

Equation-Free function toolbox for Matlab/Octave

A. J. Roberts* et al.

October 23, 2017

Contents

1	Introduction	3
1.1	Create, document and test algorithms	3
2	Projective integration of deterministic ODEs	6
2.1	projInt1()	6
2.2	A first test of basic projective integration	9
3	Runge–Kutta 2 integration of deterministic ODEs	12
3.1	rk2int()	12
3.2	A 1D test of RK2 integration	14
3.3	A 2D test of RK2 integration	14
A	Aspects of developing a ‘toolbox’ for patch dynamics	17
A.1	Macroscale grid	17
A.2	Macroscale field variables	17
A.3	Boundary and coupling conditions	18
A.4	Mesotime communication	18
A.5	Projective integration	19

*<http://www.maths.adelaide.edu.au/anthony.roberts>, <http://orcid.org/0000-0001-8930-1552>

A.6 Lift to many internal modes 19

A.7 Macroscale closure 20

A.8 Exascale fault tolerance 20

A.9 Link to established packages 21

1 Introduction

Users Place this toolbox’s folder in a path searched by MATLAB/Octave. Then read the subsection that documents the function of interest.

Blackbox scenario Assume that a researcher/practitioner has a detailed and *trustworthy* computational simulation of some problem of interest. The simulation may be written in terms of micro-positional coordinates $\vec{x}_i(t)$ in ‘space’ at which there are micro-field variable values $\vec{u}_i(t)$ for indices i in some (large) set of integers and for time t . In lattice problems the positions \vec{x}_i would be fixed in time (unless employing a moving mesh on the microscale); in particle problems the positions would evolve. The positional coordinates are $\vec{x}_i \in \mathbb{R}^d$ where for spatial problems integer $d = 1, 2, 3$, but it may be more when solving for a distribution of velocities, or pore sizes, or trader’s beliefs, etc. The micro-field variables could be in \mathbb{R}^p for any $p = 1, 2, \dots, \infty$.

Further, assume that the computational simulation is too expensive over all the desired spatial domain $\mathbb{X} \subset \mathbb{R}^d$. Thus we aim a toolbox to simulate only on macroscale distributed patches.

Developers The aim of this project is to collectively develop a MATLAB/Octave toolbox of equation-free algorithms. Initially the algorithms will be simple, and the plan is to subsequently develop more and more capability.

MATLAB appears the obvious choice for a first version since it is widespread, reasonably efficient, supports various parallel modes, and development costs are reasonably low. Further it is built on BLAS and LAPACK so potentially the cache and superscalar CPU are well utilised. Let’s develop functions that work for both MATLAB/Octave.

1.1 Create, document and test algorithms

For developers to create and document the various functions, we use an idea due to Neil D. Lawrence of the University of Sheffield: ?? gives an example

of the following structure to use.

- Each class of toolbox functions is located in separate directories in the repository, say `Dir`.
- Each toolbox function is documented as a separate subsection, with tests and examples as separate subsections.
- For each function, say `fun.m`, create a L^AT_EX file `Dir/fun.tex` of a section that `\input{Dir/*.m}`s the files of the function-subsection and the test-subsections, Table 1. Each such `Dir/fun.tex` file is to be `\include{}`ed from the main L^AT_EX file `equationFreeDoc.tex` so that people can most easily work on one section at a time.
- Each function-subsection and test-subsection is to be created as a MATLAB/Octave `Dir/*.m` file, say `Dir/fun.m`, so that users simply invoke the function in MATLAB/Octave as usual by `fun(...)`.

Some editors may need to be told that `fun.m` is a L^AT_EX file. For example, TexShop on the Mac requires one to execute in a Terminal

```
defaults write TexShop OtherTeXExtensions -array-add "m"
```

- Table 2 gives the template for the `Dir/*.m` function-subsections. The format for a example/test-subsection is similar.
- Currently I use the beautiful `minted` package to list code, but it does require a little more effort when installing.

