Equation-free computational homogenisation with Dirichlet boundaries

A. J. Roberts*

November 18, 2022

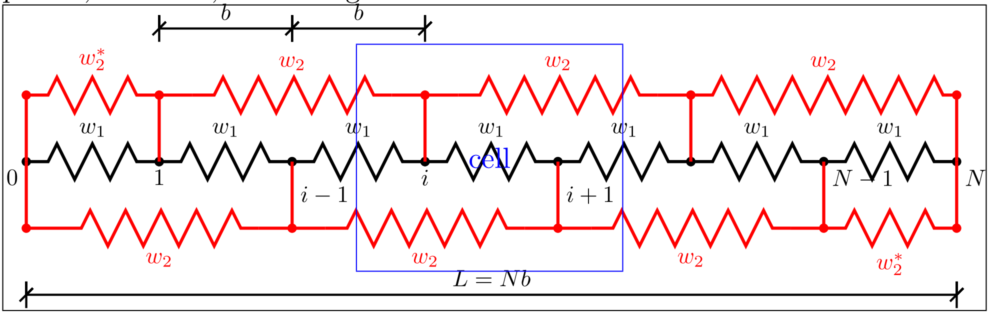
Contents

1 `Combescure2022`: example of a 1D heterogeneous toy elasticity by simulation on small patches

Plot an example simulation in time generated by the patch scheme applied to macroscale toy elasticity through a medium with microscale heterogeneity.

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

Figure 1: 1D arrangement of non-linear springs with connections to (a) next-to-neighbor node (Combescure 2022, Fig. 3(a)). The blue box is one cell of one period, width $2b$, containing an odd and an even i.



Suppose the spatial microscale lattice is at rest at points x_i, with constant spacing b (Figure 1). With displacement variables $u_i(t)$, simulate the microscale lattice toy elasticity system with 2-periodicity: for $p = 1, 2$ (respectively black and red in Figure 1) and for every i,

$$\epsilon_i^p := \frac{1}{pb}(u_{i+p/2} - u_{i-p/2}), \quad \sigma_i^p := w_p'(\epsilon_i^p), \quad \frac{\partial^2 u_i}{\partial t^2} = \sum_{p=1}^{2} \frac{1}{pb?}(\sigma_{i+p/2}^p - \sigma_{i-p/2}^p). \tag{1}$$

The system has a microscale heterogeneity via the different functions $w_p'(\epsilon) := \epsilon - M_p\epsilon^3 + \epsilon^5$ (Combescure 2022, §4):

- microscale instability with $M_1 := 2$ and $M_2 := 1$; and
- macroscale instability with $M_1 := -1$ and $M_2 := 3$.

1.1 Simulate heterogeneous toy elasticity systems

Set some physical parameters.

```
clear all
global b M vis i0 iN dFdt
b = 1  % separation of lattice points
N = 40 % # lattice steps in L
L = b*N
M = [0 0] % no cubic spring terms
M = [2 1] % small scale instability
%M = [-1 3] % large scale instability
```

```
dFdt = 0.02
vis = 0
tol = 1e-9;
```

Patch parameters: here `nSubP` is the number of cells, so `lPatch` is the distance from leftmost odd/even points to the rightmost odd/even points, respectively.

```
edgyInt = true
nSubP = 6, nP = 5 % gives ratio=1 for full-domain
%nSubP = 4, nP = 3
H=L/nP
if edgyInt, ratio=2*b*(nSubP-2)/H, end
%nP4ratio1=L/(2*b*(nSubP-2))
```

Establish the global data struct `patches` for the microscale heterogeneous lattice toy elasticity system (1). Solved on $2L$-periodic domain, with `2*nP` patches, and spectral interpolation to provide the edge-values of the inter-patch coupling conditions.

```
global patches
configPatches1(@heteroToyE,[0 2*L],nan,2*nP ...
    ,0,ratio,nSubP,'EdgyInt',edgyInt);
patches.x = patches.x-L+H/2;% shift so [0,L] is 2nd half of patches
%xGrid=squeeze(patches.x) % optionally disp the spatial grid
assert(abs(2*b-diff(patches.x(1:2)))<tol,'sub-patch grid config error
xx = patches.x+[-1 1]*b/2; % staggered sub-cell positions
```

