Chemical Engineering Journal 155 (2009) 603–616

Contents lists available at ScienceDirect

Chemical Engineering Journal

journal homepage: www.elsevier.com/locate/cej

Simulation of (bio)chemical processes with distributed parameters using Matlab®

F. Logist[a], P. Saucez[b], J. Van Impe[a], A. Vande Wouwer[c,*]

[a] Department of Chemical Engineering, Katholieke Universiteit Leuven, W. de Croylaan 46, B-3001 Leuven, Belgium
[b] Service de Mathématique et Recherche Opérationnelle, Université de Mons (UMONS), Faculté Polytechnique, Rue de Houdain 9, B-7000 Mons, Belgium
[c] Service d'Automatique, Université de Mons (UMONS), Faculté Polytechnique, Boulevard Dolez 31, B-7000 Mons, Belgium

ARTICLE INFO

Article history:
Received 10 April 2009
Received in revised form 12 August 2009
Accepted 13 August 2009

Keywords:
Numerical methods
Partial differential equations
Method of lines
Operator splitting
Reverse flow reactor

ABSTRACT

Nowadays, simulations have become indispensable for the analysis and optimisation of (bio)chemical processes. However, as a lot of these processes are distributed in nature (i.e., the properties vary both in time and space), their simulation requires the solution of non-linear convection–reaction–diffusion partial differential equations (PDEs). Therefore, this paper compares different solution methods in a comprehensible way in order to provide practical guidelines. Moreover, to stimulate their usage in practice, all techniques have been implemented in Matlab®, and all test examples have been made available (www.matmol.org). This allows practitioners to rapidly evaluate different methods when setting-up their own process simulation code. Finally, a complex reverse flow reactor case illustrates how these methods can be successfully combined with optimisation approaches.

1. Introduction

As convection–reaction–diffusion processes are omnipresent in the (bio)chemical industry, accurate and efficient simulation techniques are valuable tools for process engineers (see, e.g. [1–3] for recent results). In particular, the combination with optimisation routines is attractive from an economic point of view. In literature, a vast amount of methods have been presented over the years to solve the partial differential equations (PDEs) involved, but no single method has been found to be best for all cases. Therefore, the major objective of this paper is to compare several easily comprehensible but versatile solution techniques, and to add practical guidelines.

The simulation techniques under study can be classified in two categories: *operator splitting* (OS) [4] and *method of lines* (MOL) [5] approaches. The former category tackles the different phenomena sequentially within each time step, adapting the integration method each time to the phenomenon at hand, whereas the latter category of methods simultaneously accounts for all present phenomena (conversion, reaction and diffusion) when solving the PDEs. The MOL approach is based on two steps: the spatial derivatives are first approximated, e.g., by finite differences or finite volumes, and then the system of semi-discrete equations is integrated in time. The success of this approach is due to the availability of efficient time integrators for solving the resulting mixed systems of (ordinary) differential and algebraic equations (DAEs), e.g., the Matlab® ODE suite [6]. As a simple and easily implementable example of an OS method, the *sequencing method* [7] is selected and adapted. For the MOL approaches, the MatMOL toolbox [8] is employed. This toolbox contains different simple (uniform and non-uniform) discretisation schemes, and recently more advanced features, i.e., flux limiters and adaptive grids [9–13], have been added. To compare the different techniques three test cases are studied: a jacketed tubular reactor, a fixed bed bioreactor, and the exploitation of an oil well. In these examples the diffusion can easily be varied from high to low, allowing also to evaluate the performance of the techniques in the presence of, e.g., (steep) moving fronts. Finally, to illustrate the successful combination of simulation and optimisation techniques for an industrial setting, the performance of a reverse flow reactor is optimised.

To promote the practical usage, all techniques have been coded in the widely spread and user-friendly software package Matlab® (The MathWorks Inc., Natick), and all codes have been made available on the internet. Since every test example has been coded as a Matlab® *m*-function, changing the function arguments allows to rapidly experiment with different options. Moreover, these example files can be used by practitioners as templates to compare different methods when developing their own process simulation codes.

The organisation of the paper is as follows. In Section 2 the mathematical formulation of convection–reaction–diffusion processes is introduced. In Section 3 different numerical approaches are dis-

* Corresponding author. Tel.: +32 65 37 41 41; fax: +32 65 37 41 36.
E-mail address: alain.vandewouwer@fpms.ac.be (A. Vande Wouwer).

1385-8947/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.cej.2009.08.017

cussed, while Section 4 compares their performance for the test cases and provides practical guidelines. Section 5 illustrates the successful optimisation of a more complex reverse flow reactor. Finally, the main conclusions are summarised in Section 6.

2. Mathematical formulation

Describing convection–reaction–diffusion processes by one-dimensional balance equations[1] often results in a set of n coupled partial differential equations with as independent variables time t and spatial coordinate z, and as dependent variable x:

$$\frac{\partial x}{\partial t} = \frac{\partial f_d}{\partial z} - \frac{\partial f_c}{\partial z} - r(x) \tag{1}$$

Here, the left hand side represents the accumulation of x over time in an (arbitrarily) small element, while the terms in the right hand side account for the difference between incoming and outgoing diffusive and convective fluxes (f_d and f_c, respectively), and the disappearance of x due to reaction, respectively. These fluxes often take the form $f_d = D(\partial x/\partial z)$ and $f_c = vx$, and when the diffusion coefficient D and the velocity v are constants, they can be taken outside the derivatives.

Mathematically, such PDEs are classified as *parabolic* PDEs. Whenever the diffusion is absent, the nature and the features of the PDEs change and they are called *hyperbolic* PDEs. Initial conditions (IC) and boundary conditions (BC) are required to complete the mathematical formulation. Initial conditions define the state of the system at the initial time:

$$x(z, t = 0) = x_0(z) \tag{2}$$

whereas the boundary conditions:

$$g\left(x(z = z_L \text{ or } z_R, t), \frac{\partial x}{\partial z}(z = z_L \text{ or } z_R, t), t\right) = 0 \tag{3}$$

relate to the transport to and from the system at the left and right boundary, i.e., z_L and z_R, respectively.

3. Solution methods and strategy

First of all, it should be emphasised that there exists no single best method for all convection–reaction–diffusion problems. Hence, when setting up a simulation code the developer will often have to test several techniques. Nevertheless, an in practice often acceptable strategy is, trying a MOL approach with simple linear finite differences or finite volumes schemes on fixed grids first. When the results are not satisfactory, (i) more elaborate non-linear MOL schemes, e.g., finite volumes with non-linear flux limiting functions or adaptive gridding, can be tried, or, (ii) alternatively, the possibilities of OS approaches can be explored. This section briefly recalls the most important features of the considered MOL and OS approaches. It should be noted that in the following subsections a constant diffusion D and velocity v are assumed for reasons of clarity. However, remarks for non-constant cases are added.

3.1. Method of lines

The rationale behind the MOL involves two successive steps: *spatial discretisation* of the PDEs and *time integration* of the resulting system. Whereas the discretisation of time-dependent PDEs leads to an ODE system, the discretisation of (usually static) boundary conditions results in algebraic equations (AEs), altogether forming a system of differential-algebraic equations (DAEs). This system of DAEs with discretised vectors $\mathbf{x}$, $\mathbf{x_z}$ and $\mathbf{x_{zz}}$ for the variable x and its first- and second-order derivative at n discretisation points $\mathbf{z}$ of a *vertex-centred* grid can be written as follows:

$$\frac{d\mathbf{x}}{dt} = D\mathbf{x_{zz}} - v\mathbf{x_z} - \mathbf{r}(\mathbf{x}) \tag{4}$$

$$\mathbf{g}(\mathbf{x}, \mathbf{x_z}, t) = 0 \tag{5}$$

$$\mathbf{x}(\mathbf{z}, t = 0) = \mathbf{x}_0 \tag{6}$$

3.1.1. Finite differences on fixed grids

As the discretisation points are now positioned at the interval boundaries, the interval length for a uniform grid is given by $\Delta z = L/(n-1)$. Applying a first- and second-order approximation on a uniform grid to the first- and second-order spatial derivatives, respectively:

$$\left.\frac{\partial x}{\partial z}\right|_{z_i} = \frac{x(z_{i-1}) - x(z_i)}{\Delta z} + \mathcal{O}(\Delta z) \tag{7}$$

$$\left.\frac{\partial^2 x}{\partial z^2}\right|_{z_i} = \frac{x(z_{i-1}) - 2x(z_i) + x(z_{i+1})}{\Delta z^2} + \mathcal{O}(\Delta z^2) \tag{8}$$

results in a classic *tanks-in-series* model. Higher-order schemes can be derived based on Taylor series expansions. For instance, a fourth-order biased upwind and a fourth-order centred approximation for the first- and second-order spatial derivative, respectively, are given in the following expressions:

$$\left.\frac{\partial x}{\partial z}\right|_{z_i} = \frac{-x(z_{i-3}) + 6x(z_{i-2}) - 18x(z_{i-1}) + 10x(z_i) + 3x(z_{i+1})}{\Delta z} + \mathcal{O}(\Delta z^4) \tag{9}$$

$$\left.\frac{\partial^2 x}{\partial z^2}\right|_{z_i} = \frac{-2x(z_{i-2}) + 32x(z_{i-1}) - 60x(z_i) + 32x(z_{i+1}) - 2x(z_{i+2})}{4!\,\Delta z^2} + \mathcal{O}(\Delta z^4) \tag{10}$$

The use of discretisation stencils is not restricted to uniform grids as the weighting coefficients can be computed for non-uniform grids based on an algorithm by Fornberg [14]. Combining the stencils at all discretisation points into *differentiation matrices* $\mathbf{D_1}$ and $\mathbf{D_2}$ enables a simple matrix-vector based computation of the derivatives: $\mathbf{x_z} = \mathbf{D_1}\mathbf{x}$ and $\mathbf{x_{zz}} = \mathbf{D_2}\mathbf{x}$. In addition, these matrices can also be used to approximate the flux derivatives in case of non-constant velocity and diffusion values, i.e., $\mathbf{f_{c,z}} = \mathbf{D_1}\mathbf{f_c}$ and $\mathbf{f_{d,z}} = \mathbf{D_1}\mathbf{f_d}$ with $\mathbf{f_d} = D\mathbf{D_1}\mathbf{x}$.

Although these linear differentiation matrices are extremely useful from a computation point of view, non-linear techniques may be required whenever the amount of diffusion is limited and sharp spatial transitions are involved. Classic linear techniques are known to produce either excessive *smoothing* of the profiles due to *numerical diffusion* (first-order schemes) or excessive *non-physical* oscillations due to *numerical dispersion* (high-order schemes). Observe that both numerical diffusion and dispersion are computational artefacts and, by consequence, may not be confused with their physical equivalents. Increasing the grid density, not only decreases these undesired effects, but also increases the computational burden. Hence, either an acceptable trade-off between computation time and accuracy has to be found or alternative (non-linear) methods have to be selected. Both *flux limiters* and *adaptive grids* have been proposed as non-linear methods to mitigate the undesired effects.

[1] Although 2D or 3D equations are equally possible, they result in higher computation times, while the increase in accuracy is not always significant.

Table 1
Flux limiting functions.