Table 1: example `Dir/*.tex` file to typeset in the master document a function-subsection, say `fun.m`, and the test/example-subsections.

```
% input *.m files for ... author, date
%!TEX root = ../equationFreeDoc.tex
\section{...}
\label{sec:...}
introduction...
\input{Dir/fun.m}
\input{Dir/funExample.m}
...
```

Table 2: template for a function-subsection `Dir/*.m` file.

```
%Short explanation for users typing "help fun"
%Author, date
%!TEX root = ../equationFreeDoc.tex
%{
\subsection{\texttt{...}}
\label{sec:...}
Summary LaTeX explanation.
\begin{matlab}
%}
function ...
%{
\end{matlab}
Repeated as desired:
LaTeX between end-matlab and begin-matlab
\begin{matlab}
%}
Matlab code between %} and %{
%{
\end{matlab}
Concluding LaTeX before following final line.
%}
```

2 Projective integration of deterministic ODEs

This is a very first stab at a good projective integration function.

2.1 projInt1()

This is a basic example of projective integration of a given system of stiff deterministic ODEs via Dynamic Mode Decomposition (DMD).

```

25 %}
26 function [xs]=projInt1(fun,x0,Ts,rank,dt,nTimeSteps)
27 %{

```

Input

- **fun()** is a function such as **dxdt=fun(x,t)** that computes the right-hand side of the ODE $d\vec{x}/dt = \vec{f}(\vec{x}, t)$ where \vec{x} is a column vector, say in \mathbb{R}^n for $n \geq 1$, t is a scalar, and the result \vec{f} is a column vector in \mathbb{R}^n .
- **x0** is an \mathbb{R}^n vector of initial values at the time **ts(1)**.
- **Ts** is a vector of times to compute the approximate solution, say in \mathbb{R}^ℓ for $\ell \geq 2$.
- **rank** is the rank of the DMD extrapolation over macroscale time steps. Suspect **rank** should be at least one more than the number of slow variables.
- **dt** is the size of the microscale time-step. Must be small enough so that RK2 integration of the ODEs is stable.
- **nTimeSteps** is a two element vector:
 - **nTimeSteps(1)** is the number of microscale time-steps **dt** thought to be needed for microscale simulation to reach the slow manifold;

- `nTimeSteps(2)`, must be bigger than `rank`, is the number of subsequent time-steps to take which DMD analyses to mode the slow manifold.

Output

- `xs`, array in $\mathbb{R}^{n \times \ell}$ of approximate solution vector at the specified times—but the transpose of what MATLAB does!

Compute the time steps and create storage for outputs.

```

9  %}
10 DT=diff(Ts);
11 n=length(x0);
12 xs=nan(n,length(Ts));
13 %{
```

Initialise first result to the given initial condition.

```

9  %}
10 xs(:,1)=x0(:);
11 %{
```

Projectively integrate each of the time-steps from t_k to t_{k+1} .

```

7  %}
8  for k=1:length(DT)
9  %{
```

Microscale integration is simple, second order, Runge–Kutta method.

```

11 %}
12 x=[x0(:) nan(n,sum(nTimeSteps))];
13 for i=1:sum(nTimeSteps)
14     xh=x(:,i)+dt/2*fun(x(:,i),Ts(k)+(i-1)*dt);
15     x(:,i+1)=x(:,i)+dt*fun(xh,Ts(k)+(i-0.5)*dt);
16 end
17 %{
```

Grossly check on whether the microscale integration is stable.

```

15 %}
16 if norm(x(:,nTimeSteps(1)+(1:nTimeSteps(2)))) ...
17    > 10*norm(x(:,1:nTimeSteps(1)))
18    xMicroscaleIntegration=x, macroTime=Ts(k)
19    error('**** projInt1: microscale integration appears unstable')
20 end
21 %{

```

DMD extrapolation over the macroscale DMD appears to work better when ones are adjoined to the data vectors for some unknown reason.

```

13 %}
14 iFin=1+sum(nTimeSteps);
15 iStart=1+nTimeSteps(1);
16 x=[x;ones(1,iFin)];
17 %{

```

Then the basic DMD algorithm: first the fit.

```

19 %}
20 X1=x(:,iStart:iFin-1);
21 X2=x(:,iStart+1:iFin);
22 [U,S,V]=svd(X1,'econ');
23 S=diag(S);
24 Ur = U(:,1:rank); % truncate to rank=r, nxr
25 Sr = S(1:rank); % rx1
26 Vr = V(:,1:rank); % mrx where m is length of DMD analysis
27 AUr=bsxfun(@divide,X2*Vr,Sr'); % nxr
28 Atilde = Ur'*AUr; % low-rank dynamics, rrx
29 [Wr, D] = eig(Atilde); % rrx
30 Phi = AUr*Wr; % DMD modes, nrx
31 %{

