A toolbox of equation-free functions in MATLAB/Octave for efficient system level simulation

John Maclean1 · J. E. Bunder1 · A. J. Roberts1

Received: 15 December 2019 / Accepted: 7 October 2020 / Published online: 30 October 2020
© Springer Science+Business Media, LLC, part of Springer Nature 2020

Abstract
The ‘equation-free toolbox’ empowers the computer-assisted analysis of complex, multiscale systems. Its aim is to enable scientists and engineers to immediately use microscopic simulators to perform macro-scale system level tasks and analysis, because micro-scale simulations are often the best available description of a system. The methodology bypasses the derivation of macroscopic evolution equations by computing the micro-scale simulator only over short bursts in time on small patches in space, with bursts and patches well-separated in time and space respectively. We introduce the suite of coded equation-free functions in an accessible way, link to more detailed descriptions, discuss their mathematical support, and introduce a novel and efficient algorithm for Projective Integration. Some facets of toolbox development of equation-free functions are then detailed. Download the toolbox functions and use to empower efficient and accurate simulation in a wide range of science and engineering problems.

Keywords Multiscale methods · Code toolbox · Numerical algorithms

1 Introduction

Suppose that you have a detailed and trustworthy computational1 simulation of some problem of interest. When the detailed computation is too expensive to simulate all
the times of interest over all the space of interest, then the ‘Equation-Free Methodologies’ aim to accurately empower long-time simulation and system level analysis (e.g. [1–3]). Our toolbox provides these methodologies coded into MATLAB/Octave functions.

1.1 Simulation on only small patches of space

Figure 1 illustrates an ‘equation-free’ computation on only small well-separated patches of the spatial domain. The micro-scale simulations within each patch, here the nonlinear diffusive system (1), are craftily coupled to neighbouring patches and thus interact to provide accurate macro-scale predictions over the whole spatial domain (e.g. [4]). We have proved that the patches may be tiny, and still the scheme makes accurate macro-scale predictions [5]. Thus the computational savings may be enormous, especially when combined with projective integration (Section 1.2).

The example system illustrated in Fig. 1 is a nonlinear discrete diffusion system inspired by the lubrication flow of a thin layer of fluid, namely

\[
\frac{\partial u}{\partial t} = \nabla \cdot (3u^2 \nabla u), \quad \text{equivalently} \quad \frac{\partial u}{\partial t} = \nabla^2 (u^3),
\]

Fig. 1 Snapshots in time of a patch scheme simulation. Only 20% of the physical domain is simulated: each of the small opaque squares (‘patches’) solves a computationally expensive PDE internally, and all patches are coupled together to enable an efficient simulation over the entire domain. Simulation details are in Section 1.1

Springer
which on a 2D micro-scale lattice \( x_{i,j} \) with tiny micro-scale spacing \( d \) is here discretised simply to

\[
\frac{d u_{i,j}}{dt} = u_{i+1,j}^3 + u_{i-1,j}^3 + u_{i,j+1}^3 + u_{i,j-1}^3 - 4u_{i,j}^3 \frac{d^2}{d^2} .
\]

We want to predict the dynamics of this spatial micro-scale lattice system on the macro-scale spatial domain \([-2, 2] \times [-3, 3]\), but suppose full direct computation is too expensive. Instead, the micro-scale simulation illustrated by Fig. 1 was performed only on about 20% of the domain (it could be much less)—the small patches of space in Fig. 1. The key to an accurate macro-scale prediction is that each patch is coupled to nearby patches, at every computed time, by appropriate macro-scale interpolation that gives the edge values for every patch (e.g. [4]).

The patch scheme is most useful in applications where there is no known macro-scale closure. Then, the patch scheme automatically achieves a computational macro-scale closure, without the need for any analytic construction often invoked in numerical/computational homogenization (e.g. [6–8])—the patch scheme is ‘equation-free’. Our approach could be classed as a dynamic homogenization (e.g. [9]). Frequently, problems of interest in applications compute on a micro-scale spatial lattice as in the spatial discretisation (1). Suppose \( x_i \) are coordinates of a micro-scale lattice, for potentially exhaustingly many lattice points indexed by \( i \); for example, a full atmospheric simulation. And suppose your detailed and trustworthy simulation is coded in terms of micro-field variable values \( u_i(t) \in \mathbb{R}^p \) at lattice point \( x_i \) at time \( t \). When a detailed computational simulation is prohibitively expensive over all the desired spatial domain, \( x \in \mathbb{X} \subset \mathbb{R}^d \), our toolbox provides functions that empower you to use your micro-scale code as a ‘black-box’ inside only small, well-separated, patches of space by appropriately coupling across un-simulated space between the patches (Section 2.2). The toolbox functions have many options including both newly developed spectral coupling, and new support for ensembles of microscale heterogeneity, new symmetry preserving coupling, and in 1D, 2D or 3D space. Section 3.2 gives an introductory tutorial.

1.2 Projective integration skips through time

Simulation over time is a complementary dynamic problem. The ‘equation-free’ approach is to simulate for only short bursts of time, and then to extrapolate over un-simulated time into the future, or into the past, or perform system level analysis (e.g. [10–13]). Figure 2 plots one example where the gaps in time show the un-computed times between bursts of computation.

In general such a simulation is coded in terms of detailed (micro-scale) variable values \( u(t) \), in \( \mathbb{R}^p \) for some \( p \) (typically large in applications), and evolving in time \( t \). The details \( u \) could represent particles, agents, or states of a system. Both forward and backward in time computations may be performed by Projective Integration (PI) with provable accuracy [10, 13–17]. For efficient simulation on long times, Section 2.1 describes how to provide your micro-scale detailed MATLAB/Octave code as a ‘black box’ to the novel Projective Integration functions in the toolbox. Section 3.1 gives a user-friendly introductory tutorial, and provides further pointers to the thorough literature on parameter selection for PI.
Fig. 2 Projective Integration by the new function PIG of an example, extremely stiff, multiscale system. The macro-scale solution $U(t)$ is represented by the blue circles (◦), and times at which PIG ran short bursts of computation (each lasting $2 \times 10^{-4}$ time units) are shown with dots. The black dotted line, underneath the PI solution, shows an accurate micro-scale simulation over the whole time domain; this micro-scale simulation was two orders of magnitude more expensive than PIG. Simulation details are given in Section 1.2.

