Projective integration of deterministic ODEs via DMD

*A. J. Roberts**

February 25, 2023

Contents

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

To do

- Try examples like the modulated oscillations of 2302.09341.pdf
- Check the order of accuracy of the algorithm.
- Develop theory quantitively justifying the DMD approach.
- Develop techniques to automatically make some of the decisions about step-sizes, burst lengths, rank, and so on.
- Develop higher accuracy versions (once we have some idea about current accuracy).
- Adapt approach to algorithms for stochastic systems.

* School of Mathematical Sciences, University of Adelaide, South Australia. https://profajroberts.github.io, http://orcid.org/0000-0001-8930-1552

1 projIntDMD()

Section contents

This is a basic example of projective integration of a given system of stiff deterministic ODEs via DMD, the Dynamic Mode Decomposition (Kutz et al. 2016).

```
16 function [xs,xss,tss]=projIntDMD(fun,x0,Ts,rank,dt,timeSteps)
```

Input

- `fun()` is a function such as `dxdt=fun(t,x)` that computes the right-hand side of the ODE $d\vec{x}/dt = \vec{f}(t,\vec{x})$ 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 n-vector of initial values at the time `ts(1)`. If any entries in `x0` are `NaN`, then `fun()` must cope, and only the non-`NaN` components are projected in time.
- `Ts` is a vector of times to compute the approximate solution, say in $\mathbb{R}^\ell$ 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 effective 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.
- `timeSteps` is a two element vector:
  - `timeSteps(1)` is the time thought to be needed for microscale simulation to reach the slow manifold;
  - `timeSteps(2)` is the subsequent time which DMD analyses to model the slow manifold (must be longer than `rank` $\cdot$ `dt`).

Output

- `xs`, $n \times \ell$ array of approximate solution vector at the specified times (the transpose of what MATLAB integrators do!)
- `xss`, optional, $n \times$ big array of the microscale simulation bursts—separated by NaNs for possible plotting.
- `tss`, optional, $1 \times$ big vector of times corresponding to the columns of `xss`.

Compute the time-steps and create storage for outputs.

```
72 DT=diff(Ts);
73 n=length(x0);
```

```
74  xs=nan(n,length(Ts));
75  xss=[];tss=[];
```

If any `x0` are `NaN`, then assume the time derivative routine can cope, and here we just exclude these from DMD projection and from any error estimation. This allows a user to have space in the solutions for breaks in the data vector (that, for example, may be filled in with boundary values for a PDE discretisation).

```
86  j=find(~isnan(x0));
```

If either of the timeSteps are non-integer valued, then assume they are both times, instead of micro-time-steps, so set the number of time-steps accordingly (multiples of `dt`).

```
95  timeSteps=round(timeSteps/dt);
96  timeSteps(2)=max(rank+1,timeSteps(2));
```

Set an algorithmic tolerance for miscellaneous purposes. As at Jan 2018, it is a guess. It might be similar to some level of microscale 'noise' in the burst. Also, in an oscillatory mode for projection, set the expected maximum number of cycles in a projection.

```
107 algTol=log(1e8);
108 cycMax=3;
```

Initialise first result to the given initial condition.

```
115 xs(:,1)=x0(:);
```

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

```
122 for k=1:length(DT)
```

Microscale integration is simple, second-order, Runge–Kutta method. Reasons: the start-up time for implicit integrators, such as ode15s, is too onerous to be worthwhile for each short burst; the microscale time-step needed for stability of explicit integrators is so small that a low order method is usually accurate enough.

```
133 x=[x0(:) nan(n,sum(timeSteps))];
134 for i=1:sum(timeSteps)
135     xmid  =x(:,i)+dt/2*fun(Ts(k)+(i-1)*dt,x(:,i));
136     x(:,i+1)=x(:,i)+dt*fun(Ts(k)+(i-0.5)*dt,xmid);
137 end
```

If user requests microscale bursts, then store.

```
143 if nargout>1,xss=[xss x nan(n,1)];
144 if nargout>2,tss=[tss Ts(k)+(0:sum(timeSteps))*dt nan];
145 end,end
```

Grossly check on whether the microscale integration is stable. Use the 1-norm, the largest column sum of the absolute values, for little reason. Is this any use??

```
if norm(x(j,ceil(end/2):end),1) ...
  > 10*norm(x(j,1:floor(end/2)),1)
  xMicroscaleIntegration=x, macroTime=Ts(k)
  warning('projIntDMD: microscale integration appears unstable')
  break%out of the integration loop
end
```

