First draft Moving Mesh documentation

AJR

August 20, 2021

DRAFT

1 `mmBurgersExample`: example of moving patches for Burgers' PDE

The code here shows one way to get started: a user's script may have the following three steps (left-right arrows denote function recursion).

1. configPatches1
2. ode15s integrator ↔ mmPatchSmooth1 ↔ user's PDE
3. process results

The simulation seems perfectly happy for the patches move so that they overlap in the shock! and then separate again as the shock decays.

Establish global patch data struct to point to and interface with a function coding Burgers' PDE: to be solved on 1-periodic domain, with fifteen patches, spectral interpolation couples the patches, each patch of half-size ratio 0.2, and with five microscale points forming each patch.

```
global patches
patches = configPatches1(@mmBurgersPDE,[0 1], nan, 15, 0, 0.2, 5 ...
    ,'EdgyInt',true);
patches.mmtime=0.1;
patches.Xlim=[0 1];
```

Set usual sinusoidal initial condition. Add some microscale randomness that decays within time of 0.01, but also seeds slight macroscale variations.

```
u0 = sin(2*pi*patches.x)+0.05*randn(size(patches.x));
N = size(patches.x,4)
D0 = zeros(N,1);
```

Simulate in time using a standard stiff integrator and the interface function `mmPatchsmooth1()` (Section 2).

```
tic
[ts,us] = ode15s( @mmPatchSmooth1,linspace(0,0.8),[D0;u0(:)]);
cpuTime=toc
```

Plots Choose whether to save some plots, or not.

```
global OurCf2eps
OurCf2eps=false;
```

Plot the movement of the mesh, the centre of each patch, as a function of time: spatial domain horizontal, and time vertical.

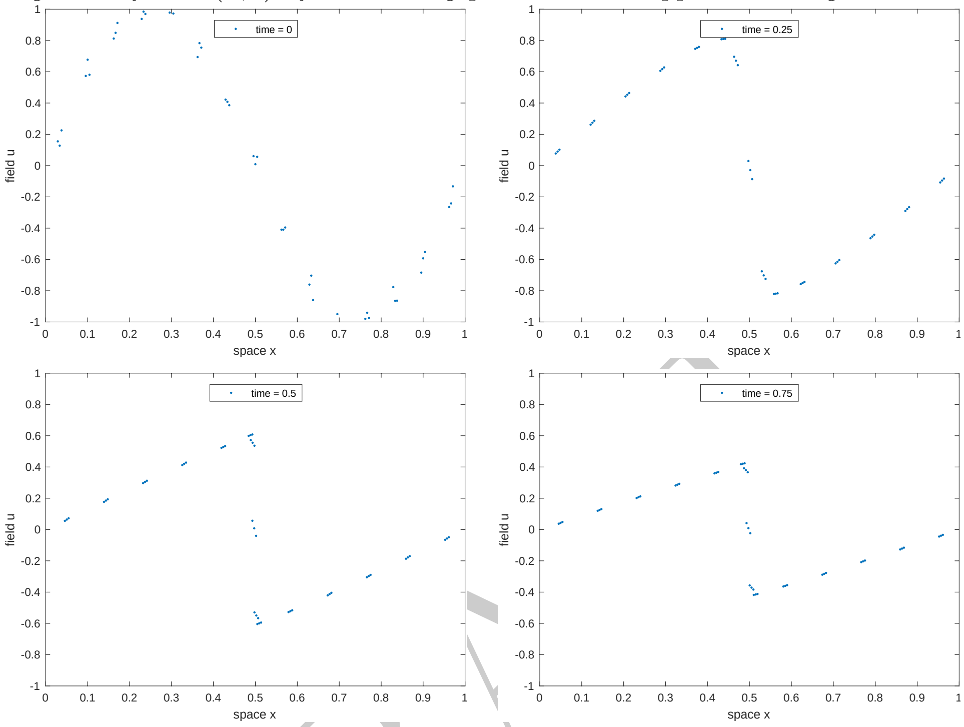
```
figure(1),clf
Ds=us(:,1:N);
Xs=shiftdim(mean(patches.x),2);
plot(Xs+Ds,ts), ylabel('time t'),xlabel('space x')
title('Burgers PDE: patch locations over time')
```