The example simulation of Fig. 2 is that of a toy system that nonetheless has challenging qualities of the multiscale phenomena that Projective Integration resolves. Here the micro-scale simulation is a pair of coupled slow-fast ODES for $\mathbf{u}(t) = (u_1(t), u_2(t))$:

\[
\begin{align*}
\frac{du_1}{dt} &= \cos(u_1) \sin(u_2) \cos(t), \\
\frac{du_2}{dt} &= 10^5 [\cos(u_1) - u_2].
\end{align*}
\]

(2a) (2b)

Using simple integration schemes, numerical solutions can be rapidly computed on micro-times of $O(10^{-5})$, but solutions over $O(1)$ times are computationally prohibitive—except by stiff integrators. Section 3.1.4 discusses scenarios where stiff integrators cannot be used or are relatively expensive, but where projective integration is effective (Fig. 4 shows the degradation in performance by the MATLAB stiff integrator ode15s()).

The system (2a) represents the realities of, for example, molecular dynamics simulations with rapid modes represented by $u_2(t)$ and slow macro-scale state variables (like temperature) represented by $u_1(t)$. In applications the dynamics of the slow
modes are usually not known, instead they emerge over the micro-scale simulation bursts (e.g. [12, 18, 19]). Consequently, although in this toy system the slow variable $u_1$ and fast variable $u_2$ are obvious, here we compute only with the full system (2a)—it is a ‘black-box’ for which we do not necessarily know what ‘variables’ are fast or slow. An alternative is to invoke algebraic analysis to construct the slow manifold of slow-fast systems like (2a) (e.g. [20], Ch. 4–5): however, in many scenarios such analysis is not feasible. Since one often only measures macro-scale state variables, here we suppose the micro-simulator only outputs $U(t) = u_1(t)$, called a restriction. Projective Integration (PI) uses only short bursts of the ‘black-box’ simulation, and then invokes an appropriate extrapolation to accurately predict large ‘projective’ steps. After a projective step the PI algorithm appropriately re-initialises $u_2(t)$, called lifting, for another burst of the micro-simulator.

Figure 2 shows the PI computation and the associated micro-scale bursts, compared to expensive simulation with $\texttt{ode45()}$. The micro-scale simulations with $u_2(t)$ show the characteristic fast transients. The absolute error between the Projective Integration simulation and the trusted $\texttt{ode45()}$ simulation is $2 \cdot 10^{-6}$.

The PI simulation of Fig. 2 was performed by a novel function, called $\texttt{PIG}$, introduced in Section 2.1 and included in the toolbox. The advantage of the Projective Integration simulation over MATLAB’s $\texttt{ode45()}$ is that PI uses only 0.6% of the number of evaluations of the ODES (2a). Section 3.1 discusses how this new PI scheme and function may be constructed through recursive use of $\texttt{ode45()}$ which inherits all of MATLAB’s adaptive error control—an error control active on both the micro-scale and the macro-scale.

### 1.3 Cognate multiscale methods

Equation-Free methods are far from the only multiscale algorithms. Though comparison is not a focus of the toolbox, we briefly introduce some key methods. In particular the molecular dynamics of [21], the development of kinetic equations for gas dynamics in [22], and the FLAVOR algorithm of [23] and the extensive Heterogeneous Multiscale Methods (e.g. [24, 25]). These last algorithms can only be used, but may be optimal, if the multiscale system can be explicitly separated into known slow and fast dynamics.

### 2 Equation-free algorithms

This section outlines the key Projective Integration and Patch Dynamics algorithms implemented in the toolbox [26]. Pseudo-code highlights the essential features of each algorithm and the accompanying discussion demonstrates the extended capabilities. Theoretical support for the algorithms is also discussed.

#### 2.1 Algorithms for projective integration in time

Our efficient simulation of the stiff dissipative system (2a) is due to the class of algorithms called Projective Integrators. Small time steps with the micro-simulator are
alternated with long time steps comprised of ‘projective’ extrapolations. Sometimes PI is done with the macrostep being a forward Euler method (e.g. [27, 28]), and so incurs global error proportional to the size of the projective time steps. Such low accuracy motivated the development of PI algorithms of provably higher accuracy [11, 29]. Here we additionally extend the methodology to a projective integration that invokes adaptive integrators.

The toolbox provides $\text{PIG}()$—denoting a Projective Integration with General macro-integrator and micro-simulator. The user-specified macro-integrator is any integrator suitable for time stepping on the macro-scale, and includes adaptive integrators such as MATLAB’s $\text{ode45}$ as used for Fig. 2. Then, $\text{PIG}()$ provides to that integrator accurate time derivatives at the required times—time derivatives estimated from appropriate relatively short bursts of the user-defined micro-simulator $\text{microSim}()$. The effect is to do PI with any integrator, explicit or implicit, taking the projective steps. Error analyses suggested that schemes of this sort usually incur significant errors proportional to the duration of the micro-simulator burst [14, 30, 31]. However, such errors are avoided by $\text{PIG}()$ through a novel implementation of a ‘constraint-defined manifold computing’ scheme based on a methodology originally proposed by [32, 33]. Algorithm 1 provides pseudo-code for $\text{PIG}()$ that details these steps. This package is the first time such functionality for Projective Integration has been developed into a general function, tested, and made available.

**Algorithm 1** The $\text{PIG}()$ function uses short bursts of a user provided micro-scale process to simulate from time $T_0$ to $T$, with initial condition $u_0$, and via a user-specified macro-scale integrator $\text{macroInt}()$.

$\text{PIG}$ tells the specified macro-scale integrator to use time derivatives estimated by a constraint-defined manifold function.

1: \textbf{function} $\text{PIG}()$($\text{macroInt}()$, $\text{microSim}()$, $\text{constrDeriv}()$, ($T_0$, $T$), $u_0$)  
2: \hspace{1em} \text{return} \{T_n, u_n\}_{n=0}^{N} := \text{macroInt}()\{(\text{constrDeriv}()), (T_0, T), u_0\}$;  
3: \textbf{end function}

Here, the toolbox’s constraint-defined manifold function uses two applications of $\text{microSim}$ and a backwards projective step in order to provide the time derivative at the time specified by $\text{macroInt}()$.