Similarly if any non-numbers generated.

```
if sum(~isfinite(x(:)))>0
  break%out of integration loop
end
```

DMD extrapolation over the macroscale But skip if the simulation has already reached the next time.

```
iFin=1+sum(timeSteps);
DTgap=DT(k)-iFin*dt;
if DTgap*sign(dt)<=1e-9
   i=round(DT(k)/dt); x0(j)=x(:,i+1);
   else
```

DMD appears to work better when ones are adjoined to the data vectors. [1]

```
iStart=1+timeSteps(1);
x=[x;ones(1,iFin)]; j1=[j;n+1];
```

Then the basic DMD algorithm: first the fit. However, need to test whether we need to worry about the microscale time-step being too small and leading to an effect analogous to 'numerical differentiation' errors: akin to the rule-of-thumb in fitting chaos with time-delay coordinates that a good time-step is approximately the time of the first zero of the autocorrelation.

```
[U,S,V]=svd(x(j1,iStart:iFin-1),'econ');
S=diag(S);
Sr = S(1:rank); % singular values, rx1
AUr=bsxfun(@rdivide,x(j1,iStart+1:iFin)*V(:,1:rank),Sr.');%nxr
Atilde = U(:,1:rank)'*AUr; % low-rank dynamics, rxr
[Wr, D] = eig(Atilde); % rxr
Phi = AUr*Wr; % DMD modes, nxr
```

Second, reconstruct a prediction for the time-step. The current micro-simulation time is `dt*iFin`, so step forward an amount to predict the systems state at `Ts(k+1)`. Perhaps should test ω and abort if 'large' and/or positive?? Answer: not necessarily as if the rank is large then the omega could contain large negative values.

[1] A reason is as follows. Consider the one variable linear ODE $\dot{x} = f + J(x - x_0)$ with $x(0) = x_0$ (as from a local linearisation of nonlinear ODE). The solution is $x(t) = (x_0 - f/J) + (f/J)e^{Jt}$ which sampled at a time-step τ is $x_k = (x_0 - f/J) + (f/J)G^k$ for $G := e^{J\tau}$. Then $x_{k+1} \neq ax_k$ for any a. However, $\left[\begin{smallmatrix} x_{k+1} \\ 1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} G & a \\ 0 & 1 \end{smallmatrix}\right] \left[\begin{smallmatrix} x_k \\ 1 \end{smallmatrix}\right]$ for a constant $a := (x_0 - f/J)(1 - G)$. That is, with ones adjoined, the data from the ODE fits the DMD approach.

Table 1: criterion for deciding if some DMD modes are to be neglected, and if not neglected then are they growing too badly?

neglectness	range for $\varepsilon \approx 10^{-8}$	reason
$\max(0, -\log_e \lvert b_i \rvert)$	$0-19$	Very small noise in the burst implies a numerical error mode.
$\max(0, -\Re\omega_i \, \Delta t)$	$0-19$	Rapidly decaying mode of the macro-time-step is a micro-mode that happened to be resolved in the data.
badness		provided not already neglected
$\max(0, +\Re\omega_i \, \Delta t)$	$0-19$	Micro-scale mode that rapidly grows, so macro-step should be smaller.
$\frac{3}{C}\lvert\Im\omega_i\rvert\Delta t$	$0-19$	An oscillatory mode with $\geq C$ cycles in macro-step Δt.

```
omega = log(diag(D))/dt; % continuous-time eigenvalues, rx1
bFin=Phi\x(j1,iFin); % rx1
```

But we want to neglect modes that are insignificant as characterised by Table 1, or be warned of modes that grow too rapidly. Assume appropriate to sum the neglect-ness, and the badness, for testing. Then warn if there is a mode that is too bad.

```
DTgap=DT(k)-iFin*dt;
negness=max(0,-log(abs(bFin)))+max(0,-real(omega*DTgap));
badness=max(0,+real(omega*DTgap))+3/cycMax*abs(imag(omega))*DTgap;
iOK=find(negness<algTol);
iBad=find(badness(iOK)>algTol);
if ~isempty(iBad)
    warning('projIntDMD: some bad modes in projection')
    badness=badness(iOK(iBad))
    rank=rank
    burstDt=timeSteps*dt
    break
    end
```

Scatter the prediction into the non-Nan elements of `x0`.

```
x0(j)=Phi(1:end-1,iOK)*(bFin(iOK).*exp(omega(iOK)*DTgap)); % nx1
```

