7] T.G. Kurtz, *Limit theorems for sequences of jump Markov processes approximating ordinary differential processes*, J.Appl. Prob. 8 (1971) 344–356