1.2 Eigenvalues of the Jacobian

Set zero to be the reference equilibrium in this linear problem. Put NaNs on the patch-edges.

```
if 0
u0 = [ 0*xx 0*xx ];
u0([1 end],:,:,:)=nan;
i=find(~isnan(u0));
nJac=length(i)
```

Remove boundary conditions.

```
i0=[]; iN=[];
```

Construct the Jacobian column-wise from the transform of a complete set of unit basis vectors (as this is linear problem at the moment).

```
Jac=nan(nJac);
for j=1:nJac
  uj=u0; uj(i(j))=1;
  dujdt=patchSys1(-1,uj);
  Jac(:,j)=dujdt(i);
end
Jac(abs(Jac)<tol)=0;
figure(3),clf,spy(Jac)
```

Find eigenvalues

```
[evecs,evals]=eig(Jac);
evals=diag(evals);
[~,j]=sort( -real(evals)+0.0001*abs(imag(evals)) );
evals=evals(j);
evecs=evecs(:,j);
leadingEvals=evals(1:18)'
```

Plot spectrum

```
   handle = plot(real(evals),imag(evals),'.');
   xlabel('real-part'), ylabel('imag-part')
   quasiLogAxes(handle,0.1,1);
   drawnow
end%if compute eigenvalues
```

Simulate Set the initial conditions of a simulation. I choose to store odd i in `u((i+1)/2,1,:)` and even i in `u(i/2,2,:)`, that is, array

$$\mathtt{u} = \begin{bmatrix} u_1 & u_2 \\ u_3 & u_4 \\ u_5 & u_6 \\ \vdots & \vdots \end{bmatrix}.$$

```
u0 = 0*[ sin(pi/L*xx)  -0.14*cos(pi/L*xx) ];
u0 = u0+0.01*( rand(size(u0))-0.5 );
```

But, impose $u_i = 0$ at $x = 0$ which here I translate to mean that $u_i = \dot u_i = 0$ for both $x_i = \pm b/2$. Slightly different to the left-end of Figure 1, but should be near enough. Here find both $u, \dot u$ locations.

```
i0=find(abs([xx xx])<0.6*b);
u(i0)=0;
```

Apply a set force at material originally at $x = L$, so start with $u_i = \dot{u}_i = 0$ for both $x_i = L \pm b/2$. Subsequently apply an additional and increasing compression force on the points initially at $x = L$. Hmmm: but that is not quite isolating the two sides of $x = L$??

```
iN=find(abs([xx xx]-L)<0.6*b)
u(iN)=0;
```

Integrate some time using standard integrator.

```
tic
[ts,ust] = ode23(@patchSys1, 60*linspace(0,1,31), u0(:));
cpuIntegrateTime = toc
```

Plot space-time surface of the simulation We want to see the edge values of the patches, so interpolate and then adjoin a row of `nans` in between patches. Because of the odd/even storage we need to do a lot of permuting and reshaping.

```
xs = reshape( permute( xx ,[2 1 3 4]), 2*nSubP,2*nP);
xs(end+1,:) = nan;
uvs = reshape( permute( reshape(ust ...
      ,length(ts),nSubP,4,1,2*nP) ,[2 3 1 4 5]) ,nSubP,[],1,2*nP);
uvs = reshape( patchEdgeInt1(uvs) ,nSubP,4,[],2*nP);
us = reshape( permute( uvs(:,1:2,:,:) ...
     ,[2 1 4 3]) ,2*nSubP,2*nP,[]);
us(end+1,:,:) = nan;
us = reshape(us,[],length(ts));
vs = reshape( permute( uvs(:,3:4,:,:) ...
     ,[2 1 4 3]) ,2*nSubP,2*nP,[]);
vs(end+1,:,:) = nan;
vs = reshape(vs,[],length(ts));
```