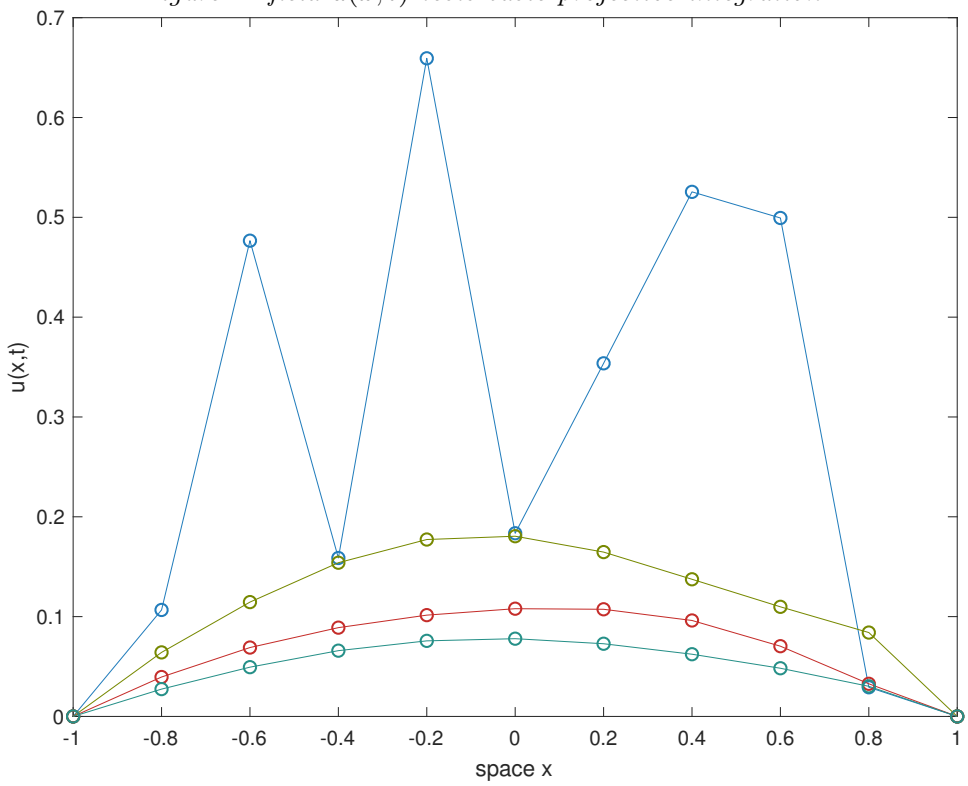
End the omission of the projection in the case when the burst is longer than the macroscale step.

```
end
```

Since some of the ω may be complex, if the simulation burst is real, then force the DMD prediction to be real.

```
if isreal(x), x0=real(x0); end
```

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




```
xs(:,k+1)=x0;
```

End the macroscale time-stepping.

```
end
```

If requested, then add the final point to the microscale data.

```
if nargout>1,xss=[xss x0];
if nargout>2,tss=[tss Ts(end)];
end,end
```