Animate the simulation using only the microscale values interior to the patches: set x-edges to `nan` to leave the gaps. Figure 1 illustrates an example simulation in time generated by the patch scheme applied to Burgers' PDE.

```
us=us(:,N+1:end).';
us(abs(us)>2)=nan;
patches.x([1 end],:,:,:)=nan;
%% section break to ease rerun of animation
figure(2),clf
for i=1:nTime
  xs=squeeze(patches.x)+Ds(i,:);
  if i==1,  hpts=plot(xs(:),us(:,i),'.');
      ylabel('field u'), xlabel('space x')
      axis([0 1 -1 1])
  else  set(hpts,'XData',xs(:),'YData',us(:,i));
  end
  legend(['time = ' num2str(ts(i),2)],'Location','north')
  if rem(i,31)==1, ifOurCf2eps([mfilename num2str(i)]), end
  pause(0.04)
end
%surf(ts,patches.x(:),us)
%view(60,40), colormap(0.8*hsv)
%xlabel('time t'), ylabel('space x'), zlabel('u(x,t)')
%ifOurCf2eps(mfilename)
```

DRAFT

Figure 1: field $u(x,t)$ of the moving patch scheme applied to Burgers' PDE.



2 mmPatchSmooth1(): interface 1D space of moving patches to time integrators

To simulate in time with moving 1D spatial patches we need to interface a user's time derivative function with time integration routines such as `ode23` or `PIRK2`. This function `mmPatchSmooth1()` provides an interface. Patch edge values are determined by macroscale interpolation of the patch-centre or edge values. Microscale heterogeneous systems may be accurately simulated with this function via appropriate interpolation. Communicate patch-design variables (**??**) either via the global struct `patches` or via an optional third argument (except that this last is required for parallel computing of `spmd`).

```
function dudt = mmPatchSmooth1(t,u,patches)
if nargin<3, global patches, end
```

Input

DRAFT

- `u` is a vector of length `nPatch`$+$`nSubP`$\cdot$`nVars`$\cdot$`nEnsem`$\cdot$`nPatch` where there are `nVars`$\cdot$`nEnsem` field values at each of the points in the `nSubP`$\times$`nPatch` grid, and because of the moving mesh there are an additional `nPatch` patch displacement values at its start.

- `t` is the current time to be passed to the user's time derivative function.

- `patches` a struct set by `configPatches1()` with the following information used here.

  - `.fun` is the name of the user's function `fun(t,u,V,D,patches)` that computes the time derivatives on the patchy lattice, where the jth patch moves at velocity V_j and at current time is displaced D_j from the fixed reference position in `.x`. The array `u` has size `nSubP` $\times$ `nVars` $\times$ `nEnsem` $\times$ `nPatch`. Time derivatives should be computed into the same sized array, then herein the patch edge values are overwritten by zeros.
  - `.x` is `nSubP` $\times 1 \times 1 \times$ `nPatch` array of the spatial locations x_i of the microscale grid points in every patch. Currently it *must* be an equi-spaced lattice on both macro- and microscales ??

Output

- `dudt` is a vector/array of of time derivatives, but with patch edge-values set to zero. It is of total length `nPatch` $+$ `nSubP` $\cdot$ `nVars` $\cdot$ `nEnsem` $\cdot$ `nPatch` and the same dimensions as `u`.

Extract the `nPatch` displacement values from the start of the vectors of evolving variables. Reshape the rest as the fields `u` in a 4D-array, and sets the edge values from macroscale interpolation of centre-patch values. **??** describes `patchEdgeInt1()`.

```
N = size(patches.x,4);
D = reshape(u(1:N),[1 1 1 N]);
u = patchEdgeInt1(u(N+1:end),patches);
```

Moving mesh velocity Developing from standard moving meshes for PDEs (Budd et al. 2009, Huang & Russell 2010, e.g.), we follow Maclean et al. (2021). There exists a set of macro-scale mesh points $X_j(t) := X_j^0 + D_j(t)$ (at the centre) of each patch with associated field values, say $U_j(t) := \overline{u_{ij}(t)}$.

```
X = mean(patches.x,1)+D;
U = mean(u,1);
```

DRAFT

Then for every patch j we set $H_j := X_{j+1} - X_j$ for periodic patch indices j

```
j=1:N; jp=[2:N 1]; jm=[N 1:N-1];
H = X(:,:,:,jp)-X(:,:,:,j);
H(N) = H(N)+diff(patches.Xlim);
```

we discretise a moving mesh PDE for node locations X_j with field values U_j via the second derivative estimate

$$U''_j := \frac{2}{H_j + H_{j-1}} \left[\frac{U_{j+1} - U_j}{H_j} - \frac{U_j - U_{j-1}}{H_{j-1}} \right], \tag{1a}$$

and its norm over all variables and ensembles (arbitrarily?? chose the mean square norm here).

```
U2 = ( (U(:,:,:,jp)-U(:,:,:,j))./H(:,:,:,j) ...
      -(U(:,:,:,j)-U(:,:,:,jm))./H(:,:,:,jm) ...
     )*2./(H(:,:,:,j)+H(:,:,:,jm));
U2 = squeeze( mean(mean( abs(U2).^2 ,2),3) );
H = squeeze(H);
```