1: \textbf{function} $\text{CONSTRDeriv}(t_0, u_0, \delta)$  
2: \hspace{1em} \{t_m, u_m\}_{m=0}^{M} := \text{MICROSIM}(t_0, u_0, \delta); \quad \text{(burst of simulation)}  
3: \hspace{1em} \left(\frac{\Delta u}{\Delta t}\right)_M := \frac{u_M - u_{M-1}}{t_M - t_{M-1}}; \quad \text{(derivative estimate at } t_0 + \delta)  
4: \hspace{1em} v_0 := u_m - 2\delta \left(\frac{\Delta u}{\Delta t}\right)_M \quad \text{(PI backwards to } t_0 - \delta)  
5: \hspace{1em} \{s_m, v_m\}_{m=0}^{M} := \text{MICROSIM}(t_0 - \delta, v_0, \delta); \quad \text{(burst of simulation)}  
6: \hspace{1em} \text{return} \left(\frac{\Delta v}{\Delta s}\right)_M := \frac{v_M - v_{M-1}}{s_M - s_{M-1}}; \quad \text{(derivative estimate at } t_0)$  
7: \textbf{end function}

A user provides a micro-scale simulator with the following input and output.

1: \textbf{function} $\text{MICROSIM}(t_0, u_0, \delta)$  
2: \hspace{1em} compute a burst with end time $t_M := t_0 + \delta$;  
3: \hspace{1em} \text{return} \{t_m, u_m\}_{m=0}^{M}$  
4: \textbf{end function}
Algorithm 1 outlines the essence of PIG(), but additional features may be invoked by a user. In particular, sometimes the projective steps are performed on a few macro-scale variables only. That is, the PI is done in a space of reduced dimension to that of the micro-simulator (e.g. [34–36]). In such cases the user provides ‘lifting’ and ‘restriction’ operators to convert between the micro- and macro-simulation spaces [37].

In addition to PIG(), the toolbox provides efficient integrators PIRK2() and PIRK4(), which are Projective Integrators similar to PIG() but with the user-defined macro-integrator replaced with second and fourth order, respectively, Runge–Kutta macro-integrators that take user-specified macro-scale time-steps.

2.2 Algorithms to simulate on patches of space

The spatial multiscale ODES (1) are simulated only on a fraction of space by employing a patch scheme (also known as the gap-tooth scheme [38]). This scheme applies to dynamic systems evolving in time with some ‘spatial’ structure (‘space’ could be some other type of domain), and is useful when direct simulation over the spatial domain of interest is infeasible due to the overwhelming computational cost. The toolbox assumes that space on the micro-scale is represented as a rectangular lattice network with grid points $x_i$—one example being the spatial discretisation (1) of a PDE. The toolbox function requires the user to provide a function that computes a micro-scale time-step or time derivative for variables $u_i$ at each of the lattice nodes.

The patch scheme computes the detailed micro-scale at only a (small) subset of lattice nodes in space—the patches as illustrated by Fig. 1. The scheme craftily couples the patches together to ensure macro-scale predictions are provably accurate [4, 5, 39, 40]. For simplicity, this article discusses specifically the case of patches in 1D space, but the scheme and the toolbox extend to 2D and 3D space [4], as in the 2D simulations of Fig. 1. The essence of the toolbox patch algorithms are outlined, for a specific case in 1D space, by Algorithms 2 to 4.

Algorithm 2 Patch scheme when a user has a function $\text{fun}(t, u)$ that computes a time-step/time-derivative on a micro-scale lattice in space. Suppose that direct simulation is prohibitively expensive with $\text{fun}$ by some time integrator, say $\text{macroInt}(\text{fun}, \{T_0, T\}, u_0)$. A user then invokes the patch scheme with the following two steps.

1: Invoke $\text{configPatches1}($fun, $\{a, b\}, \ldots$) to configure the patch scheme on the spatial domain $[a, b]$ of interest (Algorithm 3).
2: Then executing $\text{macroInt}(\text{patchSmooth1}, \{T_0, T\}, u_0)$ computes the solution on patches in space (Algorithm 4).

Algorithm 2 introduces the necessary code for using the toolbox to implement the patch scheme on a problem of interest. In the specific case of a macro-scale 1D domain $[a, b]$, the toolbox function $\text{configPatches1}$ (Algorithm 3) constructs
nPatch equi-spaced patches, each of width proportional to ratio and each containing nSubP micro-scale lattice points. The integer ordCC determines the form and order of the inter-patch coupling that ensures macro-scale accuracy:

Algorithm 3 configPatches1 initialises the basic patch structures for Algorithm 2. It creates nPatch equi-spaced patches inside the 1D interval [a, b]. The patches are of size proportional to ratio with each having nSubP micro-scale lattice points, and coupled by interpolation of order ordCC. All defined variables are saved as global for subsequent use in simulation.

1: function configPatches1(fun, {a,b}, nPatch, ratio, ordCC, nSubP)
2:  H := (b - a)/N; (distance between macro-scale mid-patches)
3:  Xj := a + (j - 1/2)H for j := 1 : nPatch; (macro-scale mid-patches)
4:  i0 := (nSubP + 1)/2; (index for the centre point of a patch)
5:  d := (ratio × H)/(i0 - 1); (micro-scale lattice spacing)
6:  xi,j := Xj + d(i - i0) for j := 1 : nPatch and i := 1 : nSubP; (micro-scale lattice points in the jth patch)
7:  Calculate coefficients of interpolation to determine the edge-values of a patch from mid-patch values, for interpolation from ordCC/2 patches to either side of a patch.
8: end function

Algorithm 4 Evolve in time on patches via function patchSmooth1 which interfaces between a user’s macroInt and fun. Function patchSmooth1 calculates the time-derivative/ time-step in the interior of all patches via function patchEdgeInt1 which couples the patches with a macro-scale interpolation.

1: function patchSmooth1(t, u)
2:  u := patchEdgeInt1(u);
3:  dudt_interior := fun(t, u, x); (evaluate time-step inside every patch)
4:  dudt_edges,j := 0 for j := 1 : nPatch;
5:  dudt := RESHAPE(dudt, nSubP × nPatch, 1); (make column vector)
6:  return dudt
7: end function

1: function patchEdgeInt1(u)
2:  u := RESHAPE(u, nSubP, nPatch); (reshape so each column is a patch)
3:  u_edges,j := Lagrange interpolation of mid-patch values u_i0,j from ordCC/2 patches to either side of each patch.
4:  return u
5: end function

– even ordCC, the usual, and as described in Algorithm 3, invokes classic Lagrange interpolation, of order ordCC, from mid-patch values to the patch-edge values to generally give a macro-scale that is consistent to errors $O(H^{ordCC})$ [5];

– the special case of ordCC = 0, as used in 2D for the example of Fig. 1, invokes a new spectral interpolation that recent numerical experiments indicate has consistency errors exponentially small in H;
odd \( \text{ordCC} \) creates a scheme with a staggered grid of patches suitable for many wave systems [41], staggered in the sense that mid-patch values for odd/even patches use order \( \text{ordCC} \) interpolation to determine edge-patch values of even/odd patches, respectively, and that typically has macro-scales consistent to errors \( \mathcal{O}(H^{\text{ordCC}+1}) \) [40];

and the special case of \( \text{ordCC} = -1 \) invokes a new staggered spectral interpolation that recent numerical experiments indicate has consistency errors exponentially small in \( H \).