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

2 `projIntDMDExample1`: A first test of basic projective integration

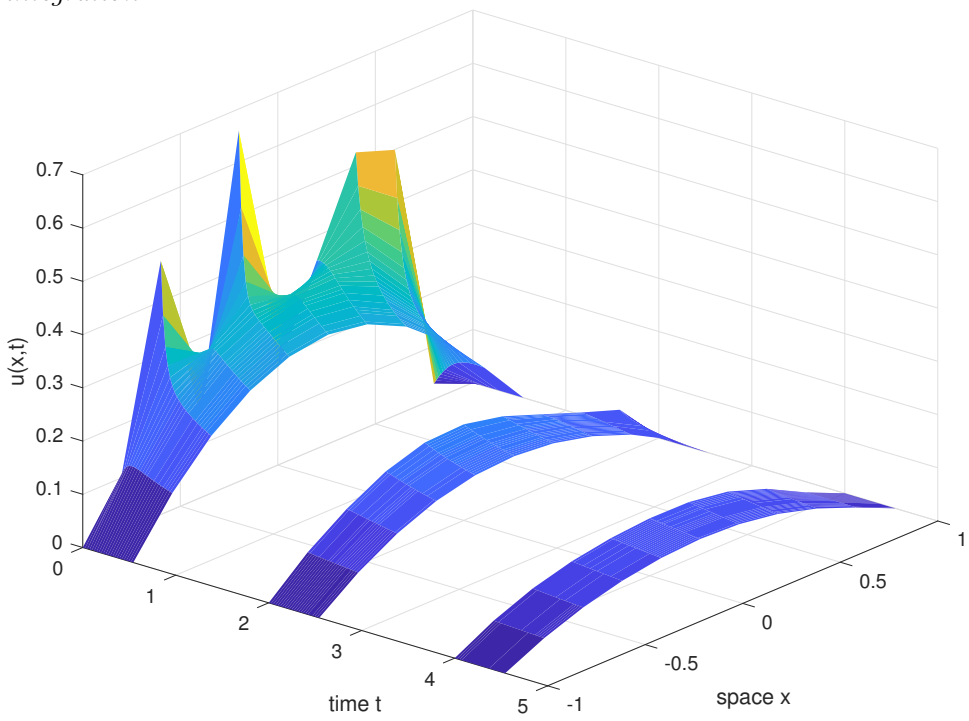
Section contents

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. Figure 1 shows solutions are attracted to the parabolic $u = a(t)(1-x^2)$ with slow algebraic decay $\dot{a} = -2a^2$.

Figure 2: field $u(x,t)$ during each of the microscale bursts used in the projective integration.



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

```
function projIntDMDExample1
n=9
ts=0:2:6
```

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

```
x=linspace(-1,1,n+2)';
%u0=(1-x.^2).*(1+1e-9*randn(n+2,1));
u0=rand(n+2,1).*(1-x.^2);
```

Projectively integrate in time with: rank-two DMD projection; guessed microscale time-step but chosen so an integral number of micro-steps fits into a macro-step for comparison; and guessed transient time 0.4 and 7 micro-steps 'on the slow manifold'.

```
dt=2/n^2
[us,uss,tss]=projIntDMD(@dudt,u0,ts,2,dt,[0.4 7*dt])
```

Plot the macroscale predictions to draw Figure 1.

```
clf,plot(x,us,'o-')
xlabel('space x'),ylabel('u(x,t)')
%matlab2tikz('pi1Example1u.ltx','noSize',true)
%print('-depsc2',['pi1Example1u' num2str(n)])
```

Also plot a surface of the microscale bursts as shown in Figure 2.

```
tss(end)=nan;% omit the last time point
clf,surf(tss,x,uss,'EdgeColor','none')
ylabel('space x'),xlabel('time t'),zlabel('u(x,t)')
view([40 30])
%print('-depsc2',['pi1Example1micro' num2str(n)])
```

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

```
end
```

The nonlinear PDE discretisation Code the simple centred difference discretisation of the nonlinear diffusion PDE with constant (usually zero) boundary values.

```
function ut=dudt(t,u)
n=length(u);
dx=2/(n-1);
j=2:n-1;
ut=[0
    u(j).*(u(j+1)-2*u(j)+u(j-1))/dx^2
    0];
end
```

3 `projIntDMDPatches`: Projective integration of patch scheme

Section contents

As an example of the use of projective integration, seek to simulate the nonlinear Burgers' PDE

$$\frac{\partial u}{\partial t} + cu\frac{\partial u}{\partial x} = \frac{\partial^2 u}{\partial x^2} \quad \text{for } 2\pi\text{-periodic } u,$$

for $c = 30$, and with various initial conditions. Use a patch scheme (Roberts & Kevrekidis 2007) to only compute on part of space as shown in Figure 3.

Function header and variables needed by discrete patch scheme.

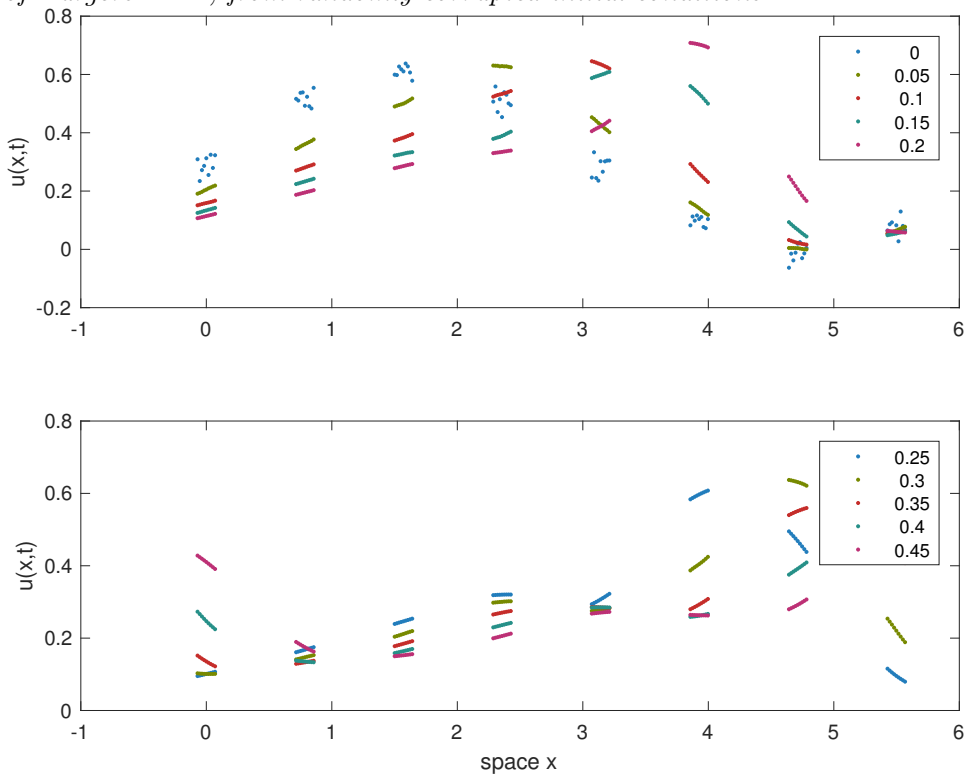
```
function projIntDMDPatches
global dx DX ratio j jp jm i I
```