Flux limiter	Expression
Koren	$\max\left(0, \min\left(2r, \frac{1+2r}{3}, 2\right)\right)$
Mc	$\max\left(0, \min\left(2r, \frac{1+r}{2}, 2\right)\right)$
Minmod	$\max(0, \min(1, r))$
Smart	$\max\left(0, \min\left(2r, \frac{1+3r}{4}, 4\right)\right)$
Superbee	$\max(0, \min(2r, 1), \min(r, 2))$
Van Leer	$\frac{r+\lvert r\rvert}{1+\lvert r\rvert}$

3.1.2. Finite volumes with flux limiters

Flux or slope limiters for finite volume schemes have been introduced for approximating the derivative of the convective flux. In smooth parts high-order schemes $f_{c,ho}$ are adopted, whereas near sharp transitions first-order schemes $f_{c,lo}$ are preferred in order to avoid oscillations. Both approximations are combined into one formula using the limiting function $\phi(\mathbf{x})$ which measures the smoothness of a solution: $f_c(\mathbf{x}) = f_{c,lo}(\mathbf{x}) + \phi(\mathbf{x})[f_{c,ho}(\mathbf{x}) - f_{c,lo}(\mathbf{x})]$. The limiter is close to 1 in smooth regions and close to 0 in non-smooth regions. A popular limiter $\phi(r_i)$ is based on the ratio of consecutive gradients: $r_i = (x(z_i) - x(z_{i-1}))/(x(z_{i+1}) - x(z_i))$. If the ratio r is near 1 a smooth solution can be expected, whereas higher and lower values indicate the presence of sharp spatial transitions. In literature (e.g., [15]) several limiting functions have been proposed as well as *total variation diminishing* (or TVD) criteria, which ensure that a flux limiter causes the oscillations to decrease monotonously as time evolves. Examples of flux limiting functions are presented in Table 1. All (except the Smart limiter) satisfy the second-order TVD-criteria [15].

3.1.3. Adaptive grids

As the undesired effects decrease with increasing grid density, another option is to use non-uniform grids and concentrate the grid points in regions with sharp spatial transitions. This approach is much more efficient than using a dense uniform grid as no points are wasted in smooth regions. However, the grid has to be adapted to track the region of high spatial activity in time. The points are each time equally distributed (i) according to a monitor function $m(\mathbf{x})$ (which is often based on the arc length or curvature of the solution), and (ii) subject to regularity constraints (in order to provide upper and lower bounds on the interval length). The grid update procedure can either be performed periodically, when the time integrator is halted (*static regridding*) or can be incorporated inside the time integrator itself (*dynamic regridding*). The former technique is easier to implement due to the uncoupled integration and regridding steps, yields smaller integration problems and can easily accommodate new fronts. The latter technique has, however, always an optimal grid distribution (and, hence, cannot lag behind), has no computational overhead due to frequent integrator restarts and does not suffer from interpolation errors induced by uncoupling the integration and regridding steps [13].

3.1.4. Time integration

The second step in the MOL approach concerns the time integration of the semi-discrete system of DAEs. DAE (and linear implicit ODE) systems can be represented as $\mathbf{M}(\mathbf{x}, t)(\mathrm{d}\mathbf{x}/\mathrm{d}t) = \mathbf{f}(\mathbf{x}, t)$ with consistent initial conditions $\mathbf{x}(0) = \mathbf{x}_0$ and a *mass matrix* $\mathbf{M}$, which is singular in the DAE case. Non-stiff ODE systems (i.e., when all variables vary at comparable time scales) are most efficiently solved by explicit integrators, which compute values at the next time point only based on the current values. However, stiff ODEs (i.e., with largely varying time scales) and DAEs have to be treated by implicit integration routines. These implicit integrators have to solve a set of (non-)linear equations (requiring the Jacobian $\mathbf{J}$ of the system equations with $\mathbf{J}_{i,j} = \partial f_i/\partial x_j$) in order to find the values at the next point in time. These Jacobians can be either provided analytically to the solvers, or, they can be approximated by the solver itself via finite difference perturbations. Alternatively, when these Jacobians are large and sparse (which is the case for the DAEs resulting from the discretisation of PDEs), the sparsity pattern of the Jacobian can be provided to the routine in order to significantly reduce the number of finite difference perturbations for the Jacobian approximation.

However, as explicit integration methods are faster because no Jacobian evaluations are required, a first important practical aspect is to check reformulation strategies, which convert the original DAE system to an ODE system, enabling the use of a simpler (and possibly faster) explicit integrator.

- A *first approach* [8] is converting the (discretised) algebraic boundary conditions $\mathbf{g}(\mathbf{x}, \mathbf{x}_\mathbf{z}, t) = \mathbf{0}$ into ODEs with a small time constant τ_f, which quickly forces the deviation from the boundary condition towards zero: $\mathrm{d}\mathbf{x}(z_1 \text{ or } z_n)/\mathrm{d}t = (1/\tau_f)\mathbf{g}(\mathbf{x}, \mathbf{x}_\mathbf{z}, t)$. However, care should be taken for (i) a correct right hand side sign ensuring a decay and not a growth of the deviation, and (ii) an appropriate tuning of the time constant τ_f for each of the boundary conditions in order to achieve a fast response (and fast satisfaction) of the boundary condition, without leading to a set of stiff ODEs.
- A *second approach* [16] involves the explicit elimination of the algebraic equations from the DAE system by (i) expressing all boundary variables $x(z_1)$ and/or $x(z_n)$ as a function of the other variables using the (discretised) boundary conditions $\mathbf{g}(\mathbf{x}, \mathbf{x}_\mathbf{z}, t) = \mathbf{0}$ and (ii) substituting the obtained relations into the (discretised) differential equations.

Another important aspect for the time integration is *time balancing*, i.e., balancing the discretisation in space and time, because it is useless from an approximation point of view to have a fine time discretisation with a coarse spatial grid or vice versa.

3.1.5. Implementation in `Matlab`®: the MatMOL toolbox

The MatMOL toolbox contains a variety of discretisation stencils on uniform and non-uniform grids which have to be combined with `Matlab`®'s ODE suite [6,17]. These templates can be used in combination with the DAE solver `ode15s` (implicit integrator) or with the ODE solvers as `ode45`, `ode23` (explicit integrators), or `ode23s` (implicit integrator) when eliminating the algebraic equations originating from the boundary conditions or converting them into an ODE with a forcing function. With respect to time balancing, the integrators do not allow a direct control on the time step length. The only parameters influencing this property are the relative and absolute integration error tolerances `RelTol` and `AbsTol`. Since the default values of 10^{-6} are in general too low, it is recommended that a user, when coding his or her own application, tries several lower values in order to check whether tighter tolerances really improve the accuracy of the solution or only result in higher computation times.

In addition, more advanced features have been recently added to the MatMOL toolbox: (i) several flux limiter functions (e.g., Koren, MC, Minmod, Smart, Superbee, and Van Leer), (ii) the static regridding method `agereg` [12], and (iii) the dynamic regridding code based on the `movgrd` code [18,19]. In general, the DAE nature is indicated by adding the option `Mass`. Analytical Jacobians or their sparsity pattern can be supplied via the options `Jacobian` and `JPattern`, respectively. All these matrices have to be provided as additional (nested) m-functions, and for computational efficiency, they are best stored as `sparse`. The flux limiters are easily incorporated as they only replace the first-order derivatives of $v\mathbf{x}_\mathbf{z}$. Although providing the Jacobian analytically is no more feasible in this case, it is still advantageous to provide its sparsity pattern. In the static regridding code the implicit integrator `ode23s` is employed. In contrast to multi-step integrators, this one-step integrator only needs the current values of the system (and no additional earlier values) which is advantageous with respect to the repeated solver restarts and the corresponding computational overhead. To stop the integration each $n_{\max}$ integration steps, an event function is specified which halts the integrator when it reaches zero. At each stop the grid is adapted based on one of the monitor functions $m(\mathbf{x}) = \sqrt{\alpha + \sum_{i=1}^{n} {\mathbf{x_z}}_i^2}$ or $m(\mathbf{x}) = \sqrt{\alpha + \max \mathbf{x_{zz}}}$, and the dependent variables are interpolated on the new grid. In the dynamic regridding code, linearly coupled ODEs which determine the grid movement, are added to the original DAEs, resulting in a banded mass matrix $\mathbf{M}$, which now also depends on the states. To facilitate the computation, the `MStateDependence` option has to switched to `strong` due to the high dependence of this mass matrix on the states. With respect to the monitoring function, several possibilities are available.

3.2. *Operator splitting methods*

The rationale behind operator splitting, or also called *time splitting* methods, is to split the original convection–reaction–diffusion PDEs into different parts which are solved sequentially within each time step, in order to take advantage of solution methods that are highly adapted to each of the different parts [4,20]. In general, splitting algorithms can be classified into two and three step procedures based on the number of parts.

Traditional two step schemes involve the isolation of the reaction part, yielding a linear convection–diffusion PDE, and a system of non-linear ODEs:

$$\frac{\partial x}{\partial t} = D\frac{\partial^2 x}{\partial z^2} - v\frac{\partial x}{\partial z} \quad \text{and} \quad \frac{\mathrm{d}x}{\mathrm{d}t} = -r(x) \tag{11}$$

Typically, the linear convection–diffusion equation is solved using a standard finite differences approach, while the reaction part is solved with an appropriate ODE integrator. However, for convection dominated problems, this scheme can be quite computationally expensive due the fine grid required for solving the convection–diffusion PDE. Alternatively, a *non-traditional two step scheme*, which isolates the diffusion term from the equation, leading to a linear diffusion equation, and a non-linear convection–reaction equation, has recently been reported [20]:

$$\frac{\partial x}{\partial t} = D\frac{\partial^2 x}{\partial z^2} \quad \text{and} \quad \frac{\partial x}{\partial t} = -v\frac{\partial x}{\partial z} - r(x) \tag{12}$$

Although the diffusion PDE can now easily be solved using finite differences and an implicit integrator, flux limiters will be required for the solution of the non-linear hyperbolic convection–reaction PDE.

Three step splitting algorithms separate all phenomena, i.e., diffusion, convection and reaction, yielding two linear PDEs, and a non-linear system of ODEs:

$$\frac{\partial x}{\partial t} = D\frac{\partial^2 x}{\partial z^2}; \qquad \frac{\partial x}{\partial t} = -v\frac{\partial x}{\partial z} \quad \text{and} \quad \frac{\mathrm{d}x}{\mathrm{d}t} = -r(x) \tag{13}$$

Each of these steps can now be solved by a technique appropriate to the nature of each phenomenon. The splitting of (non-homogeneous) boundary conditions is not always straightforward, and is often still based on *trial-and-error* as a general procedure and analysis is still lacking [4].

An additional, important question concerns whether or not the solution sequence of the different parts has an influence on the final solution. In the work of [4] and [21] necessary conditions for commutativity are given.

1. Convection commutes with diffusion, if the velocity v and the diffusion coefficient D do not explicitly depend on the spatial coordinate z.
2. Convection commutes with reaction, if the velocity v and the reaction $r(x)$ do not explicitly depend on the spatial coordinate z.
3. Diffusion commutes with reaction, if the reaction $r(x)$ is linear in x, and independent of the spatial coordinate z.

In general, the first two necessary conditions are satisfied. The third condition is, however, often not fulfilled, which may lead to different solutions for different sequences [7]. However, the exact sequence can be found by comparison with a low-order finite differences method.

3.2.1. *Sequencing method*

The *sequencing method*, developed by Renou et al. [7], is a three step splitting procedure. Within each time interval $t \in [t^*, t^* + \Delta t]$ the convection, the reaction, and the diffusion part are solved sequentially. Each independent variable x is uniformly spatially discretised on n intervals with a length of $\Delta z = L/n$, resulting in a *cell-centred* discretised profile $\mathbf{x}$ (i.e., which contains the values discretised in the middle of each of the intervals).

The order of the different phenomena is the following: convection, reaction, and diffusion. Hence, first the convection PDE is solved for a time step Δt with as initial condition the original profile, yielding a profile $\mathbf{x}^*$. This profile $\mathbf{x}^*$ is then used as initial condition for solving the reaction ODE over the interval Δt, resulting in a profile $\mathbf{x}^{**}$. This result $\mathbf{x}^{**}$ is again used as an initial condition, but now for the diffusion PDE. This provides the final solution $\mathbf{x}^{***}$ for the current time interval. This solution is set equal to the starting profile $\mathbf{x}(t + \Delta t) = \mathbf{x}^{***}$ for the time interval $[t^* + \Delta t, t^* + 2\Delta t]$, where it will undergo again the convection, reaction and diffusion steps. (See also [7] for an outline of the algorithm.)