After constructing the patches, a user-specified integrator \( \text{macroInt} \) (step 2 of Algorithm 2) simulates the user-defined \( \text{fun} \) over the time interval \( [T_0, T] \) with initial condition \( \vec{u}_0 \). Algorithm 4 overviews the toolbox function \( \text{patchSmooth1} \) which interfaces between the patch coupling and the user’s function \( \text{fun} \) that computes the micro-scale time-derivative/time-step within all the patches.

The toolbox currently implements corresponding functionality for problems in 2D space, as Fig. 1 shows, via toolbox functions \( \text{configPatches2, patchSmooth2, and patchEdgeInt2} \), and similarly for 3D space.

This patch scheme is an example of so-called computational homogenization (e.g. [6, 7, 42]) and is related to numerical homogenization (e.g. [8, 9, 43, 44]). The three main distinguishing features of the patch scheme are that a user need not perform any analysis of the micro-scale structures; that computations are done only on a (small) subset of the spatial domain; and for a wide class of systems the scheme is proved to be accurate to a user specified order [4, 5, 40]. However, in application to micro-scale heterogeneous media—the main interest of computational/numerical homogenization—more research needs to be done. [39] started exploring the patch scheme in heterogeneous media and established it generally has small macro-scale errors for diffusion in random media. Further, they found, analogous to that found in some numerical homogenization [8], that when the patch half-width is an integral number of periods of the micro-scale heterogeneity; then, the macro-scale predictions are accurate to errors \( \mathcal{O}(H^{\text{ordCC}}) \), as before.

A recent innovation in the toolbox (by setting parameter \( \text{patches.EdgyInt}=1 \)) is the capability to couple patches by interpolating next-to-edge values to the edge-patch values, but to the opposite edge. This coupling is the subject of an article currently in preparation which will discuss how the coupling usefully preserves symmetry in many applications of interest, and is of controllable macro-scale accuracy for micro-scale heterogeneous media.

### 3 Using toolbox functions

Users need to download the toolbox via GitHub\(^2\). Place the folder of this toolbox in a path searched by your MATLAB/Octave. The toolbox provides both a user’s and a developer’s manual: start by looking at the User’s Manual.

\(^2\)https://github.com/ua1184615/EquationFreeGit
Many of the main toolbox functions, when invoked without any arguments, will simulate an example. Executing the command `PIG()` reproduces the Projective Integration example presented in Section 1.2. Similarly, the non-linear diffusion/lubrication-like example of Section 1.1 is reproduced by executing `configPatches2()` . The following Sections 3.1 and 3.2 explain the code for both of these introductory examples as templates to adapt for other problems.

### 3.1 Invoking Projective Integration in general

This subsection discusses some key factors when constructing a Projective Integration (PI) simulation.

#### 3.1.1 Parameter selection for Projective Integration

Intuitively the burst must be long enough, and macro-time-steps short enough, that the fast variables remain controlled over simulation. We now provide explicit bounds guaranteeing good performance, assuming that the fast behaviour is linear. Centre Manifold Theory guarantees that this linear assumption applies to a large class of nonlinear systems, for which the dynamics is accurately described by a linearized system.

We analyse the situation in which the slow dynamics occurs at rate/frequency of magnitude about \( \alpha \); and the rate of decay of the fast modes are higher than the lower bound \( \beta \) (e.g. if three fast modes decay roughly like \( e^{-12t} \), \( e^{-34t} \), \( e^{-56t} \), then \( \beta \approx 12 \)). The PI must be able to stably project the (damped) fast modes, and so the duration \( \delta \) of the micro-scale burst must be sufficiently long. The fast modes decay like \( e^{-\beta \delta} \) over the micro-burst, and then grow like \( \beta \Delta \) on a projective step of length \( \Delta \). For stability, the product of these effects must be less than one; that is, \( \beta \Delta e^{-\beta \delta} \lesssim 1 \). Rearranging requires the burst length

\[
\delta \gtrsim \frac{1}{\beta} \log |\beta \Delta|
\]  

Figure 3 plots this lower bound as a function of \( \beta \Delta \) by writing \( \delta/\Delta \frac{1}{\beta \Delta} \log |\beta \Delta| \).

Once stability of the fast modes is assured by satisfying (3), the accuracy of the macro-scale time-step must be considered. The second and fourth order PI functions `PIRK2()` and `PIRK4()` have global errors proportional to \( \Delta^2 \) and \( \Delta^4 \) respectively, for a projective step of length \( \Delta \). More specifically, for a desired accuracy of \( \varepsilon \) and recalling that the slow dynamics occurs at rate/frequency of magnitude about \( \alpha \): standard analysis of the Runge-Kutta scheme requires \( \Delta \) such that \( \alpha \Delta \lesssim (6\varepsilon)^{1/2} \) for `PIRK2`, and \( \alpha \Delta \lesssim \varepsilon^{1/4} \) for `PIRK4`. The accuracy of `PIG()` is controlled by adjusting whatever the accuracy parameters are provided to the user-specified macro-integrator.