```

Second, reconstruct a prediction for the time step. The current microsimulation time is `dt*iFin`, so step forward an amount to predict the sys-

tems state at **Ts(k+1)**. Perhaps should test **omega** and abort if 'large' and positive??

```

21 %}
22 omega = log(diag(D))/dt % continuous-time eigenvalues, rx1
23 %Phi=Phi
24 bFin=Phi\X(:,iFin); % rx1
25 x0=Phi(1:n,:)*(bFin.*exp(omega*(DT(k)-iFin*dt))); % nx1
26 xs(:,k+1)=x0;
27 %{

```

End the macroscale time stepping.

```

11 %}
12 end
13 %{

```

End of the function with result vectors returned in columns of **xs**, one column for each time in **Ts**.

2.2 A first test of basic projective integration

Seek to simulate the nonlinear diffusion PDE

$$\frac{\partial u}{\partial t} = u \frac{\partial^2 u}{\partial x^2} \quad \text{such that } u(\pm 1) = 0,$$

with random positive initial condition. Solutions are attracted to the slow decay of $u = a(t)(1 - x^2)$ with algebraic decay $\dot{a} = -2a^2$.

Set the number of interior points in the domain $[-1, 1]$, and the macroscale time step.

```

9 %}
10 function projInt1Test1
11 n=9
12 ts=0:2:6
13 %{

```

Set the initial condition to parabola or some skewed random positive values.

```

11 %}
12 x=linspace(-1,1,n+2)';
13 %u0=1-x(2:n+1).^2 +1e-9*randn(n,1);
14 u0=rand(n,1).*(1+x(2:n+1));
15 %{

```

Projectively integrate in time one step for the moment with: rank-two DMD projection; guessed microscale time-step; and guessed numbers of transient (15) and slow (7) steps.

```

9 %}
10 us=projInt1(@dudt,u0,ts,2,2/n^2,[round(n^2/5.4) 7])
11 %{

```

Plot the solutions to draw [Figure 1](#).

```

10 %}
11 plot(x,[0*ts;us;0*ts],'o-')
12 xlabel('space x'),ylabel('u(x,t)')
13 %matlab2tikz('projInt1Test1u.ltx','noSize',true)
14 print('-depsc2',['projInt1Test1u' num2str(n)])
15 %{

```

End the main function (not needed for new enough Matlab).

```

10 %}
11 end
12 %{

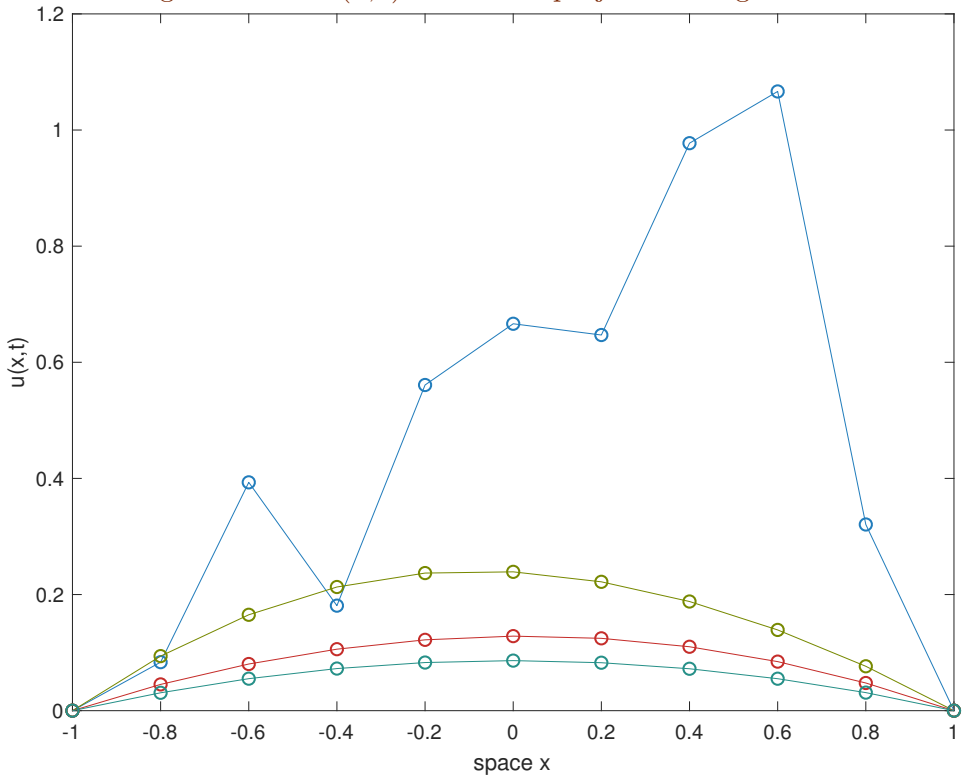
```