Numerically, convection over a time Δt is implemented as a pure delay, i.e., shifting the profile $\mathbf{x}(t = t^*)$ over a distance $\Delta z = v\Delta t$, in the direction of the flow to yield $\mathbf{x}^*$. Note the explicit relation between the time and the spatial discretisation, and the requirement of a constant velocity v.

Second, the reaction part for an interval Δt has to be solved starting from the shifted profile: $\mathrm{d}\mathbf{x}/\mathrm{d}t = -\mathbf{r}(\mathbf{x}, t)$ with $\mathbf{x}(t = t^*) = \mathbf{x}^*$ resulting in $\mathbf{x}^{**}$. If possible, an explicit analytical solution can be used, e.g., for a linear reaction term, otherwise the system of ODEs has to be integrated explicitly in time over an interval Δt. This can be done by simple forward Euler integration or by the use of more sophisticated ODE solvers.

Finally, the (discretised) diffusion problem is tackled using the results from the reaction part. Since the resulting ODE system is linear:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = D\mathbf{x_{zz}} = D\mathbf{D}_2\mathbf{x} \quad \text{with } \mathbf{x}(t = t^*) = \mathbf{x}^{**} \tag{14}$$

it can be solved for a given Δt using a transition matrix formulation:

$$\mathbf{x}(t^* + \Delta t) = \mathbf{x}^{***} = \exp(D\mathbf{D}_2\Delta t)\mathbf{x}^{**} \tag{15}$$

where $\mathbf{D}_2$ is a differentiation matrix for the second-order derivative. This matrix can be formulated using the finite differences methods of Section 3.1.1. Because the diffusion coefficient D, the second-order differentiation matrix $\mathbf{D}_2$, and the time step Δt are constant, the exponential matrix is always the same, and can be computed in advance. Hence, the solution of the diffusion system at each time step only requires a single matrix vector multiplication.

3.2.2. *Adaptations to the sequencing method*

The original algorithm can be adapted in several ways. The second-order differentiation matrix can be replaced by higher-order schemes, e.g., the fourth-order scheme. Additionally, to take advantage of *linearity*, the reaction part can be linearised around the current values. The obtained linearised system can be solved with a transition matrix formulation alleviating the need for solving the system of ODEs. Also different sequences in which the different phenomena are solved within one time step can be tried.

3.2.3. Implementation of the sequencing method in `Matlab`®

The implementation in the sequencing method in `Matlab`® is straightforward as mainly vector and matrix operations (e.g., multiplications, rearranging and matrix exponentials) are involved. At the beginning, a fixed-size solution matrix (with *Np* spatial points and *Nt* time points) is specified. This matrix is then gradually adapted in a `for`-loop over all time intervals. Within each time interval the different subproblems are each time solved in the same order. The final solution of each time interval, i.e., after all phenomena have applied sequentially, is then stored in the solution matrix. An outline of the algorithm for the sequencing method can be found in Renou et al. [7].

The default sequence is convection–reaction–diffusion. However, since only reaction and diffusion may not commute for constant velocity and diffusion values, the alternative convection–diffusion-reaction sequence can also be tried. The solution of the convection subproblem is implemented as a pure shift of the current profile in the direction of the flow, i.e., all elements are shifted one position, e.g., from position i to $i+1$, while the open position $i=1$ is occupied by the value of the input stream fed to the process. Note however, that this action requires the satisfaction of the relation $\Delta z = v\Delta t$. To solve the reaction subproblem, a simple explicit forward Euler scheme can be implemented, or more advanced integrators from `Matlab`®'s ODE suite can be exploited. Again explicit integration schemes, e.g., forward Euler, `ode23`, `ode45`, and `ode113` will be faster for non-stiff systems, while for stiff systems (e.g., in the case of slow and fast reactions) the computation time for implicit integrators will be lower. Alternatively, instead of using the ODE suite to solve the reaction part, the states at the next time point can be approximated by using a matrix exponential formulation of the linearised reaction equations. The solution of the diffusion subproblem is obtained by multiplying the current profile with the constant exponential matrix $\exp(D\mathbf{D}_2\Delta t)$, which has been precomputed before the loop starts using the built-in `Matlab`® function `exp`. The default differentiation matrix $\mathbf{D}_2$ is based on a second-order approximation, but a modified fourth-order approximation is also available.

4. Test cases and results

In the current section different methods are tested and compared for three (bio)chemical case studies: (i) a jacketed tubular reactor, (ii) a fixed bed bioreactor and (iii) an oil well exploitation example. In the next section, a real-life industrial case will be studied. In the first two examples, the convective and diffusive fluxes exhibit constant velocity and diffusion parameters, while in the third case these variables depend non-linearly on the dependent variables. Although apparently simple, these examples have been selected as the nature and difficulty of the PDEs can be flexibly adapted by changing the diffusion level, i.e., from easy when diffusion dominates (parabolic PDEs), to hard when convection is most prominent (hyperbolic PDEs). In the latter case, hard to capture steep moving fronts can be present, while these fronts are in general smoothed in the former case by the (high level of the) diffusion. Implementation details about the methods that will be tested are given in Section 4.2.

Table 2
Parameter values.

DFR		FBBR	
Parameter	Value	Parameter	Values
L	1.0 m	L	1.0 m
t_{sim}	20 s	t_{sim}	2 h
v	0.1 m/s	v	1.0 m/h
C_{in}	0.02 mol/L	S_{in}	20.0 g/L
T_{in}	340.0 K	k	2.0 g/g
E	11250.0 cal/mol	k_d	0.01 1/h
R	1.986 cal/mol K	μ_0	0.4 1/h
k_0	1.0×10^6 1/s	K_S	1.0 g/L
$\frac{4h}{\rho C_p d}$	0.2 1/s	K_i	1.0 g/L
$\frac{\Delta H}{\rho C_p}$	$0.25\times T_{\text{in}}/C_{\text{in}}$		
$T_{w,\text{min}}$	280.0 K		
$T_{w,\text{max}}$	400.0 K		
z_{sw}	0.54 m		

4.1. Model equations

4.1.1. Jacketed tubular reactor

The jacketed tubular reactor under study is a classic dispersive flow reactor (DFR) of length L [m] in which an irreversible, exothermic, first-order reaction takes place. The jacket consists of a heating and cooling part [22]:

$$\frac{\partial T}{\partial t} = D_1\frac{\partial^2 T}{\partial z^2} - v\frac{\partial T}{\partial z} - \frac{\Delta H}{\rho C_p}k_0 C e^{-E/RT} + \frac{4h}{\rho C_p d}(T_w - T) \tag{16}$$

$$\frac{\partial C}{\partial t} = D_2\frac{\partial^2 C}{\partial z^2} - v\frac{\partial C}{\partial z} - k_0 C e^{-E/RT} \tag{17}$$

subject to four Danckwerts boundary conditions [23]:

$$D_1\frac{\partial T(0,t)}{\partial z} = v(T(0,t) - T_{\text{in}}); \qquad \frac{\partial T(L,t)}{\partial z} = 0$$
$$D_2\frac{\partial C(0,t)}{\partial z} = v(C(0,t) - C_{\text{in}}); \qquad \frac{\partial C(L,t)}{\partial z} = 0$$

and two initial conditions reflecting the empty state at start-up:

$$T(z,0) = T_0(z) = T_{\text{in}} = 340\,\text{K}$$
$$C(z,0) = C_0(z) = 0\,\text{mol/L}$$

Here, C [mol/L] and T [K] indicate the reactant concentration and the temperature, D_1 [m^2/s] and D_2 [m^2/s] are the energy and mass dispersion[2] coefficients, v [m/s] the fluid velocity, $-\Delta H$ [J/kmol] the heat of reaction ($\Delta H < 0$ for an exothermic reaction) and ρ [kg/m^3], C_p [J/kgK], k_0 [1/s], E [J/mol], R [J/mol K], h [W/m^{2}K] and d [m] the fluid density, the specific heat, the kinetic constant, the activation energy, the ideal gas constant, the heat transfer coefficient and the reactor diameter, respectively. The jacket fluid temperature T_w [K] switches at the switching position z_{sw} [m] from its maximum value $T_{w,\text{max}}$ [K] to its minimum $T_{w,\text{min}}$ [K]. The simulated time t_{sim} is 20 s. The mass and heat dispersion coefficients D_1 and D_2 are assumed to be equal and take the value ($D = 10^{-1}$, 10^{-3} and 10^{-5} m^2/s). (Alternatively, also the Peclet number $Pe = vL/D$ can be used to indicate the importance of convective to dispersive transport.) The other parameter values originate from [22,24–26] and are summarised in Table 2.

[2] In the numerical literature the second-order terms are generally called *diffusion* terms whereas in chemical engineering these terms are often known as *dispersion* terms. However, in the remainder of the text, the numerical term diffusion will be mostly employed to indicate these second-order contributions.

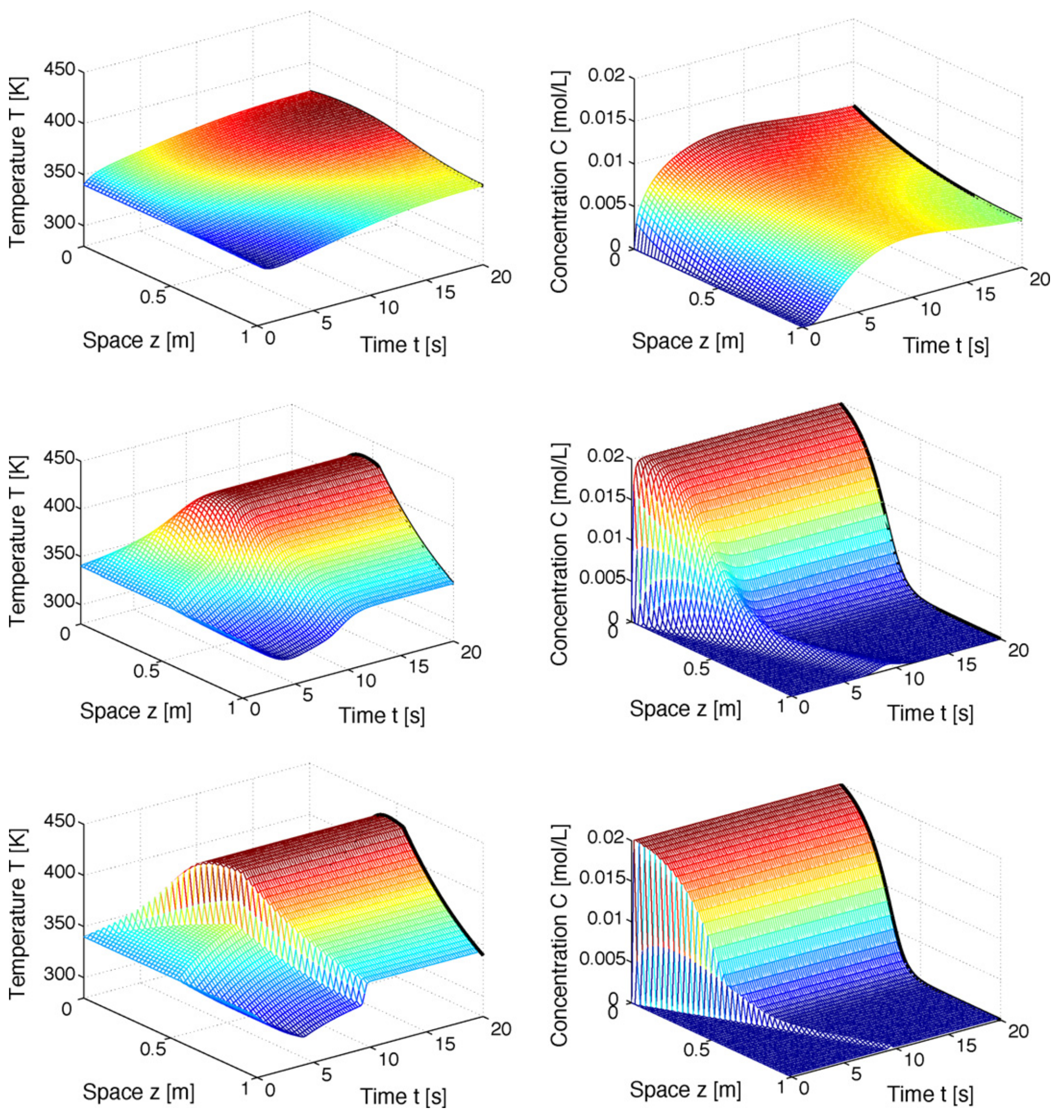

Fig. 1. Jacketed tubular reactor: reference evolutions for temperature (left) and concentration (right) for different diffusion values, i.e., $D = 10^{-1}$ m^2/s (top), 10^{-3} m^2/s (middle), and 10^{-5} m^2/s (bottom).