The topic of parameter selection for Projective Integration has received some particular attention in recent years. Diverse systems have been considered, and explicit stability and error scaling established, in [14–17] (the latter for a related algorithm for stochastic differential equations).
Fig. 3 Minimum value (3) for $\delta/\Delta$, the ratio of burst length for the micro-simulator to the macro-time-step, in order to achieve stability in Projective Integration. The horizontal axis $\beta\Delta$ is the product of the minimum decay rate for the fast modes and the macro-time-step

3.1.2 PIG tutorial

We now discuss details of the simulation of the multiscale, slow-fast, ODES (2a) shown in Fig. 2. First we code the right-hand side function of the system (often people would phrase it in terms of the small parameter $\epsilon = 1/\beta$),

```matlab
beta = 1e5;
dxdt=@(t,x) [ cos(x(1)) * sin(x(2)) * cos(t
beta*( cos(x(1))-x(2) ) ];
```

Second, we code micro-scale bursts, here using the standard `ode45()`.

```matlab
bT = 2/beta * log(beta);
microBurst = @(tb0, xb0) feval('ode45',dxdt,[tb0 tb0+bT],xb0);
```

Then define the micro-scale burst from state $xb0$ at time $tb0$ to be an integration by the adaptive `ode45` of the coded ODES (2a), over a burst-time $bT$.

```matlab
microBurst(t, x0)
```

Third, code functions to convert between macro-scale state $u_1$ and micro-scale state $(u_1, u_2)$.

```matlab
restrict = @(x) x(1);
```
lift = @(X,xApprox) [X; xApprox(2)];

The (optional) restrict() and lift() functions are detailed in the following Section 3.1.3. Fourth, invoke PIG to use MATLAB’s ode45 on the macro-scale slow evolution. Integrate the micro-bursts over $0 \leq t \leq 6$ from the initial condition $x(0) = (1, 0)$.

$Tspan = [0 \ 6]$;
$x0 = [1;0]$;
$macroInt='ode45';$
$[Ts,Xs,tms,xms] = PIG(macroInt,microBurst,Tspan,x0,restrict, lift);

We pause this example to discuss the available outputs from PIG(). Between zero and five outputs may be requested from PIG(). Most often you would store the first two output results, via say $[Ts,Xs] = PIG(...)$.

- $T$, an $L$-vector of times at which $macroInt()$ produced results.
- $X$, an $L \times N$ array of the computed solution: the $i$th row of $X, X(i,:)$, is to be the macro-state vector $X(t_i)$ at time $t_i = T(i)$.

However, micro-scale details of the underlying Projective Integration computations may be helpful, and so PIG() provides some optional outputs of the micro-scale bursts, via $[Ts,Xs,tms,xms] = PIG(...)$

- $tms$, optional, is an $\ell$-dimensional column vector containing micro-scale times with bursts, each burst separated by NaN;
- $xms$, optional, is an $\ell \times n$ array of the corresponding micro-scale states.

In some contexts it may be helpful to see directly how Projective Integration approximates a reduced slow vector field, via $[T,X,tms,xms,svf] = PIG(...)$ in which

- $svf$ is a struct containing the Projective Integration estimates of the slow vector field.
  - $svf.T$ is a $\hat{L}$-dimensional column vector containing all times at which the micro-scale simulation data is extrapolated to form an estimate of $dx/dt$ in $macroInt()$.
  - $svf.dx$ is a $\hat{L} \times N$ array containing the estimated slow vector field.

If $macroInt()$ is, for example, the forward Euler method (or the Runge–Kutta method), then $\hat{L} = L$ (or $\hat{L} = 4L$).

Returning to the example one remarkable feature of PIG() is revealed: this PI simulation, which as mentioned in Section 1.2 uses only 0.6% as many evaluations of (2a) as direct simulation with ode45(), is accomplished with a recursive call to ode45(). The standard MATLAB integrator is used to both compute the micro-scale bursts, and also compute the projective steps. All the usual machinery of adaptive time stepping and error control is used to regulate the micro-scale simulation, and also to regulate the macro-scale projective time-steps.
3.1.3 Optional PI inputs enable the user to choose how to convert between macro- and micro-scale states

As described in the penultimate paragraph of Section 2.1, the user may require an intricate or application-specific process to convert between micro-scale variables $u$ and macro-scale variables $U$. Provide these bespoke functions to the toolbox PI functions with the optional inputs restrict() and lift().

The user-provided function restrict($u$) should map a micro-scale state $u$, at some fixed time, to a lower-dimensional macro-scale state $U$ at the same time. The reverse effect is accomplished by the function lift($U, u_{Approx}$), which takes two inputs—as lifting is often a non-unique process. The first input $U$ is the macro-scale state at the (present) time at which a micro-scale state is desired, and the second input $u_{Approx}$ is the last micro-scale state output from the micro-simulator. This $u_{Approx}$ is typically a micro-scale state from an earlier time, but nonetheless should be useful in initialising a consistent micro-scale state.

A guiding principle in the restriction and lifting functions is that we cannot anticipate a user’s every need; therefore, the functions are straightforward to edit. In particular, their inputs may readily be expanded as needed.

3.1.4 Choose PI over stiff integrators in high dimensions

The 2D example of Section 1.2 may be efficiently and simply simulated by standard stiff integrators, e.g. ode15s() in MATLAB. We here demonstrate the advantage of PI over such integrators as the model dimension increases. Consider linear systems of the form $\frac{du}{dt} = Au + b$, where $(10 + N) \times (10 + N)$ matrix $A$ is randomly generated so that it has ten eigenvalues with real part within $[-0.1, 0.1]$, corresponding to ten slow variables, and $N$ eigenvalues with real part within $[-20000, -10000]$, corresponding to $N$ fast variables. The vector $b \in \mathbb{R}^{10+N}$ is randomly generated with variance one.

We generate such a system at 20 values of $N$ between 0 and 90, and at each one simulate to final time 10 from randomly chosen initial conditions with each of PIRK4(), PIG() and ode15s(). After each simulation we record the time elapsed over each simulation as well as the relative error between the simulation estimates and the exact solution. This procedure is repeated a further eleven times (the toolbox script pirk4mance.m details code for this experiment): Fig. 4 displays the recorded statistics. It appears that for system dimension larger than about 60, the cost by ode15s in setting up and managing the Jacobian is too high. For these larger systems, the Projective Integration functions appear the better choice.

The relative performance of ode15s may be improved by providing it with more information. By providing both the Jacobian $A$ explicitly to the integrator, and specifying the initial conditions on the attracting slow manifold, ode15s becomes competitive with the Projective Integration functions (up to system dimension 190). But knowing and coding both the system Jacobian and slow manifold is usually too hard in practice.
Fig. 4 Performance of Projective Integration compared to a standard stiff integrator, while scaling the dimension of the numerical model described in Section 3.1.4. Solid lines show the median, and dotted lines show the 25th and 75th percentiles. The stiff integrator is fast and reasonably accurate at low system dimension, but performs poorly at dimension 60 and above.
3.2 Patch dynamics tutorial

The code here shows one way to get started with the patch scheme. A user’s script may have the following three steps (arrows indicate function recursion).