Plot evolving function

```
figure(1),clf()
plot(xs(:),vs)
xlabel('space x')
%ylabel('displacement u')
ylabel('velocity v')
legend(num2str(ts))
```

Plot a space-time surface of field values over the macroscale duration of the simulation.

```
  figure(2), clf()
  mesh(ts,xs(:),us)
  view(60,40), colormap(0.8*hsv)
  xlabel('time t'), ylabel('space x'), zlabel('u(x,t)')
  title(['patch ratio r = ' num2str(ratio)])
  drawnow
  figure(3), clf()
  mesh(ts,xs(:),vs)
  view(60,40), colormap(0.8*hsv)
  xlabel('time t'), ylabel('space x'), zlabel('v(x,t)')
  title(['patch ratio r = ' num2str(ratio)])
  drawnow
```

1.3 `heteroToyE()`: forced heterogeneous toy elasticity

This function codes the lattice heterogeneous toy elasticity inside the patches. Computes the time derivative at each point in the interior of a patch, output in `ut`.

```
function uvt = heteroToyE(t,uv,patches)
  global b M vis i0 iN dFdt
```

Separate state vector into displacement and velocity fields.

```
  u=uv(:,1:2,:,:); v=uv(:,3:4,:,:); % separate u and v=du/dt
```

Compute the two different strain fields, and also a first derivative for some optional viscosity.

```
  eps2 = diff(u)/(2*b);
  eps1 = [u(:,2,:,:)-u(:,1,:,:) u([2:end 1],1,:,:)-u(:,2,:,:)]/b;
  eps1(end,2,:,:)=nan; % as this value is fake
  vx1  = [v(:,2,:,:)-v(:,1,:,:) v([2:end 1],1,:,:)-v(:,2,:,:)]/b;
  vx1(end,2,:,:)=nan; % as this value is fake
```

Set corresponding nonlinear stresses

```
  sig2 = eps2-M(2)*eps2.^3+eps2.^5;
  sig1 = eps1-M(1)*eps1.^3+eps1.^5;
```

Preallocate output array, and fill in time derivatives of displacement and velocity, from velocity and gradient of stresses, respectively.

```
uvt = nan+uv;           % preallocate output array
i=2:size(uv,1)-1;
% rate of change of position
uvt(i,1:2,:,:) = v(i,:,:,:);
% rate of change of velocity +some artificial viscosity??
uvt(i,3:4,:,:) = diff(sig2) ...
  +[ sig1(i,1,:,:)-sig1(i-1,2,:,:)  diff(sig1(i,:,:,:),1,2)] ...
+vis*[ vx1(i,1,:,:)-vx1(i-1,2,:,:)  diff(vx1(i,:,:,:),1,2) ];
```

Maintain boundary value of $u_i, \dot u_i$ by setting them both to be constant in time, for both $x_i = \pm b/2$. If `i0` is empty, then no boundary condition is set.

```
uvt(i0)=0;
uvt(iN(3:4))=uvt(iN(3:4))-dFdt*t;
end% function
```

2 `Eckhardt221004536`: example of a 1D heterogeneous diffusion by simulation on small patches

Plot an example simulation in time generated by the patch scheme applied to macroscale forced diffusion through a medium with microscale heterogeneity.

Suppose the spatial microscale lattice is at points x_i, with constant spacing dx. With dependent variables $u_i(t)$, simulate the microscale lattice forced diffusion system

$$\frac{\partial u_i}{\partial t} = \frac{1}{dx^2}\delta[a_{i-1/2}\delta u_i] + f_i(t), \tag{2}$$