Code the simple centred difference discretisation of the nonlinear diffusion PDE with zero boundary values.

```

13 %}
14 function ut=dudt(u,t)
15 n=length(u);
16 h=2/(n+1);

```

Figure 1: field $u(x, t)$ tests basic projective integration.

```

17 j=2:n+1;
18 u=[0;u;0];
19 ut=u(j).*(u(j+1)-2*u(j)+u(j-1))/h^2;
20 end
21 %t

```

3 Runge–Kutta 2 integration of deterministic ODEs

This describes a simple RK2 integration function, and some tests, as an example of the layout of information.

3.1 rk2int()

This is a simple example of Runge–Kutta, 2nd order, integration of a given deterministic ODE.

```

13 %}
14 function [xs,errs]=rk2int(fun,x0,ts)
15 %{

```

Input

- **fun()** is a function such as **dxdt=fun(x,t)** that computes the right-hand side of the ODE $d\vec{x}/dt = \vec{f}(\vec{x},t)$ where \vec{x} is a column vector, say in \mathbb{R}^n for $n \geq 1$, t is a scalar, and the result \vec{f} is a column vector in \mathbb{R}^n .
- **x0** is an \mathbb{R}^n vector of initial values at the time **ts(1)**.
- **ts** is a vector of times to compute the approximate solution, say in \mathbb{R}^ℓ for $\ell \geq 2$.

Output

- **xs**, array in $\mathbb{R}^{n \times \ell}$ of approximate solution vector at the specified times—but the transpose of what MATLAB does!
- **errs**, vector in $\mathbb{R}^{1 \times \ell}$ of error estimate for the step from t_{k-1} to t_k .

Compute the time steps and create storage for outputs.

```

9 %}
10 dt=diff(ts);

```

```

11  xs=nan(length(x0),length(ts));
12  errs=nan(1,length(ts));
13  %{

```

Initialise first result to the given initial condition, and evaluate the initial time derivative into **f1**.

```

11  %}
12  xs(:,1)=x0(:);
13  errs(1)=0;
14  f1=fun(xs(:,1),ts(1));
15  %{

```

Compute the time-steps from t_k to t_{k+1} , copying the derivative **f1** at the end of the last time-step to be the derivative at the start of this one.

```

10  %}
11  for k=1:length(dt)
12      f0=f1;
13  %{

```

Simple second order accurate time step.

```

11  %}
12      xh=xs(:,k)+f0*dt(k)/2;
13      fh=fun(xh,ts(k)+dt(k)/2);
14      xs(:,k+1)=xs(:,k)+fh*dt(k);
15      f1=fun(xs(:,k+1),ts(k+1));
16  %{

```

Use the time derivative at t_{k+1} to estimate an error by storing the difference with what Simpson's rule would estimate.

```

11  %}
12      errs(k+1)=norm(f0-2*fh+f1)*dt(k)/6;
13  end
14  %{

```

End of the function with results returned in **xs** and **errs**.

3.2 A 1D test of RK2 integration

Try the nonlinear, but separable, ODE

$$\dot{x} = -2x/t \quad \implies \quad x = c/t^2.$$

```

8  %}
9  fn=@(x,t) [-2*x/t]
10 %{

Solve ODE over  $1 \leq t \leq 4$  with initial condition  $x(1) = 1$ 

10 %}
11 ts=linspace(1,4,21);
12 [xs,errs]=rk2int(fn,1,ts)
13 plot(ts,xs,ts,100*errs,'o',ts,100*(xs-1 ./ts.^2),'x')
14 print -depsc2 RKInt/rk2intPlot1.eps
15 %{

```

Figure 2 shows the step-errors would accumulate to a reasonably conservative estimate of the solution error.

3.3 A 2D test of RK2 integration

Try the nonlinear fast-slow system

$$\dot{x} = -xy, \quad \dot{y} = -y + x^2.$$

```

10 %}
11 fn=@(x,t) [-x(1)*x(2);-x(2)+x(1)^2]
12 %{

```

Solve over $0 \leq t \leq 5$ with initial condition way off the slow manifold of $x(0) = 1$ and $y(0) = -1$.