1. configPatches2
2. ode integrator ⇔ patchSmooth2 ⇔ user’s micro-scale code
3. process results

We describe the code to reproduce the simulation (Fig. 1) of the lattice spatial discretisation (1) of a nonlinear diffusion PDE. As a micro-scale discretisation of the PDE we use the following function to compute the time derivatives: an array variable \( u(i, j, :, :) \) refers to the \((i, j)\)th point in each and every patch as the third and fourth indices index the 2D array of patches.

```matlab
function ut = nonDiffPDE(t,u,x,y)
dx = diff(x(1:2)); dy = diff(y(1:2));  % microgrid spacing
i = 2:size(u,1)-1; j = 2:size(u,2)-1;  % interior patch points
ut = nan(size(u));  % preallocate storage
ut(i,j,:,:) = diff(u(:,j,:,:).^3,2,1)/dx^2 +diff(u(i,:,:,:).^3,2,2)/dy^2;
end
```

To use this micro-scale code on patches we need to establish global data, in struct patches, to characterise the patches. Here we aim to simulate the nonlinear diffusive (1) on a macro-scale \(6 \times 4\)-periodic domain with a 2D array of \(9 \times 7\) patches. Choose spectral interpolation (\(\text{ordCC} = 0\)) to couple the patches, and set each patch of half-size ratio 0.25 (relatively large for visualization), and with \(5 \times 5\) micro-scale lattice points within each patch. [4] established that such a patch scheme is consistent with the discretisation (1), as the patch spacing \(H\) decreases, and hence is consistent to the original PDE.

```matlab
global patches
nSubP = 5;
configPatches2(@nonDiffPDE,[-3 3 -2 2],nan,[9 7],0,0.25,nSubP);
```

The third argument to `configPatches2()` is intended for boundary conditions on the macro-scale simulation, not yet implemented in the toolbox. The inputs are otherwise 2D analogues of the inputs to `configPatches1` in Algorithm 2.

For an initial condition of the simulation, here set a perturbed-Gaussian. The `reshape` functions give the \(x\) and \(y\) lattice coordinates as two 4D arrays of size \(5 \times 1 \times 9 \times 1\) and \(1 \times 5 \times 1 \times 7\) respectively. Then, auto-replication fills \(u0\) with values from the Gaussian \(u = e^{-x^2-y^2}\).

```matlab
x = reshape(patches.x,nSubP,1,[],1);
y = reshape(patches.y,1,nSubP,1,[]);
u0 = exp(-(x.^2-y.^2));
u0 = u0.*(0.9+0.1*rand(size(u0)));
```

Initiate a plot of the simulation using only the micro-scale values interior to the patches: set \(x\) and \(y\)-edges to \(\text{nan}\) to leave, in plots, gaps between patches.
Start by showing the initial conditions—the top-left panel of Fig. 1.

\[
\begin{align*}
    u &= \text{reshape}(\text{permute}(u_0, [1 \ 3 \ 2 \ 4]), \ [\text{numel}(x) \ \text{numel}(y)]); \\
    hsurf &= \text{surf}(x(:,), y(:,), u'); \
    \text{drawnow}
\end{align*}
\]

Integrate in time using a standard function, or alternatively via projective integration.

\[
\begin{align*}
    [ts, us] &= \text{ode15s}( \text{@patchSmooth2}, \ [0 \ 4], u_0(:)); \\
    \text{Animate the computed simulation to finish with the bottom-right picture in Fig. 1.}
\end{align*}
\]

Use patchEdgeInt2 to interpolate patch-edge values (even if not drawn) as it reconstructs the row orientated output from \text{ode15s} into arrays that reflect the 2D patch structure.

\[
\begin{align*}
    \text{for } i &= 1:\text{length}(ts) \\
    u &= \text{patchEdgeInt2}(us(i,:)); \\
    u &= \text{reshape}(\text{permute}(u, [1 \ 3 \ 2 \ 4]), \ [\text{numel}(x) \ \text{numel}(y)]); \\
    \text{set}(hsurf, 'ZData', u'); \\
    \text{legend}(['\text{time} = ' \ \text{num2str}(ts(i),2)])
\end{align*}
\]

4 Discussion

This project is collectively developing a MATLAB/Octave toolbox of equation-free algorithms [26]. The algorithms currently implement a useful functionality, but much more is desirable so the plan is to subsequently develop more capability. MATLAB/Octave appears a good choice for a first version since it is widespread, efficient, supports various parallel modes, and development costs are reasonably low. Further, it is built on BLAS and LAPACK so the cache and superscalar CPU are potentially well utilised.

Projective Integration and Patch Scheme functions are designed to generally enable users to invoke ‘equation-free’ algorithms in a wide variety of applications. In addition to simulations analogous to those in previous sections, the toolbox may empower users in other scenarios such as the following list. In the User Manual [26] each toolbox function and application is presented in its own section, so here we refer to sections of the User Manual by using the name of the appropriate function.

- To projectively integrate in time a multiscale, slow-fast, system of ODES with a simple PI macro-integrator, you could use \text{PIRK2()}, or \text{PIRK4()} for higher-order accuracy. Perhaps adapt the Michaelis–Menten example presented in the User Manual at the beginning of \text{PIRK2.m}.
- One may use short forward bursts of micro-scale simulation in order to stably predict the slow dynamics backward in time using PI [45], as in \text{egPIMM.m}.
- The lifting and restriction functions may be utilised in more complicated applications like those others have previously addressed (e.g. [34–37]). The lifting
function has two inputs: the macro-scale state (after a macro-scale time step); and the micro-scale state at the end of the last micro-simulation. Full details for constructing these functions are in PIG.m.

- For the patch scheme in 1D adapt the code at the beginning of configPatches1.m for Burgers’ PDE, or the staggered patches of 1D water wave equations in waterWaveExample.m.
- For the patch scheme in 2D adapt the code at the beginning of configPatches2.m for nonlinear diffusion, or the regular patches of the 2D wave equation of wave2D.m.
- The previous two examples are for systems that have smooth spatial structures on the micro-scale: when the micro-scale is ‘rough’ with a known heterogeneous period, then adapt the example of homogenisationExample.m.
- Employ an ensemble of patch dynamics simulations, averaging appropriately, by adapting the example of homoDiffEdgy1.m.
- Combine the projective integration and patch functions to simulate a system with both time and spatial scale separation. In this case use a PI function as the integrator for a patch scheme, as done in the second portion of homogenisationExample.m.