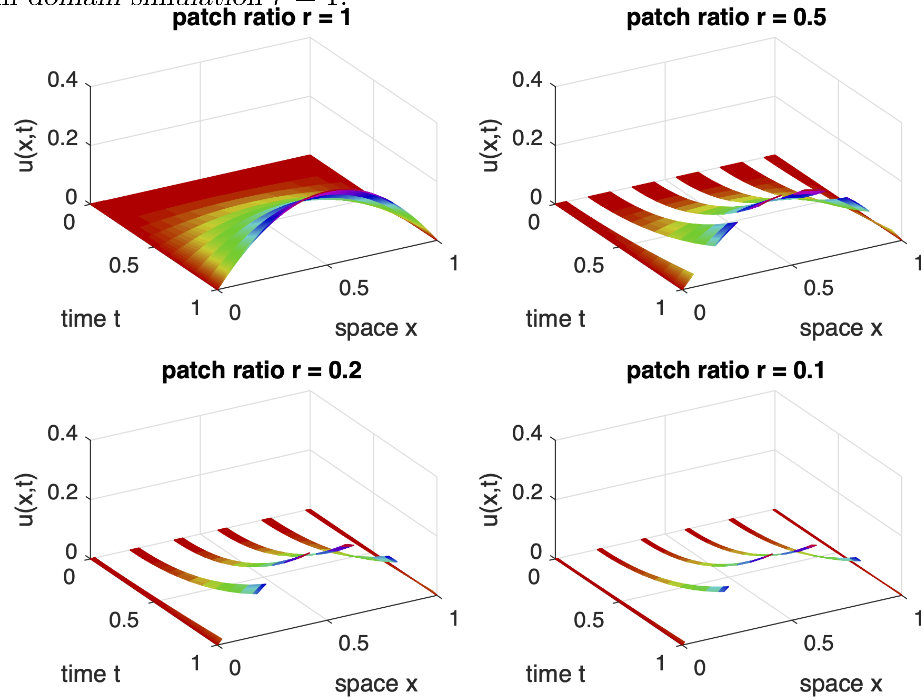
in terms of the centred difference operator δ. The system has a microscale heterogeneity via the coefficients $a_{i+1/2}$ which has some given known periodicity ϵ.

Here use period $\epsilon = 1/200$ (so that computation completes in seconds). The patch scheme computes only on a fraction r of the spatial domain, see Figure 2. Compute *errors* as the maximum difference (at time $t = 1$) between the patch scheme prediction and a full domain simulation of the same underlying spatial discretisation (which here has space step 1/1200).

patch ratio r	0.5	0.2	0.1	0.05
sine-forcing error	$0.5e{-}8$	$1.4e{-}8$	$1.8e{-}8$	$2.1e{-}8$
Eckhardt-forcing error	0.0018	0.0038	0.0046	0.0051

The smooth sine-forcing leads to errors that appear due to the integration tolerance of `ode15s`. The Eckhardt-forcing errors are then viewed as either due to boundary layers next to the Dirichlet boundaries, or equivalently due to

Figure 2: diffusion field $u(x,t)$ of the patch scheme applied to the forced heterogeneous diffusive (2). Simulates for various patch ratios r including the full-domain simulation $r = 1$.



the lack of smoothness in the odd-periodic extensions of the forcing required to preserve the Dirichlet conditions.

2.1 Simulate heterogeneous diffusion systems

First establish the microscale heterogeneity has micro-period `mPeriod` on the lattice, and coefficients to match Eckhardt2210.04536 §6.2.1. Set the phase of the heterogeneity so that each patch centre is a point of symmetry of the diffusivity. Then the heterogeneity is repeated to fill each patch. If an odd number of odd-periods in a patch, then the centre patch is a grid point of the field u, otherwise the centre patch is at a half-grid point.

```
mPeriod = 6
y = linspace(0,1,mPeriod+1)';
a = 1./(2-cos(2*pi*y(1:mPeriod)))
global microTimePeriod; microTimePeriod=0;
```

Set the periodicity, via integer $1/\epsilon$, and other parameters.

```
rEpsilon = 200
dx = 1/(mPeriod*rEpsilon)
nP = 5 % the number of patches on [0 1]
maxPeriodsPatch = rEpsilon/nP
tol=1e-9;
```

Loop to explore errors on various sized patches.

```
nPPs = maxPeriodsPatch./[1 2 5 10 20 50 100];
nPPs = nPPs(nPPs>1)
Us=[]; Uerr=0;% for storing results to compare
for iPP = 1:length(nPPs)
nPeriodsPatch = nPPs(iPP)
```