Set parameters of the patch scheme: the number of patches; number of micro-grid points within each patch; the patch size to macroscale ratio.

```
nPatch=8
nSubP=11
ratio=0.1
```

The points in the microscale, sub-patch, grid are indexed by `i`, and `I` is the index of the mid-patch value used for coupling patches. The macroscale patches are indexed by `j` and the neighbours by `jp` and `jm`.

Figure 3: field $u(x,t)$ tests basic projective integration of a basic patch scheme of Burgers' PDE*, from randomly corrupted initial conditions.*




```
i=2:nSubP-1; % microscopic internal points for PDE
I=round((nSubP+1)/2); % midpoint of each patch
j=1:nPatch; jp=mod(j,nPatch)+1; jm=mod(j-2,nPatch)+1; % patch index
```

Make the spatial grid of patches centred at X_j and of half-size $h = r\Delta X$. To suit Neumann boundary conditions on the patches make the micro-grid straddle the patch boundary by setting $dx = 2h/(n_\mu - 2)$. In order for the microscale simulation to be stable, we should have $dt \ll dx^2$. Then generate the microscale grid locations for all patches: x_{ij} is the location of the ith micro-point in the jth patch.

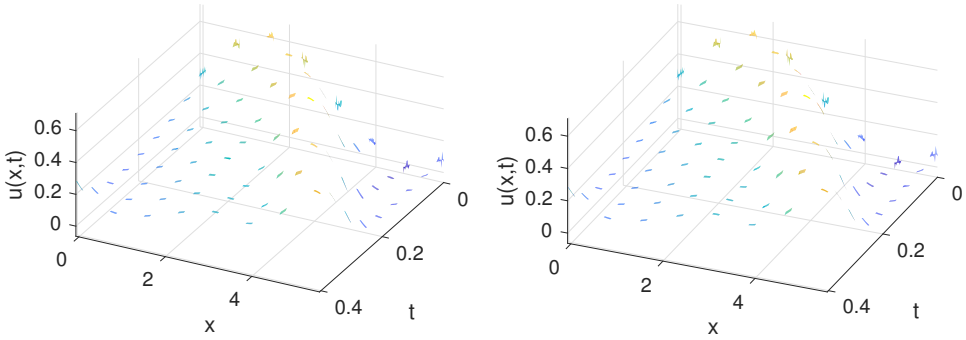
```
X=linspace(0,2*pi,nPatch+1); X=X(j); % patch mid-points
DX=X(2)-X(1) % spacing of mid-patch points
dx=2*ratio*DX/(nSubP-2) % micro-grid size
dt=0.4*dx^2; % micro-time-step
x=bsxfun(@plus,dx*(-I+1:I-1)',X); % micro-grids
```

Set the initial condition of a sine wave with random perturbations, surrounded with entries for boundary values of each patch.

```
u0=[nan(1,nPatch)
    0.3*(1+sin(x(i,:)))+0.03*randn(size(x(i,:)))
    nan(1,nPatch)];
```

Set the desired macroscale time-steps over the time domain.