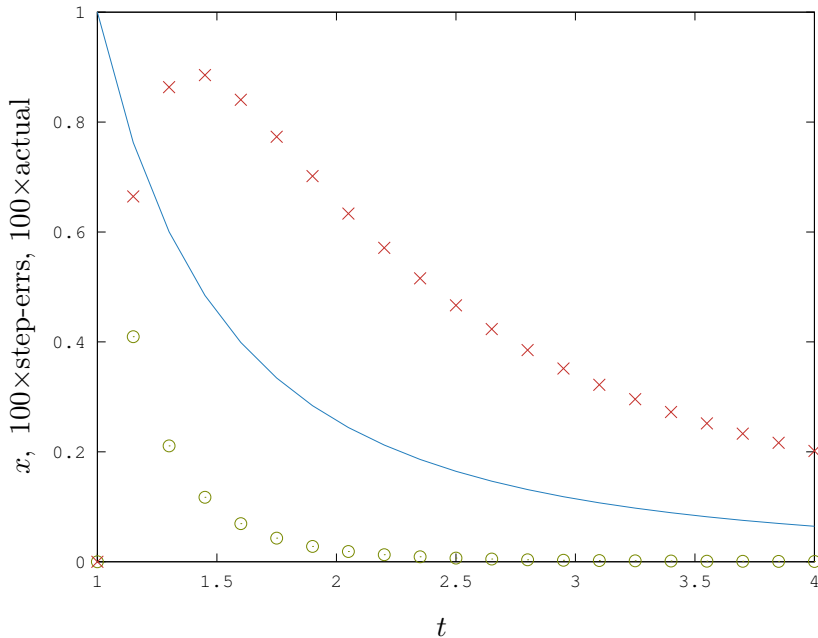


Figure 2: example trajectory and errors

```

10 %}
11 ts=linspace(0,5,21);
12 [xs,errs]=rk2int(fn,[1;-1],ts)
13 plot(ts,xs,ts,100*errs,'o')
14 print -depsc2 RKInt/rk2intPlot2.eps
15 %{\

```

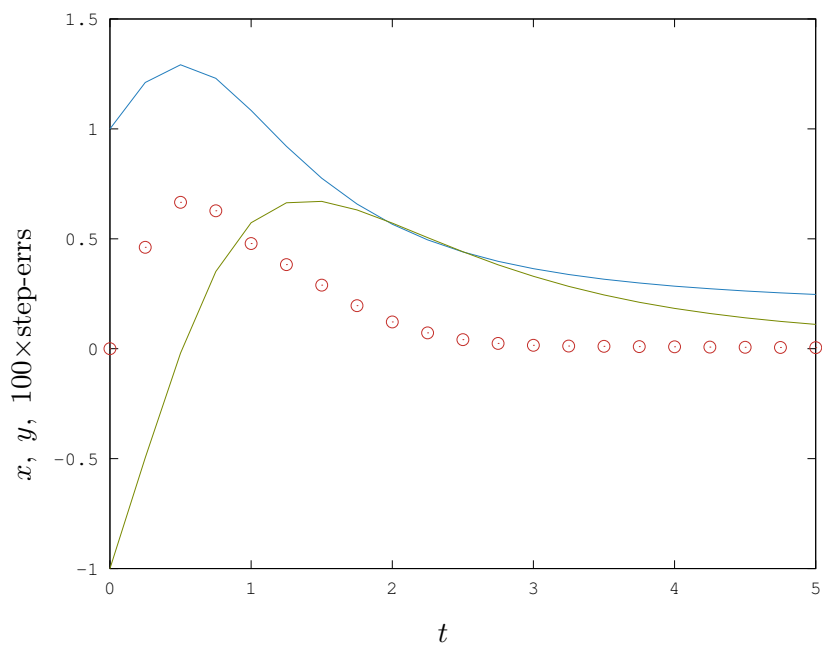


Figure 3: example trajectory and errors

A Aspects of developing a ‘toolbox’ for patch dynamics

This appendix documents sketchy further thoughts on aspects of the development.