Establish the global data struct `patches` for the microscale heterogeneous lattice diffusion system (2) solved on 2-periodic domain, with `2*nP` patches, and spectral interpolation to provide the edge-values of the inter-patch coupling conditions[1]. Setting `patches.EdgyInt` true means the edge-values come from interpolating the opposite next-to-edge values of the patches (not the mid-patch values).

```
ratio = nPeriodsPatch/maxPeriodsPatch
nSubP = nPeriodsPatch*mPeriod+2
global patches
configPatches1(@heteroDiffF,[0 2],nan,2*nP ...
    ,0,ratio,nSubP,'EdgyInt',true,'hetCoeffs',a);
patches.x = patches.x-1+1/(2*nP);% shift so [0,1] is 2nd half of patc
%x=squeeze(patches.x) % optionally disp the spatial grid
assert(abs(dx-diff(patches.x(1:2)))<tol,'sub-patch-grid config error'
```

Set the forcing coefficients as the odd-periodic extensions, accounting for roundoff error in `f2`.

```
if 1 % odd-periodic extension of given forcing
patches.f1=2*( patches.x-sign(patches.x).*patches.x.^2 ...
              +(patches.x>1).*(patches.x-1).^2*2 );
patches.f2=2*0.5*sign(patches.x.*(1-patches.x)) ...
    .*(abs(patches.x.*(1-patches.x))>tol);
else% simple sine forcing give errors less than 2e-8
patches.f1=sin(pi*patches.x);
```

[1] Curiously, for low-order interpolation—less than order 8—the error for large patches is larger than that for small patches

```
patches.f2=pi/2*sin(pi*patches.x);
end%if
%f1=squeeze(patches.f1)% optionally disp spatial pattern f1
%f2=squeeze(patches.f2)% optionally disp spatial pattern f2
```

Simulate Set the initial conditions of a simulation to be zero. Integrate to time 1 using standard integrators.

```
u0 = 0*patches.x;
[ts,us] = ode15s(@patchSys1, [0 1], u0(:));
```

Plot space-time surface of the simulation We want to see the edge values of the patches, so adjoin a row of `nans` in between patches. For the field values (which are rows in `us`) we need to reshape, permute, interpolate to get edge values, pad with `nans`, and reshape again.

```
xs = squeeze(patches.x);
us = patchEdgeInt1( permute( reshape(us ...
    ,length(ts),nSubP,1,nPatch) ,[2 1 3 4]) );
us = squeeze(us);
xs(end+1,:) = nan;  us(end+1,:,:) = nan;
uss=reshape(permute(us,[1 3 2]),[],length(ts));
```

Test the error in BC is negligible, for both when micro-grid point on boundary and when micro-grid points straddle boundary.

```
i=[ max(find(xs<+tol)) min(find(xs>1-tol)) ];
j=[ min(find(xs>-tol)) max(find(xs<1+tol)) ];
maxBCerror=max(max( abs(uss(i,:)+uss(j,:))/2 ));
assert(maxBCerror<tol,'BC failure')
```

Plot a space-time surface of field values over the macroscale duration of the simulation.

```
if iPP<=4 % only draw four subplots
  i=j(1):j(2);
  figure(1), if iPP==1, clf(), end
  subplot(2,2,iPP)
  mesh(ts,xs(i),uss(i,:))
  view(60,40), colormap(0.8*hsv)
  xlabel('time t'), ylabel('space x'), zlabel('u(x,t)')
  title(['patch ratio r = ' num2str(ratio)])
  drawnow
end%if
```

At the end of the iPP-loop, store field from centre region of each patch for comparison.

```
i=nPeriodsPatch/2*mPeriod+1+(-mPeriod/2+1:mPeriod/2);
Us(:,:,iPP)=squeeze(us(i,end,:));
Xs=squeeze(patches.x(i,1,1,:));
if iPP>1
   assert(norm(Xs-Xsp)<tol,'sampling error in space')
   Uerr(iPP)=max(max(abs(squeeze(Us(:,:,iPP)-Us(:,:,1)))))
   end
Xsp=Xs;
end%for iPP
ifOurCf2eps(mfilename) %optionally save figure
```