Figure 4: stereo pair of the field $u(x,t)$ during each of the microscale bursts used in the projective integration.




```
ts=linspace(0,0.45,10)
```

Projectively integrate in time with: DMD projection of rank `nPatch` + 1; guessed microscale time-step `dt`; and guessed numbers of transient and slow steps.

```
[us,uss,tss]=projIntDMD(@dudt,u0(:),ts,nPatch+1,dt,[20 nPatch*2]);
```

Plot the macroscale predictions to draw Figure 3, in groups of five in a plot.

```
figure(1),clf
k=length(ts); ls=nan(5,ceil(k/5)); ls(1:k)=1:k;
for k=1:size(ls,2)
  subplot(size(ls,2),1,k)
  plot(x(:),us(:,ls(:,k)),'.')
  ylabel('u(x,t)')
  legend(num2str(ts(ls(:,k))'))
end
xlabel('space x')
%matlab2tikz('pi1Test1u.ltx','noSize',true)
%print('-depsc2','pi1PatchesU')
```

Also plot a surface of the microscale bursts as shown in Figure 4.

```
tss(end)=nan; %omit end time-point
figure(2),clf
for k=1:2, subplot(2,2,k)
  surf(tss,x(:),uss,'EdgeColor','none')
  ylabel('x'),xlabel('t'),zlabel('u(x,t)')
  axis tight, view(121-4*k,45)
end
%print('-depsc2','pi1PatchesMicro')
```

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

```
end
```

Discretisation of Burgers PDE in coupled patches Code the simple centred difference discretisation of the nonlinear Burgers' PDE, 2π-periodic in

space.

```
function ut=dudt(t,u)
global dx DX ratio j jp jm i I
nPatch=j(end);
u=reshape(u,[],nPatch);
```

Compute differences of the mid-patch values.

```
dmu=(u(I,jp)-u(I,jm))/2; % \mu\delta
ddu=(u(I,jp)-2*u(I,j)+u(I,jm)); % \delta^2
dddmu=dmu(jp)-2*dmu(j)+dmu(jm);
ddddu=ddu(jp)-2*ddu(j)+ddu(jm);
```

Use these differences to interpolate fluxes on the patch boundaries and hence set the edge values on the patch (Roberts & Kevrekidis 2007).

```
u(end,j)=u(end-1,j)+(dx/DX)*(dmu+ratio*ddu ...
        -(dddmu*(1/6-ratio^2/2)+ddddu*ratio*(1/12-ratio^2/6)));
u(1,j)=u(2,j)      -(dx/DX)*(dmu-ratio*ddu ...
        -(dddmu*(1/6-ratio^2/2)-ddddu*ratio*(1/12-ratio^2/6)));
```

Code Burgers' PDE in the interior of every patch.

```
ut=(u(i+1,j)-2*u(i,j)+u(i-1,j))/dx^2 ...
   -30*u(i,j).*(u(i+1,j)-u(i-1,j))/(2*dx);
ut=reshape([nan(1,nPatch);ut;nan(1,nPatch)] ,[],1);
end
```

4 projIntDMDExplore1: explore effect of varying parameters

Section contents

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.

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

```
function projIntDMDExplore1
n=9
ts=0:2:6
dt=2/n^2
ICNoise=0.3
```

Very strangely, the results from Matlab and Octave are different for the zero noise case!???? It should be deterministic. Significantly different in that Matlab fails more often.

Figure 5: errors in the projective integration of the nonlinear diffusion PDE *from initial conditions that are on the slow manifold. Plotted are stereo views of isosurfaces in parameter space: the first row is after the first projective step; the second row after the second step.*