A.1 Macroscale grid

The patches are to be distributed on a macroscale grid: the j th patch ‘centred’ at position $\vec{X}_j \in \mathbb{X}$. In principle the patches could move, but let’s keep them fixed in the first version. The simplest macroscale grid will be rectangular (`meshgrid`), but we plan to allow a deformed grid to secondly cater for boundary fitting to quite general domain shapes \mathbb{X} . And plan to later allow for more general interconnect networks for more topologies in application.

A.2 Macroscale field variables

The researcher/practitioner has to know an appropriate set of macroscale field variables $\vec{U}(t) \in \mathbb{R}^{d_{\vec{U}}}$ for each patch. For example, first they might be a simple average over a core of a patch of all of the micro-field variables; second, they might be a subset of the average micro-field variables; and third in general the macro-variables might be a nonlinear function of the micro-field variables (such as temperature is the average speed squared). The core might be just one point, or a sizeable fraction of the patch.

The mapping from microscale variable to macroscale variables is often termed the restriction.

In practice, users may not choose an appropriate set of macro-variables, so will eventually need to code some diagnostic to indicate a failure of the assumed closure.

A.3 Boundary and coupling conditions

The physical domain boundary conditions are distinct from the conditions coupling the patches together. Start with physical boundary conditions of periodicity in the macroscale.

Second, assume the physical boundary conditions are that the macro-variables are known at macroscale grid points around the boundary. Then the issue is to adjust the interpolation to cater for the boundary presence and shape. The coupling conditions for the patches should cater for the range of Robin-like boundary conditions, from Dirichlet to Neumann. Two possibilities arise: direct imposition of the coupling action (Roberts and Kevrekidis 2007), or control by the action.

Third, assume that some of the patches have some edges coincident with the boundary of the macroscale domain \mathbb{X} , and it is on these edges that macroscale physical boundary conditions are applied. Then the interpolation from the core of these edge patches is the same as the second case of prescribed boundary macro-variables. An issue is that each boundary patch should be big enough to cater for any spatial boundary layers transitioning from the applied boundary condition to the interior slow evolution.

Alternatively, we might have the physical boundary condition constrain the interpolation between patches.

Often microscale simulations are easiest to write when ‘periodic’ in microscale space. To cater for this we should also allow a control at perhaps the quartiles of a micro-periodic simulator.

A.4 Mesotime communication

Since communication limits large scale parallelism, a first step in reducing communication will be to implement only updating the coupling conditions when necessary. Error analysis indicates that updating on times longer the microscale times and shorter than the macroscale times can be effective (Bunder, Roberts, and Kevrekidis 2016). Implementations can communicate

one or more derivatives in time, as well as macroscale variables.

At this stage we can effectively parallelise over patches: first by simply using Matlab’s `parfor`. Probably not using a GPU as we probably want to leave GPUs for the black box to utilise within each patch.

A.5 Projective integration

To take macroscale time steps, invoke several possible projective integration schemes: simple Euler projection, Heun-like method, etc (Samaey, Roberts, and Kevrekidis 2010). Need to decide how long a microscale burst needs to be.

Should not need an implicit scheme as the fast dynamics are meant to be only in the micro variables, and the slow dynamics only in the macroscale variables. However, it could be that the macroscale variables have fast oscillations and it is only the amplitude of the oscillations that are slow. Perhaps need to detect and then fix or advise.

A further stage is to implement a projective integration scheme for stochastic macroscale variables: this is important because the averaging over a core of microscale roughness will almost invariably have at least some stochastic legacy effect.

A.6 Lift to many internal modes

In most problems the number of macroscale variables at any given position in space, $d_{\vec{U}}$, is less than the number of microscale variables at a position, $d_{\vec{u}}$; often much less (Kevrekidis and Samaey 2009, e.g.). In this case, every time we start a patch simulation we need to provide $d_{\vec{u}} - d_{\vec{U}}$ data at each position in the patch: this is lifting. The first methodology is to first guess, then run repeated short bursts with reinitialisation, until the simulation reaches a slow manifold. Then run the real simulation.

If the time taken to reach a local quasi-equilibrium is too long, then it is likely that the macroscale closure is bad and the macroscale variables need

to be extended.

A second step is to cater for cases where the slow manifold is stochastic or is surrounded by fast waves: when it is hard to detect the slow manifold, or the slow manifold is not attractive.