2.2 `heteroDiffF()`: forced heterogeneous diffusion

This function codes the lattice heterogeneous diffusion inside the patches. Computes the time derivative at each point in the interior of a patch, output in `ut`. The column vector of diffusivities a_i has been stored in struct `patches.cs`, as has the array of forcing coefficients.

```
function ut = heteroDiffF(t,u,patches)
  global microTimePeriod
  dx = diff(patches.x(2:3));   % space step
  i = 2:size(u,1)-1;   % interior points in a patch
  ut = nan+u;          % preallocate output array
  if microTimePeriod>0 % optional time fluctuations
     at = cos(2*pi*t/microTimePeriod)/30;
  else at=0; end
  ut(i,:,:,:) = diff((patches.cs(:,1,:)+at).*diff(u))/dx^2 ...
      +patches.f2(i,:,:,:)*t^2+patches.f1(i,:,:,:)*t;
end% function
```

3 `piEckhart1`: example of 1D space-time heterogeneous diffusion via computational homogenisation with projective integration and small patches

Plot an example simulation in time generated by projective integration allied with the patch scheme applied to forced diffusion in a medium with microscale heterogeneity.

Suppose the spatial microscale lattice is at points x_i, with constant spacing dx. With dependent variables $u_i(t)$, simulate the microscale lattice forced diffusion system

$$\frac{\partial u_i}{\partial t} = \frac{1}{dx^2}\delta[a_{i-1/2}(t)\delta u_i] + f_i(t), \tag{3}$$

in terms of the centred difference operator δ. The system has a microscale heterogeneity via the coefficients $a_{i+1/2}$ which has given periodicity ϵ in space, and periodicity ϵ^2 in time.

The approximate homogenised PDE is $U_t = A_0U_{xx} + F$ with $U = 0$ at $x = 0, 1$. Its slowest mode is then $U = \sin(\pi x)e^{-A_0\pi^2 t}$. When $A_0 = 3.3524$ as in Eckhart then the rate of evolution is about 33 which is relatively fast on the simulation time-scale of $T = 1$. Let's slow down the dynamics by reducing diffusivities by a factor of 30, so effectively $A_0 \approx 0.1$ and $A_0\pi^2 \approx 1$.

Also, in the microscale fluctuations change the time variation to cosine, not its square (because I cannot see the point of squaring it!).

The highest wavenumber mode on the macro-grid of patches, spacing H, is the zig-zag mode on $\dot{U}_i = A_0(U_{I+1} - 2U_I + U_{I-1})/H^2 + F_I$ which evolves like $U_I = (-1)^I e^{-\alpha t}$ for the fastest 'slow rate' of $\alpha = 4A_0^2/H^2$. When $H = 0.2$ and $A_0 \approx 0.1$ this rate is $\alpha \approx 10$.

Here use period $\epsilon = 1/100$ (so that computation completes in seconds, and because we have slowed the dynamics by 30). The patch scheme computes only on a fraction r of the spatial domain. Projective integration computes only on a fraction of the time domain determined by the 'burst length'.

3.1 Simulate heterogeneous diffusion systems

First establish the microscale heterogeneity has micro-period `mPeriod` on the spatial lattice, and coefficients inspired by Eckhardt2210.04536 §6.2. Set the phase of the heterogeneity so that each patch centre is a point of symmetry of the diffusivity. Then the heterogeneity is repeated to fill each patch. If an odd number of odd-periods in a patch, then the centre patch is a grid point of the field u, otherwise the centre patch is at a half-grid point.

```
mPeriod = 6
y = linspace(0,1,mPeriod+1)';
a = ( 3+cos(2*pi*y(1:mPeriod)) )/30
A0 = 1/mean(1./a) % roughly the effective diffusivity
```

The microscale diffusivity has an additional additive component of $+\frac{1}{30}\cos(2\pi t/\epsilon^2)$ which is coded into time derivative routine via global `microTimePeriod`.