4.1.2. *Fixed bed bioreactor*

As a biochemical application, the fixed bed bioreactor (FBBR) [27], which may represent an anaerobic digester for waste water treatment, has been selected. In this case the biomass X [g/L] which has been fixed on a bed inside a tubular reactor of length L [m], grows on a limiting substrate S [g/L] that is fed at the reactor inlet:

$$\frac{\partial S}{\partial t} = D\frac{\partial^2 S}{\partial z^2} - v\frac{\partial S}{\partial z} - k\mu(S,X)X \qquad (18)$$

$$\frac{\partial X}{\partial t} = -k_d X + \mu(S,X), X \qquad (19)$$

where the specific growth rate μ [1/h] involves both substrate and biomass inhibition:

$$\mu(S,X) = \mu_0 \frac{S}{K_S X + S + (1/K_i)S^2}$$

The boundary and initial conditions are

$$D\frac{\partial S(0,t)}{\partial z} = v(S(0,t) - S_{\text{in}}); \qquad \frac{\partial S(L,t)}{\partial z} = 0$$

and

$$S(z,0) = S_0(z) = S_{\text{in}} = 20\,\text{g/L}$$
$$X(z,0) = X_0(z) = 300\,\text{g/L}$$

Here, D [m^2/h], k [g/g], k_d [1/h], K_S [g/g], K_i [g/L], S_{in} [g/L], v [m/s] and μ_0 [1/h] are the dispersion coefficient, the yield coefficient, the death rate, the biomass inhibition constant, the substrate inhibition constant, the substrate inlet concentration, the fluid velocity and the maximum specific growth rate, respectively. A simulated time t_{sim} of 2 h is adopted. The dispersion coefficient D is varied ($D = 10^0$, 10^{-2} and 10^{-4} m^2/s), while the other parameter values are taken from [27] (see also Table 2). (Also here the Peclet number can be used to indicate the dispersion values.)

4.1.3. *Oil well exploitation*

The oil well exploitation (OWE) example is described by the classic Buckley–Leverett equation [28] with capillary pressure:

$$\frac{\partial S}{\partial t} = \frac{\partial}{\partial z}\left(4\epsilon S(1-S)\frac{\partial S}{\partial z}\right) - \frac{\partial}{\partial z}\left(\frac{S}{S^2+(1-S)^2}S\right) \qquad (20)$$

with initial conditions:

$$S(z,0) = \begin{cases} 1 - 3 \cdot z & \text{if } z \in [0, 1/3] \\ 0 & \text{if } z \in [1/3, 1] \end{cases} \tag{21}$$

where S indicates the water saturation and ϵ the capillary constant. Here, the velocity and diffusion terms are clearly dependent on the dependent variable S. The simulated time t_{sim} is 0.4 s. As before different values for the diffusion parameter ϵ are adopted ($\epsilon = 10^{-1}$, 10^{-2} and 10^{-3}).

4.2. Selected methods for comparison

Due to the implementation via *m*-functions a large number of different algorithms can be tested by changing the function's arguments. However, in view of brevity, quantitative results for only the seven major algorithms are presented. **SM1** and **SM2** indicate the original sequencing method as described by Renou et al. [7] using for the reaction part an explicit Euler scheme and the explicit integrator `ode45`, respectively. **MM1** and **MM2** are based on the MatMOL toolbox with fixed finite differences on a uniform grid and solve the resulting DAE with `ode15s`. The former algorithm employs low-order discretisation schemes (corresponding to the *tanks-in-series* approximation), while the latter uses high-order stencils. Advanced features as *flux/slope limiters*, *static regridding*, and (iii) *dynamic regridding* techniques are implemented in **MM3**, **MM4**, and **MM5**, respectively. Different uniform grids are employed (N = 50, 100 and 250) to enable an appropriate trade-off between accuracy and computation time. It should be mentioned that N concerns the number of points for a *cell-centred* spatial grid (used in the sequencing method), while $N + 1$ points are required for a *vertex-centred* spatial grid (employed in the MatMOL toolbox) in order to have spatial intervals of equal length.

4.3. Measures and references

The comparison between the different methods is based on several measures. To measure the accuracy, the relative deviation RE of a solution profile $x(z, t)$ is compared with a reference profile $x_{ref}(z, t)$ based on the function 1-norm:

$$RE(x(z,t)) = \frac{\|x_{ref}(z,t) - x(z,t)\|_1}{\|x_{ref}(z,t)\|_1} \tag{22}$$

with as function 1-norm the L^1 norm:

$$\|f(z,t)\|_1 = \left(\int_0^L |f(z,t)| \, dz \right). \tag{23}$$

A time averaged relative deviation (ARE) is computed over the simulated period, employing the trapezium rule to approximate the integral in the norm definition. For the biochemical and chemical reactor cases, the reference transient solutions are obtained with **MM2** on a 501 ($N + 1$) point grid for the high dispersion cases (i.e., $D = 1$–10^{-1} and 10^{-2}–10^{-3}), and by **SM2** on a 500 point grid for the low diffusion cases (i.e., $D = 10^{-4}$–10^{-5}). For the reference steady-state solutions for the jacketed tubular reactor the profiles obtained via a shooting method [22] are selected. For the oil well example the references are given by **MM2** on a 501 ($N + 1$) point grid. Other interesting points for comparison are (i) the problem size Np (i.e., the number of discretisation points times the number of equations), (ii) the number of integration steps taken by the integrator (Nsteps), (iii) the number of times the right hand side function of the DAEs/ODEs is evaluated (Nfevals), and (iv) the computation time (CPU), measured when using `Matlab`® 7.0 and a computer with an Intel 1.86 GHz processor and 2 GB RAM, as well as (iv) the ratio of simulated to computation time (R).

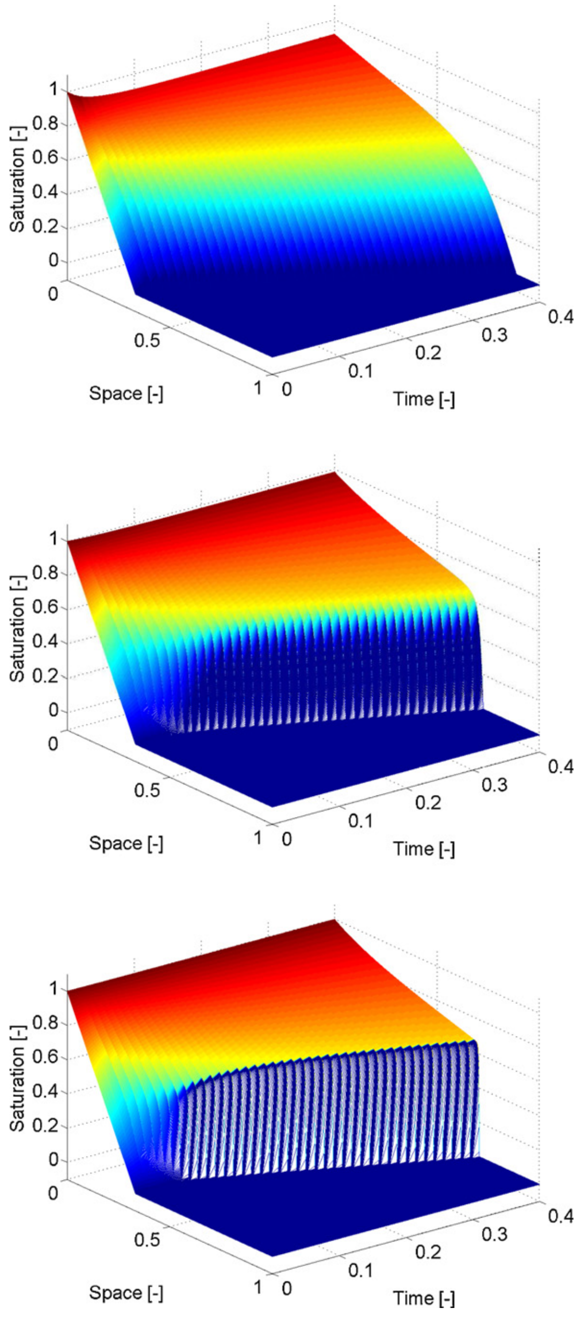

Fig. 2. Oil well exploitation: reference evolutions for water saturation for different capillary values, i.e., $\epsilon = 10^{-1}$ (top), 10^{-2} (middle), and 10^{-3} (bottom).

4.4. General results

Since the results for the FBBR are in general similar to ones for the DFR, they are not discussed in detail in view of brevity. Nevertheless, the corresponding *m*-files are also made available. Fig. 1 displays the reference solutions for the temperature and concentration in the DFR. As can be seen, the fluid flows from the left to the right and, as expected, the fronts become steeper, when diffusion decreases. The steady-state profiles have been added at the final time, and it is clear that convergence is safely achieved after 20 s. Fig. 2 depicts the reference solution of the water saturation for a time span of 0.4 s. Clearly, the water front sharpens and moves to the right as time evolves. Also, the influence of ϵ is similar to that of D. Lower values lead to steeper fronts. Due to space restrictions, Figs. 3 and 4 depict snapshots of the transient (spatial)