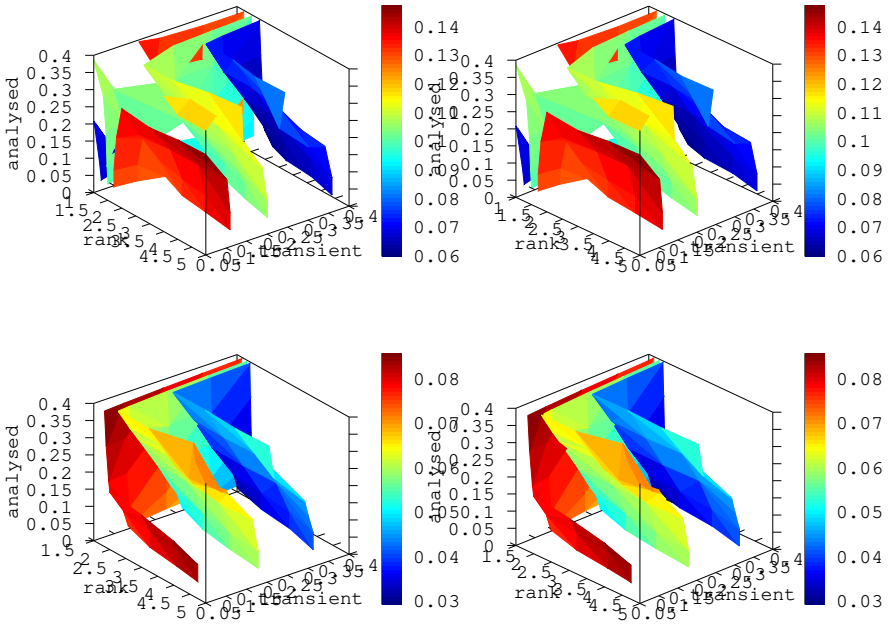


Figure 6: errors in the projective integration of the nonlinear diffusion PDE *from initial conditions with noise* `0.3*rand`*. Plotted are stereo views of isosurfaces in parameter space: the first row is after the first projective step; the second row after the second step.*

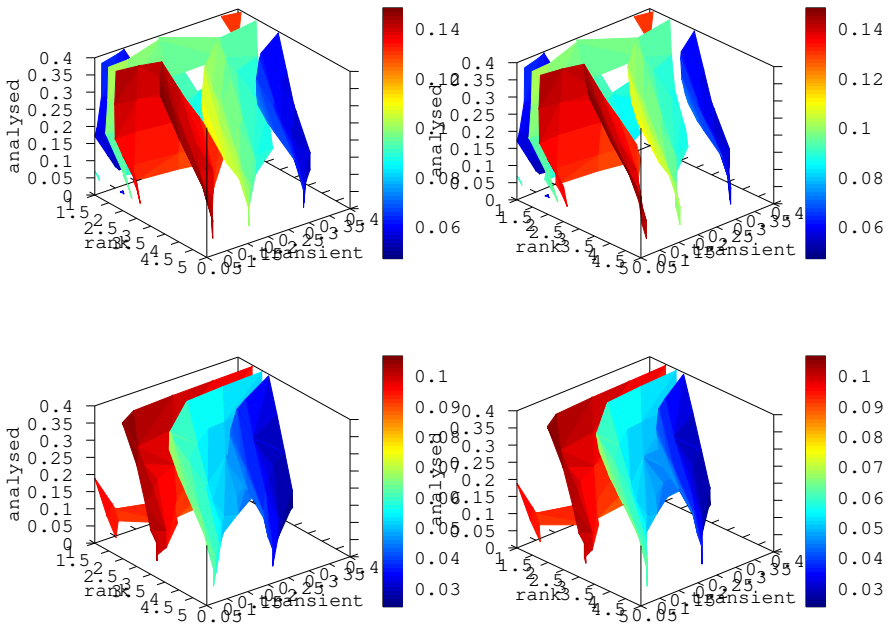


Figure 5 shows the parameter variations when the system is already on the slow manifold. The picture after two time-steps, bottom row, appears clearer than for one time-step. The errors do not vary with rank provided that it is ≥ 2. There is only a very weak dependence upon the length of the burst being analysed—and that could be due to reduction in the gap. There is a weak dependence upon the transient-time, but only by a factor of two across the domain considered.

With the addition of a noisy initial conditions, Figure 6, the rank has an effect, and the transient-time appears to be a slightly stronger influence. I suspect this means that we need to allow the initial burst to have a longer transient time than subsequent bursts. Initial conditions may typically need a longer 'healing' time. Thus code an extra `timeStep` parameter.

Set the initial condition to parabola or some skewed random positive values. Without noise this initial condition is already on the slow manifold so only little reason for transient time.

```
56 x=linspace(-1,1,n+2)';
57 u0=(0.5+ICNoise*rand(n+2,1)).*(1-x.^2);
```

First find a reference solution of the microscale dynamics over all time.

```
64 [Us,Uss,Tss]=projIntDMD(@dudt,u0,ts,2,dt,[0 2]);
```

Set up various combinations of parameters.

```
[rank,trant,slowt]=meshgrid(1:5,[1 2 4 6 8]*0.05,[2 4 8 12 16]*dt);
ps=[rank(:) trant(:) slowt(:)];
```

Projectively integrate in time with various parameters.

```
errs=[]; relerrs=[];
for p=ps'
[us,uss,tss]=projIntDMD(@dudt,u0,ts,p(1),dt,p(2:3));
```

Plot the macroscale predictions

```
if 0
  clf,plot(x,Us,'o-',x,us,'x--')
  xlabel('space x'),ylabel('u(x,t)')
  pause(0.01)
end
```