Having squeezed out all microscale information, the coefficient

$$\alpha := \max\left\{ 1 , \left[\frac{1}{b-a} \sum_j H_{j-1} \frac{1}{2} \left({U''_j}^{2/3} + {U''_{j-1}}^{2/3} \right) \right]^3 \right\} \tag{1b}$$

```
alpha = sum( H(jm).*( U2(j).^(1/3)+U2(jm).^(1/3) ))/sum(H);
alpha = max(1,alpha^3);
```

Then the importance function

$$\rho_j := \left(1 + \frac{1}{\alpha} {U''_j}^2 \right)^{1/3}, \tag{1c}$$

```
rho = ( 1+U2/alpha ).^(1/3);
```

For every patch, we move all micro-grid points according to the following velocity of the notional macro-scale node of that patch:

$$V_j := \frac{dX_j}{dt} = \frac{(N-1)^2}{2\rho_j \tau} \left[(\rho_{j+1} + \rho_j) H_j - (\rho_j + \rho_{j-1}) H_{j-1} \right]. \tag{1d}$$

```
V = nan+D; % allocate storage
V(:) = ( (rho(jp)+rho(j)).*H(j) -(rho(j)+rho(jm)).*H(jm) ) ...
    ./rho(j) *((N-1)^2/2/patches.mmtime);
```

DRAFT

Evaluate system differential equation Ask the user function for the advected time derivatives on the moving patches, overwrite its edge values with the dummy value of zero (since `ode15s` chokes on NaNs), then return to the user/integrator as a vector.

```
dudt=patches.fun(t,u,V,D,patches);
dudt([1 end],:,:,:) = 0;
dudt=[V(:); dudt(:)];
```

Fin.

2.1 `mmBurgersPDE()`: Burgers PDE inside a moving mesh of patches

For the evolving scalar field $u(t,x)$, we code a microscale discretisation of Burgers' PDE $u_t = \epsilon u_{xx} - uu_x$, for say $\epsilon = 0.02$, when the patches of microscale lattice move with various velocities V.

```
function ut = mmBurgersPDE(t,u,V,D,patches)
epsilon = 0.02;
```

Generic input/output variables

- `t` (scalar) current time—not used here as the PDE has no explicit time dependence (autonomous).
- `u` ($n \times 1 \times 1 \times N$) field values on the patches of microscale lattice.
- `V` ($1 \times 1 \times 1 \times N$) moving velocity of the jth patch.
- `D` ($1 \times 1 \times 1 \times N$) displacement of the jth patch from the fixed spatial positions stored in `patches.x`—not used here as the PDE has no explicit space dependence (homogeneous).
- `patches` struct of patch configuration information.
- `ut` ($n \times 1 \times 1 \times N$) output computed values of the time derivatives Du/Dt on the patches of microscale lattice.

Here there is only one field variable, and one in the ensemble, so for simpler coding of the PDE we squeeze them out (no need to reshape when via `mmPatchSmooth1`).

```
  u=squeeze(u);     % omit singleton dimensions
  V=shiftdim(V,2);  % omit two singleton dimens
```

DRAFT

Burgers PDE In terms of the moving derivative $Du/Dt := u_t + Vu_x$ the PDE becomes $Du/Dt = \epsilon u_{xx} + (V-u)u_x$. So code for every patch that $\dot u_{ij} = \frac{\epsilon}{h^2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) + (V_j - u_{ij})\frac{1}{2h}(u_{i+1,j} - u_{i-1,j})$ at all interior lattice points.

```
  dx=diff(patches.x(1:2)); % microscale spacing
  i=2:size(u,1)-1;  % interior points in patches
  ut=nan+u;         % preallocate output array
  ut(i,:) = epsilon*diff(u,2)/dx^2 ...
      +(V-u(i,:)).*(u(i+1,:)-u(i-1,:))/(2*dx);
end
```

References

Budd, C. J., Huang, W. & Russell, R. D. (2009), ‘Adaptivity with moving grids’, *Acta Numerica* **18**, 111–241.

Huang, W. & Russell, R. D. (2010), *Adaptive moving mesh methods*, Vol. 174, Springer Science & Business Media.

Maclean, J., Bunder, J. E., Kevrekidis, I. G. & Roberts, A. J. (2021), Adaptively detect and accurately resolve macro-scale shocks in an efficient equation-free multiscale simulation, Technical report, University of Adelaide.

DRAFT