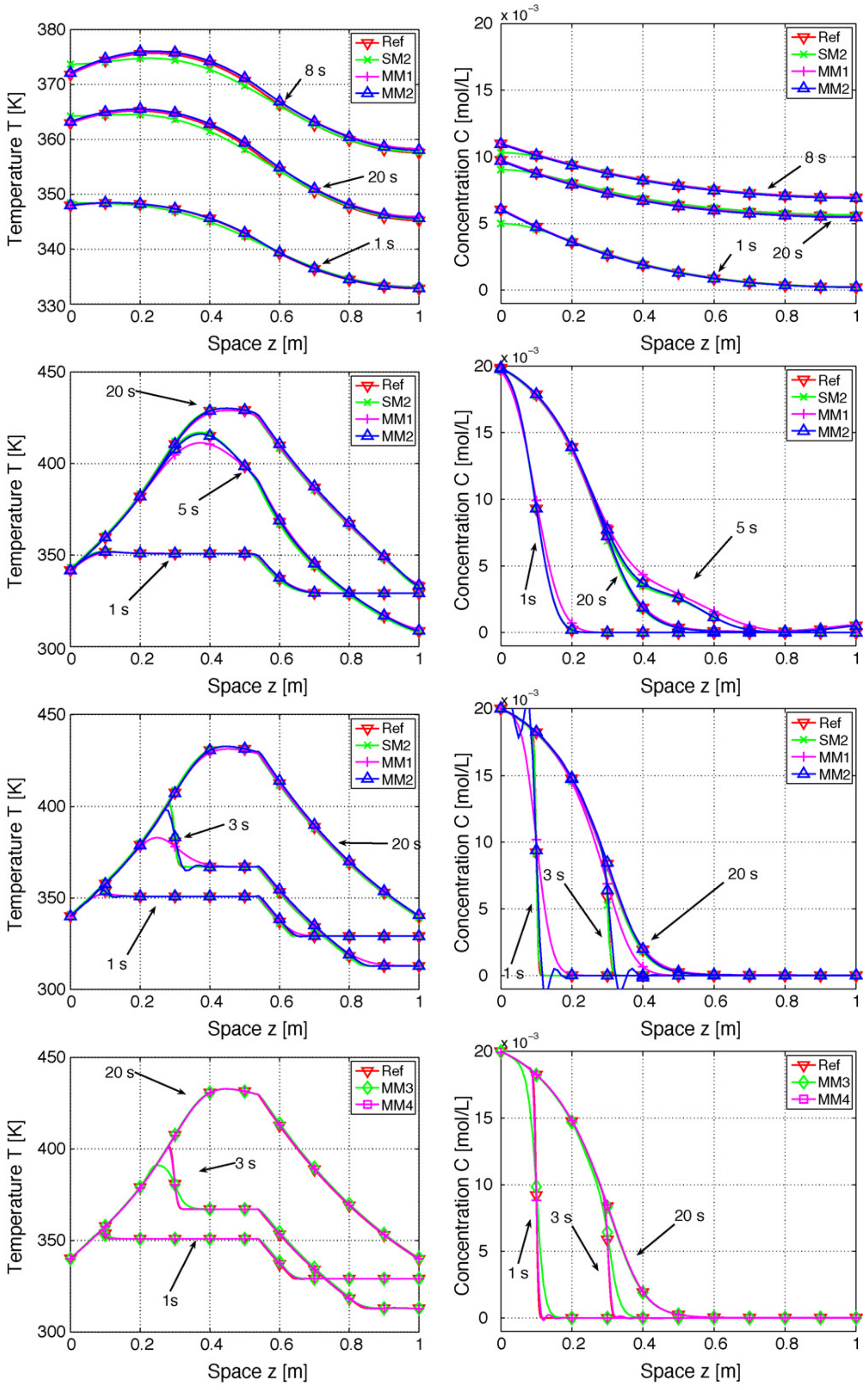

Fig. 3. Jacketed tubular reactor: snapshots of spatial profiles for temperature (left) and concentration (right) for the different methods for different diffusion values, i.e., $D = 10^{-1}$ (top), 10^{-3} (second row), and 10^{-5} (third row and bottom).

concentration and temperature profiles based on 100 discretisation intervals for only the basic methods (**SM2**, **MM1**, **MM2**, **MM3**, **MM4**, and **MM5**) in order to qualitatively illustrate their general features. Table 3 on the other hand provides a more quantitative and detailed description. With respect to the computation time, it has to be noted that in most of the cases for the DFR and FBBR the computation time is much smaller than the simulated time, while this is not true for the OWE example. Explanations are the smaller time scales and the non-linear convection term involved in the latter example.

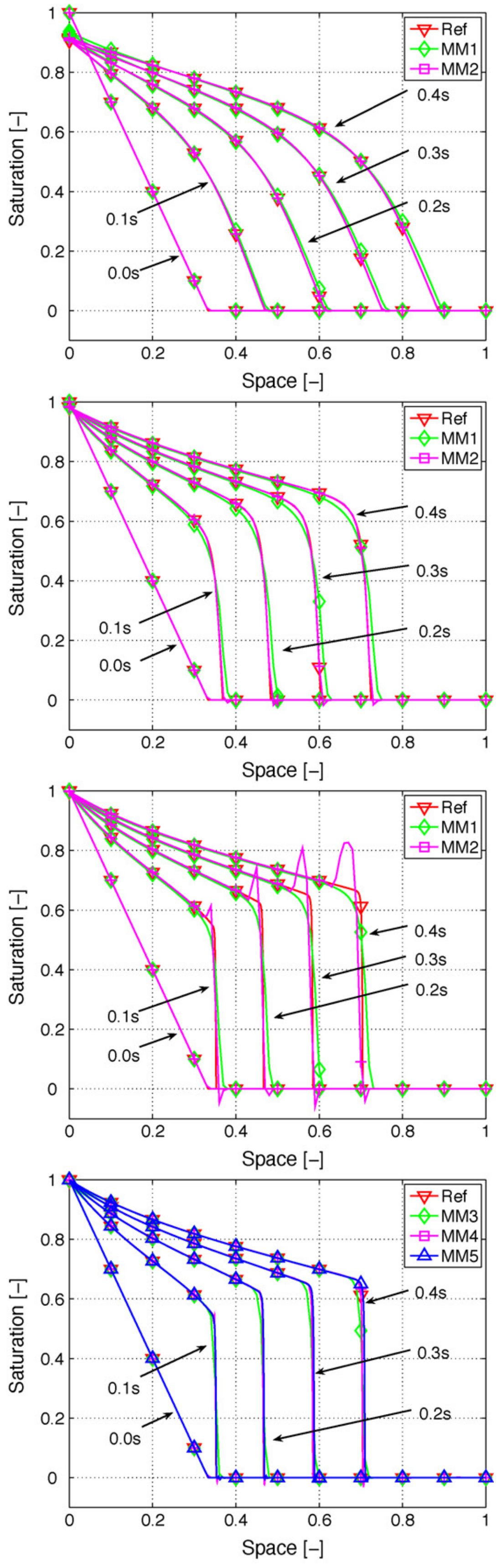

Fig. 4. Oil well exploitation: snapshots of spatial profiles for water saturation for the different methods for different capillary values, i.e., $\epsilon = 10^{-1}$ (top), 10^{-2} (second row), and 10^{-3} (third row and bottom).

4.5. MOL approaches

As a first remark, it should be noted that the MOL approaches are able to yield accurate results for cases with constant velocity and diffusion coefficients (e.g., DFR and FBBR) as well as for cases in which they are function of the dependent variables (e.g., OWE).

4.5.1. Finite differences on fixed grids

With respect to the discretisation order, the following observations are made. For low diffusion values accurate profiles are obtained for both the low-order scheme **MM1** (i.e., the *tanks-in-series* model) and the high-order scheme **MM2**. Although for $D = 10^{-1}$ m^2/s and $\epsilon = 10^{-1}$ a similar accuracy is observed, the high-order solutions are generally more accurate than their low-order equivalents. Nevertheless, for low diffusion values (e.g., $D = 10^{-5}$ m^2/s and $\epsilon = 10^{-3}$) inaccuracies in the movement of the sharp fronts appear: (i) excessive smearing due to *numerical diffusion* for the low-order scheme and (ii) non-physical oscillations (in front of and/or behind the sharp front) due to *numerical dispersion* for the high-order scheme. From a computational point of view, it is noticed that in general the low-order scheme is slightly faster than the high-order scheme. Increasing the number of grid points decreases the undesired effects, but also leads to higher computation times.

Concerning time balancing, Fig. 5 depicts the averaged transient relative errors and the computation time versus the integration tolerances for $D = 10^{-3}$ m^2/s and $\epsilon = 10^{-2}$ with a grid of 251 points. Clearly, the averaged relative deviations remain more or less constant up to a certain integration tolerance after which a sharp increase is observed. This constant level is higher for the low-order scheme (**MM1**) than for the high-order scheme (**MM2**). Also the point of increase lies at higher tolerances for **MM1**. The computation time increases as lower tolerances are requested. Hence, an acceptable trade-off between computation time and accuracy has to be found. For the cases under study this trade-off is located around integration tolerance values of 10^{-4}–10^{-3}. Therefore, although the user is recommended to test some different values when coding his own application, 10^{-3} can often be regarded as an acceptable starting value.

For the introduction of the boundary conditions, the results are briefly highlighted. The *first approach* (based on forcing the deviations from the boundary conditions towards zero) turned out to be cumbersome since a careful and time consuming tuning procedure is required for each of the different time constants τ_f. Too high time constants yield inaccurate transient profiles, while too low values result in a stiff ODE system, which typically requires high computation times, when solved with the non-stiff integrator `ode45`. The *second approach*, involving the elimination of the algebraic equations, yields results which hardly differ from the ones found when solving the original DAEs with the integration routine `ode15s`. The computation time is, unfortunately, only slightly lower. In summary, the effort to convert the DAE into an equivalent ODE in order to solve it with an explicit solver, is not worth the limited decrease in computation time for the current cases.

4.5.2. Finite volumes with flux limiters

To mitigate non-physical oscillations, several flux limiting functions can be exploited. In general, the oscillations (in front of and behind the sharp front) are smoothed out. For instance, all negative concentrations are eliminated. The various limiting functions behave rather similarly, only a minor influence in the amount of smoothing is observed. The Superbee limiter, exhibiting the sharpest front and the most oscillatory behaviour, smooths the least, whereas the Minmod limiter, displaying the least oscillations but also the smoothest front, obviously smooths the most. The com-

Table 3
Detailed overview of test results for the jacketed tubular reactor (DFR), the fixed bed bioreactor (FBBR) and the oil well exploitation (OWE).

DFR		Np			Nsteps			Nfeval			CPU			R			ARE1 *C*			ARE1 *T*		
D	Method	50	100	250	50	100	250	50	100	250	50	100	250	50	100	250	50	100	250	50	100	250
1E−1	SM1	100	200	500	NA	NA	NA	100	200	500	0.04	0.10	0.69	0.002	0.005	0.035	2.83E−2	1.64E−2	8.17E−3	2.21E−3	1.24E−3	5.79E−4
	SM2	100	200	500	100	200	500	700	1400	3500	0.26	0.57	2.30	0.013	0.029	0.115	2.74E−2	1.58E−2	7.78E−3	2.32E−3	1.31E−3	6.49E−4
	MM1	102	202	502	79	85	92	124	122	135	0.13	0.34	3.61	0.007	0.017	0.181	1.13E−2	5.12E−3	2.04E−3	2.31E−3	1.01E−3	2.62E−4
	MM2	102	202	502	77	83	92	126	131	141	0.13	0.36	3.55	0.007	0.018	0.178	1.67E−2	7.52E−3	1.98E−3	2.44E−3	1.09E−3	2.73E−4
1E−3	SM1	100	200	500	NA	NA	NA	100	200	500	0.04	0.07	0.40	0.002	0.004	0.020	2.54E−2	1.31E−2	5.33E−3	4.73E−3	2.46E−3	1.09E−3
	SM2	100	200	500	100	200	500	700	1400	3500	0.26	0.55	2.07	0.013	0.028	0.104	3.39E−2	1.71E−2	6.90E−3	4.31E−3	2.26E−3	1.01E−3
	MM1	102	202	502	109	132	154	170	201	219	0.16	0.49	4.78	0.008	0.025	0.239	5.11E−2	2.64E−2	1.08E−2	3.21E−3	1.67E−3	7.17E−4
	MM2	102	202	502	127	144	159	191	213	228	0.19	0.57	4.92	0.010	0.029	0.246	1.64E−3	5.42E−4	1.14E−4	1.65E−3	7.24E−4	1.80E−4
1E−5	SM1	100	200	500	NA	NA	NA	100	200	500	0.03	0.07	0.31	0.002	0.004	0.016	1.97E−2	8.39E−3	3.23E−3	4.41E−3	2.00E−3	5.43E−4
	SM2	100	200	500	100	200	500	700	1400	3500	0.24	0.53	1.97	0.012	0.027	0.099	3.26E−2	1.43E−2	3.52E−3	3.93E−3	1.75E−3	4.37E−4
	MM1	102	202	502	121	158	224	186	228	302	0.17	0.57	6.25	0.009	0.029	0.313	7.25E−2	4.26E−2	2.12E−2	4.85E−3	3.04E−3	1.77E−3
	MM2	102	202	502	223	335	573	345	416	685	0.32	1.20	16.47	0.016	0.060	0.824	2.09E−2	1.15E−2	5.10E−3	2.71E−3	1.54E−3	8.24E−4
	MM3a	102	202	502	894	1759	NC	4731	10037	NC	29.41	124.71	NC	1.471	6.236	NC	3.46E−2	1.66E−2	NC	3.52E−3	1.87E−3	NC
	MM3b	102	202	502	699	3230	NC	3190	20437	NC	20.33	251.50	NC	1.017	12.58	NC	3.15E−2	1.34E−2	NC	3.40E−3	1.71E−3	NC
	MM3c	102	202	502	800	3955	NC	3871	23807	NC	25.50	309.23	NC	1.275	15.46	NC	3.36E−2	1.46E−2	NC	3.53E−3	1.79E−3	NC
	MM3d	102	202	502	551	5631	NC	1545	33106	NC	11.36	431.59	NC	0.568	21.58	NC	2.94E−2	1.20E−2	NC	3.39E−3	1.70E−3	NC
	MM3e	102	202	502	649	2633	NC	3373	16066	NC	20.67	190.75	NC	1.034	9.538	NC	3.08E−2	1.37E−2	NC	3.34E−3	1.71E−3	NC
	MM3f	102	202	502	462	4014	NC	1658	25378	NC	9.79	309.03	NC	0.490	15.45	NC	2.94E−2	1.26E−2	NC	3.27E−3	1.66E−3	NC
	MM4	102	202	502	760	890	980	10336	12104	13328	11.06	11.87	14.11	0.553	0.594	0.706	8.57E−3	5.27E−3	4.28E−3	5.45E−4	5.46E−4	4.38E−4
	MM5	102	202	502	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC

FBBR		Np			Nsteps			Nfeval			CPU			R			ARE1 *C*			ARE1 *T*		
D	Method	50	100	250	50	100	250	50	100	250	50	100	250	50	100	250	50	100	250	50	100	250
1E−0	SM1	100	200	500	NA	NA	NA	100	200	500	0.02	0.04	0.62	5.22E−6	1.22E−5	1.72E−4	1.04E−4	5.60E−5	2.42E−5	1.75E−2	9.28E−3	3.91E−3
	SM2	100	200	500	100	200	500	700	1400	3500	0.24	0.53	2.07	6.67E−5	1.48E−4	5.74E−4	9.18E−5	4.85E−5	2.07E−5	1.78E−2	9.48E−3	3.99E−3
	MM1	102	202	502	148	161	178	183	203	220	0.16	0.42	3.62	4.46E−5	1.15E−4	1.01E−3	7.42E−5	3.68E−5	1.47E−5	1.40E−2	6.97E−3	2.79E−3
	MM2	102	202	502	149	164	181	187	204	231	0.17	0.42	3.56	4.76E−5	1.16E−4	9.90E−4	1.51E−7	2.14E−7	2.59E−7	7.84E−5	4.22E−5	3.09E−5
1E−2	SM1	100	200	500	NA	NA	NA	100	200	500	0.02	0.04	0.27	5.05E−6	1.08E−5	7.58E−5	7.78E−5	4.00E−5	1.64E−5	4.27E−3	2.28E−3	9.70E−4
	SM2	100	200	500	100	200	500	700	1400	3500	0.24	0.54	1.74	6.67E−5	1.49E−4	4.82E−4	7.38E−5	3.80E−5	1.56E−5	4.26E−3	2.28E−3	9.69E−4
	MM1	102	202	502	168	207	248	223	279	344	0.18	0.63	4.97	5.06E−5	1.76E−4	1.38E−3	2.09E−4	1.11E−4	4.67E−5	3.34E−2	1.81E−2	7.66E−3
	MM2	102	202	502	226	242	264	333	293	317	0.25	0.61	4.74	7.06E−5	1.71E−4	1.32E−3	9.55E−7	2.05E−7	3.21E−8	5.23E−4	1.43E−4	2.04E−5
1E−4	SM1	100	200	500	NA	NA	NA	100	200	500	0.02	0.04	0.18	5.01E−6	1.08E−5	4.89E−5	6.39E−5	2.97E−5	8.00E−6	5.87E−3	2.45E−3	5.34E−4
	SM2	100	200	500	100	200	500	700	1400	3500	0.24	0.53	1.64	6.68E−5	1.47E−4	4.55E−4	6.35E−5	2.95E−5	7.90E−6	5.89E−3	2.46E−3	5.38E−4
	MM1	102	202	502	190	249	379	271	367	590	0.22	0.76	8.27	6.08E−5	2.11E−4	2.30E−3	3.95E−4	2.57E−4	1.41E−4	7.31E−2	4.96E−2	2.91E−2
	MM2	102	202	502	423	680	1131	618	957	1254	0.58	2.09	22.95	1.61E−4	5.80E−4	6.37E−3	1.90E−5	2.70E−6	7.33E−6	2.47E−2	1.11E−2	2.47E−3
	MM3a	102	202	502	497	737	1229	1147	1684	2855	4.30	13.07	84.86	1.20E−3	3.63E−3	2.36E−2	1.59E−4	8.64E−5	3.13E−5	3.79E−2	2.09E−2	8.73E−3
	MM3b	102	202	502	1408	2688	5532	3115	5466	10934	12.21	53.96	528.74	3.39E−3	1.50E−2	1.47E−1	5.05E−5	1.86E−5	6.24E−6	2.69E−2	1.24E−2	3.49E−3
	MM3c	102	202	502	1669	2671	4917	3268	5312	10507	13.59	53.63	499.95	3.77E−3	1.49E−2	1.39E−1	4.93E−5	1.85E−5	6.38E−6	2.80E−2	1.31E−2	3.90E−3
	MM3d	102	202	502	1872	3475	7587	4274	7975	16045	17.91	82.21	902.66	4.98E−3	2.28E−2	2.51E−1	2.88E−5	1.93E−5	1.79E−5	2.08E−2	8.32E−3	2.20E−3
	MM3e	102	202	502	596	960	1765	1154	1922	3479	4.34	15.25	96.16	1.21E−3	4.24E−3	2.67E−2	9.21E−5	4.19E−5	8.00E−6	2.97E−2	1.47E−2	4.93E−3
	MM3f	102	202	502	1538	2906	6786	3351	6504	13310	12.92	61.61	718.00	3.59E−3	1.71E−2	1.99E−1	6.18E−5	2.43E−5	4.19E−6	2.72E−2	1.29E−2	4.06E−3
	MM4	102	202	502	2750	3290	4370	36852	44126	58718	24.21	25.74	35.58	6.72E−3	7.15E−3	9.88E−3	3.41E−6	4.95E−6	7.00E−6	2.28E−3	1.31E−3	1.06E−3
	MM5	102	202	502	3445	2993	5900	23870	21524	42053	206.77	293.65	1741.95	5.74E−2	8.16E−2	4.84E−1	1.11E−4	4.67E−5	1.24E−5	1.46E−2	7.22E−3	2.98E−3

Table 3 (*Continued*)

OWE		Np			Nsteps			Nfeval			CPU			R			ARE1 *S*		
ϵ	Method	50	100	250	50	100	250	50	100	250	50	100	250	50	100	250	50	100	250
1E−1	MM1	51	101	251	101	162	306	268	444	930	0.11	0.22	1.08	0.275	0.550	2.700	1.85E−2	1.02E−2	5.87E−3
	MM2	51	101	251	114	181	348	284	500	1100	0.18	0.35	1.66	0.450	0.875	4.150	1.44E−3	2.83E−4	9.34E−5
1E−2	MM1	51	101	251	139	223	367	333	520	1039	0.17	0.29	0.83	0.425	0.725	2.075	3.43E−2	1.81E−2	7.54E−3
	MM2	51	101	251	173	259	441	383	547	1283	0.20	0.32	1.12	0.500	0.800	2.800	5.58E−3	1.54E−3	2.62E−4
1E−3	MM1	51	101	251	152	279	644	347	643	1542	0.18	0.35	1.87	0.450	0.875	4.675	3.75E−2	1.96E−2	8.21E−3
	MM2	51	101	251	195	356	860	409	758	1916	0.21	0.44	2.73	0.525	1.100	6.825	2.20E−2	1.01E−2	1.88E−3
	MM3a	51	101	251	224	415	943	518	944	2187	0.46	1.41	7.87	1.150	3.525	19.68	1.71E−2	7.78E−3	2.70E−3
	MM3b	51	101	251	303	542	1260	699	1328	3050	0.62	2.00	12.08	1.550	5.000	30.20	1.44E−2	6.20E−3	1.95E−3
	MM3c	51	101	251	298	586	1313	715	1452	3096	0.62	2.15	12.37	1.550	5.375	30.93	1.43E−2	6.18E−3	1.95E−3
	MM3d	51	101	251	303	574	1322	695	1299	3003	0.61	1.92	11.90	1.525	4.800	29.75	1.53E−2	6.40E−3	1.96E−3
	MM3e	51	101	251	222	417	919	571	1052	2337	0.40	1.16	6.61	1.000	2.900	16.53	1.51E−2	6.64E−3	2.20E−3
	MM3f	51	101	251	284	531	1184	691	1289	3186	0.60	1.91	12.27	1.500	4.775	30.68	1.46E−2	6.30E−3	2.00E−3
	MM4	51	101	251	7680	7760	7760	61896	62549	62521	15.17	15.59	15.41	37.93	38.98	38.53	1.33E−3	9.43E−4	8.67E−4
	MM5	51	101	251	319	1836	NC	1675	7421	NC	9.41	77.40	NC	23.53	193.5	NC	4.07E−3	3.44E−3	NC

NA: not applicable, NC: no convergence within the specified time, Nsteps: number of integration steps, Nfeval: number of right hand side evaluations, CPU: computation time, and ARE1: averaged relative error according to the L^1-norm. Codes for the different algorithms: 'SMx': Sequencing Method with 'x': '1' Euler, '2' explicit (`ode45`); 'MMx': MatMOL with 'x': '1': low-order implicit (`ode15s`), '2': high-order implicit (`ode15s`), '3y': flux limiter implicit (`ode15s`) with 'y': 'a' Minmod, 'b': Koren, 'c': Smart, 'd': Superbee, 'e': Van Leer, 'f' MC; '4' static regridding implicit (`ode23s`) and '5' dynamic regridding implicit (`ode23s`).

putation time increases compared to the original finite difference schemes, i.e., from a factor 3 for the OWE up to 1 or 2 orders of magnitude for the DFR. This increase is due to the fact that the linear first-order differentiation matrix is now replaced by non-linear functions, and, hence, `Matlab`®'s matrix-vector oriented implementation can no longer be fully exploited. This causes the DFR simulation for the dense grid of $N = 250$ not to converge within the given time. Therefore, supplying the sparsity pattern is extremely important to keep even for the coarse grids the computation times bounded. It should be noted that differences in computation time exist between the different flux limiters. The Van Leer limiter is in general one of the faster flux limiters.

4.5.3. *Adaptive grids*

As an alternative for flux limiting functions, both static and dynamic adaptive grids can be employed. Clearly, only the static regridding code is able to produce an accurate profile in a limited amount of time for the DFR. Nevertheless, the dynamic regridding code has been successfully applied to the OWE (and also the FBBR) case. The accuracy for both methods is approximately one order of magnitude higher than the one for the flux limiting functions. With respect to the computation time the conclusions are less straightforward. For the DFR, the static regridding is much faster than the flux limiters, whereas the inverse is to true for the OWE. It should be noted that the computation time hardly depends on the initial grid size, since this code removes unnecessary grid points during the computation and moves them to places with sharp fronts. On the other hand, the computation time for the dynamic regridding code increases with increasing grid points, resulting only for the $N = 50$ in a faster computation than the static regridding.

4.6. *OS approaches*

Although OS approaches can in general be adapted to also treat cases with non-constant velocities and diffusion coefficient, this modification is not possible in a straightforward way for the sequencing method. Therefore, only the results for the DFR case can be discussed.