A.7 Macroscale closure

In some circumstances a researcher/practitioner will not code the appropriately set of macroscale variables for a complete closure of the macroscale. For example, in thin film fluid dynamics at low Reynolds number the only macroscale variable is the fluid depth; however, at higher Reynolds number, circa ten, the inertia of the fluid becomes important and the macroscale variables must additionally include a measure of the mean lateral velocity/momentum (Roberts and Li 2006, e.g.).

At some stage we need to detect any flaw in the closure, and perhaps suggest additional appropriate macroscale variables, or at least their characteristics. Indeed, a poor closure and a stochastic slow manifold are really two faces of the same problem: the problem is that the chosen macroscale variables do not have a unique evolution in terms of themselves. A good resolution of the issue will account for both faces.

A.8 Exascale fault tolerance

Matlab is probably not an appropriate vehicle to deal with real exascale faults. However, we should cater by coding procedures for fault tolerance and testing them at least synthetically. Eventually provide hooks to a user routine to be invoked under various potential scenarios. The nature of fault tolerant algorithms will vary depending upon the scenario, even assuming that each patch burst is executed on one CPU (or closely coupled CPUs): if there are much more CPUs than patches, then maybe simply duplicate all patch simulations; if much less CPUs than patches, then an asynchronous scheduling of patch bursts should effectively cater for recomputation of failed bursts; if comparable CPUs to patches, then more subtle action is needed.

Once mesotime communication and projective integration is provided, a recomputation approach to intermittent hardware faults should be effective because we then have the tools to restart a burst from available macroscale data. Should also explore proceeding with a lower order interpolation that misses the faulty burst—because an isolated lower order interpolation probably will not affect the global order of error (it does not in approximating some boundary conditions (Gustafsson 1975; Svärd and Nordström 2006))

A.9 Link to established packages

Several molecular/particle/agent based codes are well developed and used by a wide community of researchers. Plan to develop hooks to use some such codes as the microscale simulators on patches. First, plan to connect to LAMMPS (Plimpton et al. 2016). Second, will evaluate performance, issues, and then consider what other established packages are most promising.

References

- Bunder, Judith, A. J. Roberts, and Ioannis G. Kevrekidis (2016). “Accuracy of patch dynamics with mesoscale temporal coupling for efficient massively parallel simulations”. In: *SIAM Journal on Scientific Computing* 38.4, pp. C335–C371. DOI: [10.1137/15M1015005](https://doi.org/10.1137/15M1015005) (cit. on p. 18).
- Gustafsson, Bertil (1975). “The convergence rate for difference approximations to mixed initial boundary value problems”. In: *Mathematics of Computation* 29.10, pp. 396–406 (cit. on p. 21).
- Kevrekidis, Ioannis G. and Giovanni Samaey (2009). “Equation-Free Multi-scale Computation: Algorithms and Applications”. In: *Annu. Rev. Phys. Chem.* 60, pp. 321–44. DOI: [10.1146/annurev.physchem.59.032607.093610](https://doi.org/10.1146/annurev.physchem.59.032607.093610) (cit. on p. 19).
- Plimpton, Steve et al. (2016). *Large-scale Atomic/Molecular Massively Parallel Simulator*. URL: <http://lammps.sandia.gov> (cit. on p. 21).
- Roberts, A. J. and I. G. Kevrekidis (2007). “General tooth boundary conditions for equation free modelling”. In: *SIAM J. Scientific Computing* 29.4, pp. 1495–1510. DOI: [10.1137/060654554](https://doi.org/10.1137/060654554) (cit. on p. 18).
- Roberts, A. J. and Zhenquan Li (Apr. 2006). “An accurate and comprehensive model of thin fluid flows with inertia on curved substrates”. In: *J. Fluid Mech.* 553, pp. 33–73. DOI: [10.1017/S0022112006008640](https://doi.org/10.1017/S0022112006008640) (cit. on p. 20).
- Samaey, G., A. J. Roberts, and I. G. Kevrekidis (2010). “Equation-free computation: an overview of patch dynamics”. In: *Multiscale methods: bridging the scales in science and engineering*. Ed. by Jacob Fish. Oxford University Press. Chap. 8, pp. 216–246. ISBN: 978-0-19-923385-4 (cit. on p. 19).
- Svärd, Magnus and Jan Nordström (2006). “On the order of accuracy for difference approximations of initial-boundary value problems”. In: *Journal of Computational Physics* 218, pp. 333–352. DOI: [10.1016/j.jcp.2006.02.014](https://doi.org/10.1016/j.jcp.2006.02.014) (cit. on p. 21).