We encourage adaptation and further development of the toolbox algorithms, and are keen to include new collaborators in future versions of the toolbox. In particular, as well as developing multi-D functions further, corresponding 1D coded functionality, we need to code and prove capability for general macro-scale boundaries. We also plan to develop projective integration for oscillatory/stochastic systems, perhaps via Dynamic Mode Decomposition (e.g. [46, 47]). Currently we are developing functionality in the Toolbox to support massively parallel computation based upon a spatial domain decomposition of the distributed patches [26, see the git-branch parallelMatlabPatches].

**Funding** This project is supported by the Australian Research Council via grants DP150102385 and DP180100050.

**References**

1. Kevrekidis, I.G., Samaey, G.: Equation-free multiscale computation: Algorithms and applications. Annu. Rev. Phys. Chem. 60, 321–44 (2009). https://doi.org/10.1146/annurev.physchem.59.032607.093610
2. Kevrekidis, I.G., Gear, C.W., Hummer, G.: Equation-free: the computer-assisted analysis of complex, multiscale systems. A. I. Ch. E. Journal 50, 1346–1354 (2004). https://doi.org/10.1002/aic.10106
3. Kevrekidis, I.G., Gear, C.W., Hyman, J.M., Kevrekidis, P.G., Runborg, O., Theodoropoulos, K.: Equation-free, coarse-grained multiscale computation: enabling microscopic simulators to perform system level tasks. Comm. Math. Sciences 1, 715–762 (2003). https://doi.org/10.4310/CMS.2003.v1.n4.a5
4. Roberts, A.J., MacKenzie, T., Bunder, J.E.: A dynamical systems approach to simulating macroscale spatial dynamics in multiple dimensions. J. Engineering Mathematics 86(1), 175–207 (2014). https://doi.org/10.1007/s10665-013-9653-6
5. Roberts, A.J., Kevrekidis, I.G.: General tooth boundary conditions for equation free modelling. SIAM J. Scientific Computing 29(4), 1495–1510 (2007). https://doi.org/10.1137/060654554
6. Saeb, S., Steinmann, P., Javili, A.: Aspects of computational homogenization at finite deformations: a unifying review from Reuss’ to Voigt’s bound. Appl. Mech. Rev. 68(5), 1–33 (2016). https://doi.org/10.1115/1.4034024

7. Geers, M.G.D., Kouznetsova, V.G., Matouš, K., Yvonnet, J.: Homogenization methods and multiscale modeling: Nonlinear problems. In: Encyclopedia of Computational Mechanics, Second Edition, pp 1–34. Wiley (2017). https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119176817.emc2107

8. Peterseim, D.: Numerical homogenization beyond scale separation and periodicity. Technical report, AMSI Winter School on Computational Modeling of Heterogeneous Media. https://ws.amsi.org.au/wp-content/uploads/sites/70/2019/06/nhomamsi2019.pdf (2019)

9. Craster, R.V.: Dynamic homogenization. In: Mityushev, V.V., Ruzhansky, M. (eds.) Springer Proceedings in Mathematics and Statistics, vol 116, pp 41–50. Springer (2015)

10. Gear, C.W., Kevrekidis, I.G.: Projective methods for stiff differential equations: problems with gaps in their eigenvalue spectrum. SIAM J. Sci. Comput. 24(4), 1091–1106 (2003). https://doi.org/10.1137/S1064827501388157. http://link.aip.org/link/?SCE/24/1091/1

11. Rico-Martinez, R., Gear, C.W., Kevrekidis, I.G.: Coarse projective kMC integration: forward/reverse initial and boundary value problems. J. Comput. Phys. 196(2), 474–489 (2004). https://doi.org/10.1016/j.jcp.2003.11.005. http://www.sciencedirect.com/science/article/B6WHY-4B8B9GY-1/2/e92e0d513d9f01ca9c449d37d9d1a80

12. Erban, R., Kevrekidis, I.G., Othmer, H.G.: An equation-free computational approach for extracting population-level behavior from individual-based models of biological dispersal. Physica D: Nonlinear Phenomena 215(1), 1–24 (2006). https://doi.org/10.1016/j.physd.2006.01.008. http://www.sciencedirect.com/science/article/B6TVK-4JDVNSP-1/2/f31e9e396a32dcb2a811f41ed6a6d6f6

13. Lafitte, P., Melis, W., Samaey, G.: A high-order relaxation method with projective integration for solving nonlinear systems of hyperbolic conservation laws. J. Comput. Phys. 340, 1–25 (2017). https://doi.org/10.1016/j.jcp.2017.03.027. http://www.sciencedirect.com/science/article/pii/S002199911730222X

14. Maclean, J., Gottwald, G.A.: On convergence of higher order schemes for the projective integration method for stiff ordinary differential equations. J. Comput. Appl. Math. 288, 44–69 (2015). https://doi.org/http://dx.doi.org/10.1016/j.cam.2015.04.004

15. Lafitte, P., Lejon, A., Samaey, G.: A high-order asymptotic-preserving scheme for kinetic equations using projective integration. SIAM Journal on Numerical Analysis 54(1), 1–33 (2016). https://doi.org/10.1137/140966708. https://epubs.siam.org/doi/abs/10.1137/140966708, Publisher: Society for Industrial and Applied Mathematics

16. Lafitte, P., Leijnse, H., Samaey, G.: A high-order asymptotic-preserving scheme for kinetic equations using projective integration. SIAM J. Sci. Comput. 32(4), 1729–1748 (2010). https://doi.org/10.1137/090760925. http://link.siam.org/doi/abs/10.1137/090760925, Publisher: Society for Industrial and Applied Mathematics

17. Zieliński, P., Lafitte, P., Samaey, G.: Convergence and stability of a micro–macro acceleration method: Linear slow–fast stochastic differential equations with additive noise. J. Comput. Phys. 340, 1–25 (2017). https://doi.org/10.1016/j.jcp.2017.03.027. http://www.sciencedirect.com/science/article/pii/S002199911730222X

18. Roberts, A.J.: Model emergent dynamics in complex systems. SIAM, Philadelphia (2015). http://bookstore.siam.org/mmm20/

19. Car, R., Parrinello, M.: Unified approach for molecular dynamics and density-functional theory. Phys. Rev. Lett. 55(22), 2471–2474 (1985). https://doi.org/10.1103/PhysRevLett.55.2471. https://link.aps.org/doi/10.1103/PhysRevLett.55.2471, Publisher: American Physical Society

20. Craster, R.V., Perthame, B.: Numerical passage from kinetic to fluid equations. SIAM Journal on Numerical Analysis 28(1), 26–42 (1991). https://doi.org/10.1137/0728002. https://epubs.siam.org/doi/abs/10.1137/0728002, Publisher: Society for Industrial and Applied Mathematics