4.6.1. *Sequencing method*

For the original sequencing method (**SM2**), a rather accurate approximation of the reference profile is obtained for the entire range of diffusion values. For high values the discontinuity is excellently captured, but for low values, the gradients at the inlet are underestimated. This deviation decreases as the number of grid points is increased, as can be seen from the decreasing relative errors. Typically, these methods exhibit higher errors then the MatMOL based methods for high diffusion values, while they outperform the latter techniques for low diffusion values.

Clearly, the fastest algorithm is obtained when an explicit Euler scheme (**SM1**) is employed, indicating the non-stiffness of the reaction part. The use of an explicit integrator (`ode45`) still yields fast results. However, when more than one reaction is present and their characteristic times are different, implicit integrators will be more appropriate. Not all conditions for commutativity are fulfilled, as reaction and diffusion do not commute. However, changing the order from 'convection–reaction–diffusion' to 'convection–diffusion–reaction', leads in all cases to the same solution. Nevertheless, for lower discretisation grids in the 'convection–diffusion–reaction' sequence, a minor jump is visible across the switching position, because in this case the discretised reaction part exhibits a discontinuity due to the switching from the maximum to the minimum value in the jacket temperature, which the diffusion part cannot smooth as it has already been applied in the current time step.

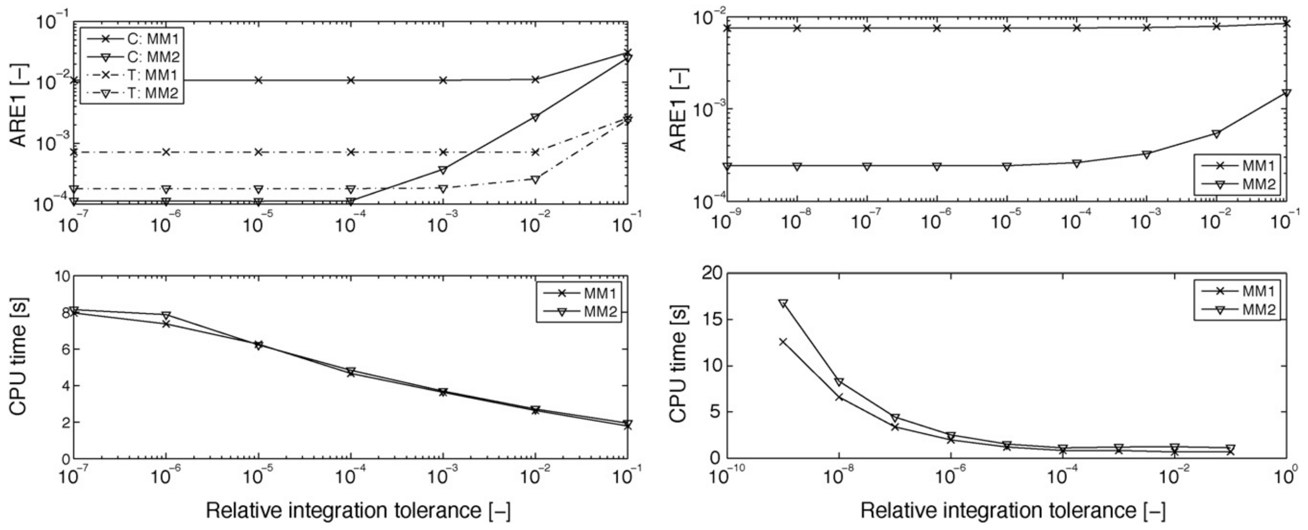

Fig. 5. Time averaged relative errors, and computation time versus the integration tolerance: jacketed tubular reactor with $D = 10^{-3}$ m^2/s (left) and oil well exploitation with $\epsilon = 10^{-3}$ (right).

4.6.2. Sequencing method: adaptation

Removal of the inaccuracies at the inlet by adopting non-uniform grids has been found to be non-trivial. Several attempts to employ a higher number of discretisation points near the inlet have lead to unsatisfactory results. Linearising the reaction part leads to significantly higher computation times. Applying the higher-order scheme for diffusion does not yield more accurate profiles with respect to the reference profiles. Furthermore, it should be mentioned that a low-order approximation of the boundary conditions is required in order to obtain a stable algorithm.

4.7. Guidelines

First of all, it should be mentioned that there is no single best approach for all convection–reaction–diffusion processes, since this choice depends on the nature of the underlying PDEs.

However, an often acceptable strategy is to first try a *MOL approach with finite differences on a fixed grid*. The MatMOL toolbox contains a collection of linear finite difference spatial discretisation schemes which can deal with both constant and non-constant velocities and diffusion coefficients. This spatial discretisation converts the PDEs to semi-discrete DAEs. For the time integration the `Matlab`® time integrators can be employed. Implicit integrators (e.g., ode15s) can directly deal with the resulting DAE systems, while explicit integration routines (e.g., ode45) can be applied after converting the DAEs into ODEs. With respect to the time integrator choice, the implicit integration routine `ode15s`, requires no reformulation as it can directly solve the DAEs and is often faster than when the explicit routine `ode45` is used. Because of the high degree of similarity in the program code, a user can not only easily switch between different discretisations (low-order/high-order, uniform/non-uniform, etc.) but also build new simulation programs without a large recoding effort. Often the only parameter to be tuned is the integration tolerance in order to achieve a balanced discretisation in both space and time.

For highly diffusive processes, the MatMOL toolbox routines (especially, the high-order schemes) outperform the sequencing method both in computation time and accuracy. For convection dominated processes, however, inaccuracies appear: low-order schemes (e.g., the classic *tanks-in-series* approximation) introduce smoothing of steep gradients due to numerical diffusion, while high-order schemes induce non-physical oscillations due to numerical dispersion. Hence, steep moving fronts are not easily captured. More advanced discretisation techniques such as *MOL approaches* based on *finite volumes discretisations with flux limiters* and *a* daptive gridding may then resolve this problem at the expense of a higher computational cost. Hereto, the MatMOL toolbox contains a variety of TVD flux limiting functions and both a static and dynamic regridding scheme.

On the other hand, when the velocity and the diffusion are constants, the easily implemented *sequencing method* provides a valuable alternative for convection dominated cases. This method exhibits a superior performance at low diffusion values since it captures the steep moving fronts excellently, but tends to underestimate the inlet gradient at high diffusion values. The computation time is modest and is hardly influenced by the diffusion values. Therefore, this method can be regarded as an excellent and flexible tool for simple models over a wide range of diffusion values whenever ease of implementation and outstanding transient performance in the presence of sharp changes are required. No parameters have to be tuned, only the sequence in which the subproblems are solved may require some attention. To enhance the computation time linearisation of the reaction part can be an option. However, introducing high-order schemes for the diffusion matrix generally does not improve the solution. Adapting the method to more complex situations, e.g., (i) when the convection and/or diffusion depend on the dependent variables, (ii) when different fluid velocities are involved (e.g., in a reactor with counter current cooling) or (iii) when non-uniform grids are required, is, however, tedious or even impossible.

5. Industrial example

This section illustrates how the techniques that have been tested and compared in the previous section, can successfully be exploited in an industrially relevant case, i.e., the optimisation of a jacketed reverse flow reactor (see also [29]). Due to the periodic switching of the flow direction in this fixed bed tubular reactor, slowly moving temperature fronts are induced which finally become repetitive in a cyclic steady-state regime. The reactor under study can accurately be described by a pseudo-homogeneous model leading to the following set coupled non-linear parabolic partial differential

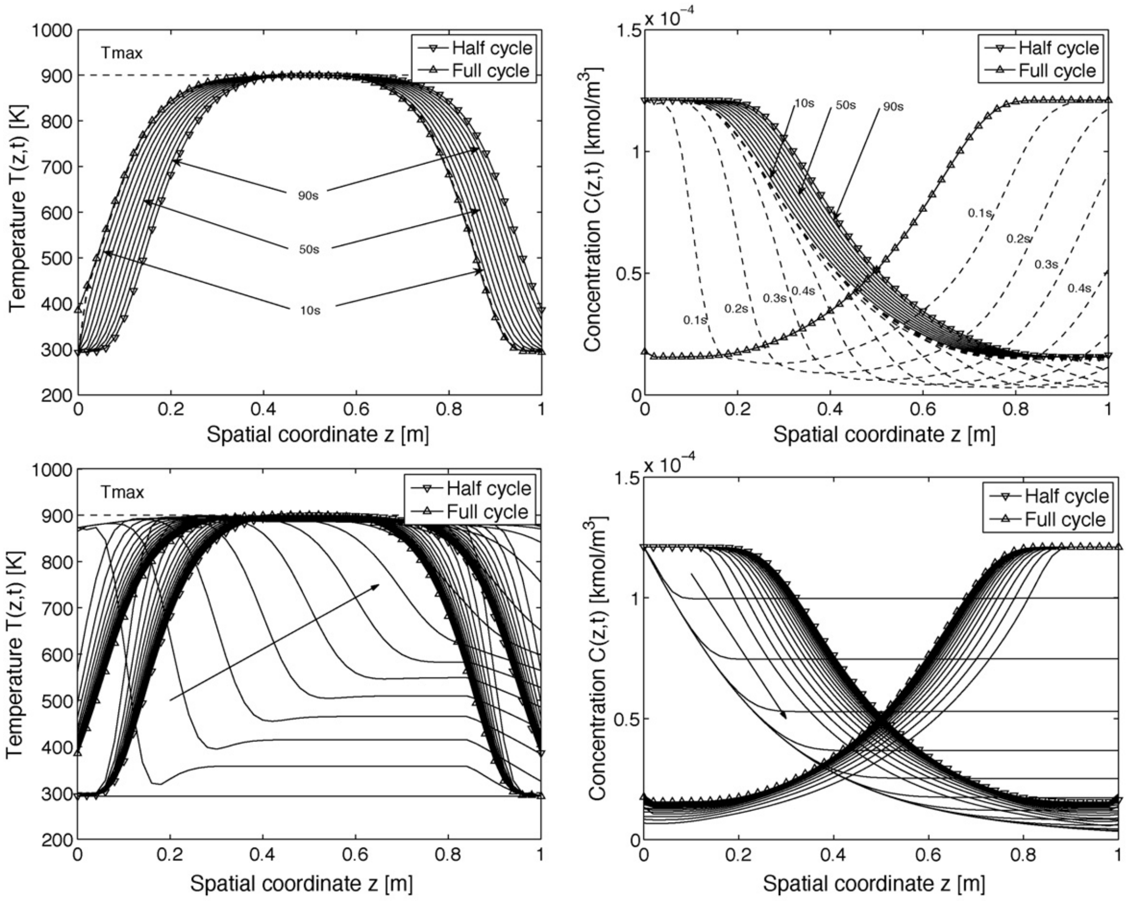

Fig. 6. Temperature (left) and concentration (right) evolution for the optimised reverse flow reactor: during the cyclic steady-state state (forward flow) (top) and during start-up (bottom).

equations:

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial z^2} - v\frac{\partial C}{\partial z} - kCe^{-E/RT} \tag{24}$$

$$\overline{\rho c}_p \frac{\partial T}{\partial t} = \lambda_{\text{ax,eff}}\frac{\partial^2 T}{\partial z^2} - \rho_g c_{pg} v \varepsilon \frac{\partial T}{\partial z} - \Delta H \varepsilon k C e^{-E/RT} - \frac{4h}{d}(T - T_w) \tag{25}$$

and boundary conditions:

$$C(0,t) = C_{\text{in}} \quad \text{and} \quad T(0,t) = T_{\text{in}} \quad \text{for } 2(i-1)\tau \le t < 2i\tau \tag{26}$$

$$C(L,t) = C_{\text{in}} \quad \text{and} \quad T(L,t) = T_{\text{in}} \quad \text{for } 2i+1\tau \le t < 2(i+1)\tau \tag{27}$$

where C [kmol/m^3] is the reactant concentration, C_{in} [kmol/m^3] the incoming reactant concentration, T [K] the temperature, T_{in} the temperature of the incoming gas, τ [s] the switching time, i an integer, D [m^2/s] the dispersion coefficient, v [m/s] the gas velocity (positive for forward flow and negative for backward flow), k [1/s] the rate constant, E/R [K] the activation temperature, $\overline{\rho c}_p$ [kJ/m^{3}K] the fixed bed heat capacity, $\lambda_{\text{ax,eff}}$ [kW/mK] the effective axial heat conductivity, $\rho_g c_{pg}$ [kJ/m^{3}K] the heat capacity of the gas, ε the void fraction, $-\Delta H$ [kJ/kmol] the heat of reaction, L [m] the reactor length, T_w [K] the temperature of the jacket fluid, d [m] the reactor diameter and h [kW/m^{2}K] the (global) heat transfer coefficient. The value of this heat transfer coefficient is constant and positive in the jacketed zone, i.e., $[L/2 - L_j/2, L/2 + L_j/2]$ with L_j [m] the jacket length, and zero in the insulated zones, i.e., $[0, L/2 - L_j/2]$ and $[L/2 + L_j/2, L]$.