Accumulate errors as function of time.

```
err=sqrt(sum((us-Us).^2))
errs=[errs;err];
relerrs=[relerrs;err./sqrt(sum(Us.^2))];
```

End the loop over parameters.

```
end
```

Stereo view of isosurfaces of errors after both one and two time-steps. The three surfaces are the quartiles of the errors, coloured accordingly, but with a little extra colour from position for clarity.

```
clf()
vals=nan(size(rank));
for k=1:2
  vals(:)=errs(:,k+1);
  q=prctile(vals(:),(0:4)*25)
  for j=1:2, subplot(2,2,j+2*(k-1)),hold on
    for i=2:4 % draw three quartiles
      isosurface(rank,trant,slowt,vals,q(i) ...
      ,q(i)+0.03*(rank/10-trant+slowt))
    end,hold off
    xlabel('rank'),ylabel('transient'),zlabel('analysed')
    colorbar
    set(gca,'view',[57-j*5,30])
  end%j
end%k
```

Save to file

```
print('-depsc2',['explore1icn' num2str(ICNoise*10)])
```

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

```
end
```

The nonlinear PDE discretisation Code the simple centred difference discretisation of the nonlinear diffusion PDE with constant (usually zero) boundary values.

```
155 function ut=dudt(t,u)
156 n=length(u);
157 dx=2/(n-1);
158 j=2:n-1;
159 ut=[0
160     u(j).*(u(j+1)-2*u(j)+u(j-1))/dx^2
161     0];
162 end
```

5 projIntDMDExplore2: explore effect of varying parameters

Section contents

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.

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

```
20 function projIntDMDExplore2
21 n=9
22 dt=2/n^2
23 ICNoise=0
```

Set micro-simulation parameters. Rank two is fine when starting on the slow manifold. Choose middle of the road transient and analysed time.

```
32 rank=2
33 timeSteps=[0.2 0.2]
```

Try integrating with macro time-steps up to this sort of magnitude.

```
39 Ttot=9
```

Set the initial condition to parabola or some skewed random positive values. Without noise this initial condition is already on the slow manifold so only little reason for transient time.

```
51 x=linspace(-1,1,n+2)';
52 u0=(0.5+ICNoise*rand(n+2,1)).*(1-x.^2);
```

First find a reference solution of the microscale dynamics over all time, here stored in `Uss`.

```
59 [Us,Uss,Tss]=projIntDMD(@dudt,u0,[0 Ttot],2,dt,[0 Ttot]);
```

Projectively integrate two steps in time with various parameters. But remember that `projIntDMD` rounds `timeSteps` etc to nearest multiple of `dt`, so some of the following is a little dodgy but should not matter for overall trend.

```
Dts=0.1*[1 2 4 6 10 16 26]
errs=[]; relerrs=[]; DTs=[];
for p=Dts
[~,j]=min(abs(sum(timeSteps)+p-Tss))
ts=Tss(j)*(0:2)
js=1+(j-1)*(0:2);
[us,uss,tss]=projIntDMD(@dudt,u0,ts,rank,dt,timeSteps);
```

Plot the macroscale predictions

```
if 1
  clf,plot(x,Uss(:,js),'o-',x,us,'x--')
  xlabel('space x'),ylabel('u(x,t)')
  pause(0.01)
end
```

Accumulate errors as function of time.

```
err=sqrt(sum((us-Uss(:,js)).^2))
errs=[errs;err];
relerrs=[relerrs;err./sqrt(sum(Uss(:,js).^2))];
```

End the loop over parameters.

```
end
```

Plot errors

```
loglog(Dts,errs(:,2:3),'o:')
xlabel('projective time-step')
ylabel('steps error')
legend('one','two')
grid
matlab2tikz('pi1x2.ltx')
```

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

```
end
```

The nonlinear PDE discretisation Code the simple centred difference discretisation of the nonlinear diffusion PDE with constant (usually zero) boundary values.

```
function ut=dudt(t,u)
n=length(u);
dx=2/(n-1);
j=2:n-1;
ut=[0
    u(j).*(u(j+1)-2*u(j)+u(j-1))/dx^2
```

```
137       0];
138   end
```

References

Kutz, J. N., Brunton, S. L., Brunton, B. W. & Proctor, J. L. (2016), *Dynamic Mode Decomposition: Data-driven Modeling of Complex Systems*, number 149 *in* 'Other titles in applied mathematics', SIAM, Philadelphia.

Roberts, A. J. & Kevrekidis, I. G. (2007), 'General tooth boundary conditions for equation free modelling', *SIAM J. Scientific Computing* **29**(4), 1495–1510.