21. Tao, M., Owhadi, H., Marsden, J.E.: Nonintrusive and structure preserving multiscale integration of stiff ODEs, SDEs, and Hamiltonian systems with hidden slow dynamics via flow averaging. Multiscale
24. E., W., Engquist, B., Li, X., Ren, W., Vanden-Eijnden, E.: Heterogeneous multiscale methods: a review. Communications in Computational Physics 2(3), 367–450 (2007). https://nyu-staging.pure.elsevier.com/en/publications/heterogeneous-multiscale-methods-a-review, Publisher: Global Science Press

25. Abdulle, A., Weinan, E., Engquist, B., Vanden-Eijnden, E.: The heterogeneous multiscale method. Acta Numerica 21, 1–87 (2012). https://doc.rero.ch/record/290539, Publisher: Cambridge University Press

26. Roberts, A.J., Maclean, J., Bunder, J.E.: Equation-free function toolbox for Matlab/Octave. Technical report, [https://github.com/uoa1184615/EquationFreeGit] (2020)

27. Siettos, C.I., Graham, M.D., Kevrekidis, I.G.: Coarse Brownian dynamics for nematic liquid crystals: bifurcation, projective integration, and control via stochastic simulation. J. Chemical Physics 118(22), 10149–10156 (2003). https://doi.org/10.1063/1.1572456

28. Chuang, C.Y., Han, S.M., Zepeda-Ruiz, L.A., Sinno, T.: On coarse projective integration for atomic deposition in amorphous systems. J. Chem. Phys. 143(13), 134703 (October 2, 2015). https://doi.org/10.1063/1.4931991

29. Lee, S.L., Gear, C.W.: Second-order accurate projective integrators for multiscale problems. J. Comput. Appl. Math. 201(1), 258–274 (2007)

30. E, W.: Analysis of the heterogeneous multiscale method for ordinary differential equations. Commun. Math. Sci. 1(3), 423–436 (2003). https://doi.org/10.4310/CMS.2003.v1.n3.a3

31. Maclean, J.: A note on implementations of the boosting algorithm and heterogeneous multiscale methods. SIAM J. Numer. Anal. 53(5), 2472–2487 (2015). https://doi.org/10.1137/140982374

32. Gear, C.W., Kaper, T.J., Kevrekidis, I.G., Zagaris, A.: Projecting to a slow manifold: singularly perturbed systems and legacy codes. SIAM J. Applied Dynamical Systems 4(3), 711–732 (2005). https://doi.org/10.1137/040608295

33. Gear, C.W., Kevrekidis, I.G.: Constraint-defined manifolds: a legacy code approach to low-dimensional computation. J. Sci. Comput. 25(1), 17–28 (2005). https://doi.org/10.1007/s10915-004-4630-x

34. Frederix, Y., Samaey, G., Vandekerckhove, C., Roose, D.: Equation-free methods for molecular dynamics: a lifting procedure. Proc. Appl. Meth. Mech. 7, 20100003–20100004 (2007). https://doi.org/10.1002/pamm.200700025

35. Bold, K.A., Rajendran, K., Rath, B., Kevrekidis, I.G.: An equation-free approach to coarse-graining the dynamics of networks. Technical report, [http://arxiv.org/abs/1202.5618v1] (2012)

36. Sieber, J., Marschler, C., Starke, J.: Convergence of equation-free methods in the case of finite time scale separation with application to deterministic and stochastic systems. SIAM J. Appl. Dyn. Syst. 17(4), 2574–2614 (January 2018). https://doi.org/10.1137/17M1126084

37. Roose, D., Nies, E., Li, T., Vandekerckhove, C., Samaey, G., Frederix, Y.: Lifting in equation-free methods for molecular dynamics simulations of dense fluids. Discrete and Continuous Dynamical Systems—Series B 11(4), 855–874 (April 2009). https://doi.org/10.3934/dcdsb.2009.11.855

38. Samaey, G., Roberts, A.J., Kevrekidis, I.G.: Equation-free computation: an overview of patch dynamical. In: Fish, J. (ed.) Multiscale Methods: Bridging the Scales in Science and Engineering, pp 216–246. Oxford University Press (2010)

39. Bunder, J.E., Roberts, A.J., Kevrekidis, I.G.: Good coupling for the multiscale patch scheme on systems with microscale heterogeneity. J. Computational Physics 337, 154–174 (2017). https://doi.org/10.1016/j.jcp.2017.02.004

40. Cao, M., Roberts, A.J.: Multiscale modelling couples patches of nonlinear wave-like simulations. IMA J. Applied Maths. 81(2), 228–254 (2016). https://doi.org/10.1093/imamat/hxv034

41. Cao, M., Roberts, A.J.: Multiscale modelling couples patches of wave-like simulations. In: McCue, S., Moroney, T., Mallet, D., Bunder, J. (eds.) Proceedings of the 16th Biennial Computational Techniques and Applications Conference, CTAC-2012, vol 54 of ANZIAM J., pp C153–C170 (May 2013)

42. Geers, M.G.D., Kouznetsova, V.G., Brekelmans, W.A.M.: Multi-scale computational homogenization: trends and challenges. J. Comput. Appl. Math. 234(7), 2175–2182 (August 2010). https://doi.org/10.1016/j.cam.2009.08.077

43. Owhadi, H.: Bayesian numerical homogenization. Multiscale Modeling & Simulation 13(3), 812–828 (2015). https://doi.org/10.1137/140974596

44. Maier, R., Peterseim, D.: Explicit computational wave propagation in micro-heterogeneous media. BIT Numer. Math. 59(2), 443–462 (June 2019). https://doi.org/10.1007/s10543-018-0735-8
45. Gear, C.W., Kevrekidis, I.G.: Computing in the past with forward integration. Phys. Lett. A 321, 335–343 (2003). https://doi.org/10.1016/j.physleta.2003.12.041
46. Kutz, J.N., Brunton, S.L., Brunton, B.W., Proctor, J.L.: Dynamic mode decomposition: data-driven modeling of complex systems. Number 149 in Other titles in applied mathematics. SIAM, Philadelphia (2016)
47. Kutz, J.N., Proctor, J.L., Brunton, S.L.: Applied koopman theory for partial differential equations and data-driven modeling of spatio-temporal systems. Complexity 6010634, 1–16 (2018). https://doi.org/10.1155/2018/6010634

Publisher’s note  Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.