The goal is to determine the optimal cyclic steady-state, which involves a trade-off between conversion and energy costs:

$$J = (1-A)\frac{\int_0^{2\tau} C_{\text{outlet}}(t)\,\mathrm{d}t}{2\tau} + \frac{A}{K}\frac{\int_0^{2\tau}\int_0^{L}(4h/(\rho_g c_{pg} T_{\text{inlet}} L d))(T_w - T(z,t))\,\mathrm{d}z\,\mathrm{d}t}{2\tau}$$

where the trade-off value A and the scaling factor K are taken equal to 0.5 and 10^4 m^3/kmol, respectively. The conversion cost measures the time averaged unconverted reactant concentration, while the energy cost accounts for the net heat which can be recuperated from the reactor via the jacket. The degrees of freedom are the jacket temperature T_w ($\in[200, 900]$ K), the switching period τ ($\in[100, 500]$ s) and the jacket length L_j ($\in[0.4, 1]$ m), while an upper limit of 900 K is imposed on the temperature inside the reactor for constructive reasons.

This problem has been successfully solved in `Matlab`® by coupling the MatMOL toolbox to `Matlab`®'s optimisation routine `fmincon`. For simulation, the MatMOL toolbox is employed within a direct dynamic simulation approach, i.e., simulating the reactor start-up for 80 cycles. This number has been found to be high enough to guarantee (based on measures presented in [30]) that a (cyclic) steady-state is reached. Other approaches to compute the cyclic steady-state are, e.g., global discretisation and (multiple) shooting (see, e.g., [31]). Although these techniques are in general more efficient, the size of the problem increases significantly inducing the need for tailored structure exploiting (optimisation) algorithms. However, direct dynamic simulation always yields the entire transient start-up behaviour, which is important for the plant operators.

In the current case a fourth-order discretisation scheme (Eqs. (9) and (10)) is adopted for both the convection and the diffusion terms on a uniform grid of 51 discretisation points. The boundary conditions are implemented using a forcing function with a time constant τ_f equal to 0.001. The resulting DAE is integrated as such using `ode15s` with relative and absolute integration tolerances of 10^{-3} and 10^{-6}, respectively. To enhance the numerical computation of the resulting DAE system, the sparsity pattern of the Jacobian has been supplied. Each time the entire 80 cycles are computed without restarting the `Matlab`® integrators at the different flow reversals. However, due the switching between the right hand sides for forward and backward flow at these flow reversal instants, an increased number of integration steps are needed just before and just after the flow reversals.

After optimisation, a minimum cost $J^* = 6.6392 \cdot 10^{-6}$ kmol/m^3 is found for the optimal values $L_j^* = 0.7$ m, $T_w^* = 843.5$ K, and $\tau^* = 100$ s. Hence, only the middle part is jacketed in order to extract heat, while the outer parts are still insulated. The intermediate value of the jacket temperature is employed to control and to limit the maximum reactor temperature. The short switching time induces steep temperature profiles and avoids the reactor to extinguish.

The two top plots in Fig. 6 display the first half cycle of the optimal cyclic steady-state, i.e., forward flow. During this period the initial profiles, i.e., at the end of the backward flow period (indicated by $\triangle$), move to the right and finally coincide with their symmetric counterparts (indicated by $\triangledown$) at the switching moment. The dashed lines represent the transient evolution during the first second with an interval of 0.1 s while the solid lines cover the entire half cycle with an interval of 10 s. From this figure, it is clear that the concentration front moves much faster than the temperature front. However, the employed discretisation scheme is able to handle them both.

The two lower plots in Fig. 6 depict the start-up of an initially empty reactor. The reactor is first preheated during 1500 s by using a feed stream of 873 K. Afterwards the temperature drops to 293 K and the flow reversals start. The reactor profiles are depicted with an interval of 100 s, i.e., the optimal switching time. As clearly can be observed, the reactor profiles gradually evolve towards the cyclic steady-state indicated by $\triangledown$ and $\triangle$, without violating the upper limit of 900 K. Hence, it can be concluded that it is possible to start-up the reactor in such a way that it safely converges towards the previously optimised cyclic steady-state.

6. Conclusions

This paper has discussed and compared `Matlab`® implementations of two classes of simulation methods for convection–reaction–diffusion processes: (i) method of lines (MOL) approaches and (ii) operator splitting (OS) methods. These two classes have been selected based on their (relatively) easy implementation. Several typical advantages, disadvantages and possible pitfalls (e.g., steep moving fronts) have been demonstrated over a large range of convection–reaction–diffusion processes using a jacketed tubular reactor, a fixed bed bioreactor and an oil well exploitation example. Based on these results practical guidelines have been provided. Moreover, since all codes for the test examples have been made available (www.matmol.org), practitioners can easily compare different approaches when coding their own process simulator. Finally, the successful combination with optimisation routines, has been illustrated by optimising an industrially relevant reverse flow reactor case.

References

[1] I. Queinnec, J.C. Ochoa, A. Vande Wouwer, E. Paul, Development and calibration of a nitrification PDE model based on experimental data issued from biofilter treating drinking water, Biotechnol. Bioeng. 94 (2006) 209–222.
[2] R. David, J.-L. Vasel, A. Vande Wouwer, Settler dynamic modeling and matlab simulation of the activated sludge process, Chem. Eng. J. 146 (2009) 174–183.
[3] P. Marín, S. Ordó nez, F. Díez, Procedures for heat recovery in the catalytic combustion of lean methane–air mixtures in a reverse flow reactor, Chem. Eng. J. 147 (2009) 356–365.
[4] W. Hundsdorfer, J. Verwer, Numerical Solution of Time-dependent Advection–Diffusion–Reaction Equations, vol. 33 of Springer Series in Computational Mathematics, Springer, Berlin, 2003.
[5] W. Schiesser, The Numerical Method of Lines, Academic Press, 1991.
[6] L. Shampine, M. Reichelt, The Matlab ODE suite, SIAM J. Sci. Comput. 18 (1997) 1–22.
[7] S. Renou, M. Perrier, D. Dochain, S. Gendron, Solution of the convection–dispersion–reaction equation by a sequencing method, Comput. Chem. Eng. 27 (2003) 615–629.
[8] A. Vande Wouwer, P. Saucez, W. Schiesser, Simulation of distributed parameter systems using a matlab-based method of lines toolbox: chemical engineering applications, Ind. Eng. Chem. Res. 43 (2004) 3469–3477.
[9] K. Alhumaizi, R. Henda, M. Soliman, Numerical analysis of a reaction–diffusion–convection system, Comput. Chem. Eng. 27 (2003) 579–594.
[10] K. Alhumaizi, Comparison of finite difference methods for the numerical simulation of reacting flow, Comput. Chem. Eng. 28 (2004) 1759–1769.
[11] M. do Carmo Coimbra, C. Sereno, A. Rodrigues, Applications of a moving finite element method, Chem. Eng. J. 84 (2001) 23–29.
[12] A. Vande Wouwer, P. Saucez, W. Schiesser, Adaptive Method of Lines, Chapman & Hall/CRC, 2001.
[13] A. Vande Wouwer, P. Saucez, W. Schiesser, S. Thompson, A MATLAB implementation of upwind finite differences and adaptive grids in the method of lines, J. Comput. Appl. Math. 183 (2005) 245–258.
[14] B. Fornberg, Calculation of weights in finite difference formulas, SIAM Rev. 40 (1998) 685–691.
[15] P. Sweby, High-resolution schemes using flux limiters for hyperbolic conservation laws, SIAM J. Numer. Anal. 21 (1984) 995–1011.
[16] A.R. Mitchell, D.F. Griffiths, The Finite Difference Method in Partial Differential Equations, John Wiley and Sons, 1980.
[17] L. Shampine, M. Reichelt, J. Kierzenka, Solving index-1 DAEs in MATLAB and simulink, SIAM Rev. 41 (1999) 538–552.
[18] J. Verwer, J. Blom, R. Furzeland, P. Zegeling, Adaptive Methods for Partial Differential Equations, SIAM. Ch. A Moving-Grid Method for One-dimensional PDEs based on the Method of Lines, 1989, pp. 160–175.
[19] J. Blom, P. Zegeling, Algorithm 731, a moving-grid interface for systems of one-dimensional time-dependent partial differential equations, ACM Trans. Math. Softw. 20 (2) (1994) 194–214.
[20] M. Simpson, K. Landman, T. Clement, Assessment of a non-traditional operator split algorithm for simulation of reactive transport, Math. Comput. Simul. 70 (2005) 44–60.
[21] D. Lanser, J.G. Verwer, Analysis of operator splitting for advection–diffusion–reaction problems from air pollution modelling, J. Comput. Appl. Math. 111 (1999) 201–216.
[22] F. Logist, I. Smets, J. Van Impe, Optimal control of dispersive tubular chemical reactors: Part I, in: Proceedings of the 16th IFAC World Congress, 2005, pp. DVD–ROM, 6 pp.
[23] P. Danckwerts, Continuous flow systems, Chem. Eng. Sci. 2 (1953) 1–13.
[24] I. Smets, D. Dochain, J. Van Impe, Optimal temperature control of a steady-state exothermic plug flow reactor, AIChE J. 48 (2002) 279–286.
[25] F. Logist, I.Y. Smets, J.F. Van Impe, Derivation of generic optimal reference temperature profiles for steady-state exothermic jacketed tubular reactors, J. Process Contr. 18 (2008) 92–104.
[26] F. Logist, P.M. Van Erdeghem, I.Y. Smets, J.F. Van Impe, Optimal design of dispersive tubular reactors at steady-state using optimal control theory, J. Process Contr. 19 (2009) 1191–1198.
[27] M. Laabissi, J.J. Winkin, D. Dochain, M.E. Achhab, Dynamical analysis of a tubular biochemical reactor infinite-dimensional nonlinear model, in: Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference, 2005, pp. 5965–5970.
[28] S. Buckley, M. Leverett, Mechanism of fluid displacements in sands, Trans. AIME 146 (1946) 107–116.
[29] F. Logist, A. Vande Wouwer, I. Smets, J. Van Impe, Optimal temperature profiles for tubular reactors implemented through a flow reversal strategy, Chem. Eng. Sci. 62 (2007) 4675–4688.
[30] K. Gosiewski, Effective approach to cyclic steady-state in the catalytic reverse-flow combustion of methane, Chem. Eng. Sci. 59 (2004) 4095–4101.
[31] J. Unger, M. Kolios, G. Eigenberger, On the efficient simulation and analysis of regenerative processes in cyclic operation, Comput. Chem. Eng. 54 (1997) 2597–2607.