Set the periodicity, via integer $1/\epsilon$, and other parameters.

```
rEpsilon = 100
dx = 1/(mPeriod*rEpsilon)
nP = 5 % the number of patches on [0 1]
tol=1e-9;
nPeriodsPatch = 2
global microTimePeriod
microTimePeriod = 1/rEpsilon^2
```

Establish the global data struct `patches` for the microscale heterogeneous lattice diffusion system (3) solved on 2-periodic macroscale domain, with `2*nP` patches, and spectral interpolation to provide the edge-values of the inter-patch coupling conditions. Setting `patches.EdgyInt` true means the edge-values come from interpolating the opposite next-to-edge values of the patches (not the mid-patch values).

```
ratio = nPeriodsPatch/(rEpsilon/nP)
nSubP = nPeriodsPatch*mPeriod+2
global patches
configPatches1(@heteroDiffF,[0 2],nan,2*nP ...
    ,0,ratio,nSubP,'EdgyInt',true,'hetCoeffs',a);
patches.x = patches.x-1+1/(2*nP);% shift so [0,1] is 2nd half of patc
assert(abs(dx-diff(patches.x(1:2)))<tol,'sub-patch-grid config error'
```

Set the forcing coefficients as the odd-periodic extensions, accounting for roundoff error in `f2`.

```
if 1 % odd-periodic extension of given forcing
patches.f1=2*( patches.x-sign(patches.x).*patches.x.^2 ...
              +(patches.x>1).*(patches.x-1).^2*2 );
patches.f2=2*0.5*sign(patches.x.*(1-patches.x)) ...
    .*(abs(patches.x.*(1-patches.x))>tol);
else% simple sine forcing give errors less than ??
patches.f1=sin(pi*patches.x);
patches.f2=pi/2*sin(pi*patches.x);
end%if
%f1=squeeze(patches.f1)% optionally disp spatial pattern f1
%f2=squeeze(patches.f2)% optionally disp spatial pattern f2
```

Simulate Set the initial conditions of a simulation to be zero. Mark that edge of patches are not to be used in the projective extrapolation by setting initial values to NaN.

```
u0 = 0*patches.x;
u0([1 end],:) = nan;
```

Set the desired macro- and microscale time-steps over the time domain. The macroscale step is in proportion to the effective mean diffusion time on the macroscale, here $1/(A_0\pi^2) \approx 1$ so for macro-scale error less than 1% need $\Delta t < 0.24$, so use 0.1 say.

The burst time depends upon the sub-patch effective diffusion rate β where here rate $\beta \approx \pi^2 A_0/h^2 \approx 400$ for patch width $h = r/N \approx 0.02$: use the formula from the Manual, with 50% extra, and rounded to the nearest multiple of the time micro-periodicity.

```
ts = linspace(0,1,11)
beta = pi^2*A0/(ratio/nP)^2 % slowest rate of fast modes
bT = 1.5*log(beta*diff(ts(1:2)))/beta
bT = max(10,round(bT/microTimePeriod))*microTimePeriod +1e-12
addpath('../../ProjInt')
```

Time the projective integration simulation.

```
tic
[us,tss,uss] = PIRK2(@heteroBurstF, ts, u0(:), bT);
cputime=toc
```

Test the error in BC is negligible, for both when micro-grid point on boundary and when micro-grid points straddle boundary. For some reason the BC error climbs after $t = 0.7$—could it be `ode45` quirk?

```
xs = squeeze(patches.x);
i=[ max(find(xs<+tol)) min(find(xs>1-tol)) ];
j=[ min(find(xs>-tol)) max(find(xs<1+tol)) ];
medianBCerror=median( abs(uss(:,i)+uss(:,j))/2 ,'omitnan')
maxBCerror=max(max( abs(uss(:,i)+uss(:,j))/2 ))
assert(maxBCerror<5e-5,'BC failure')
```

Plot space-time surface of the simulation First, just a macroscale mesh plot—stereo pair.

```
Xs=mean(xs);
Us=squeeze(mean( reshape(us,length(ts),[],2*nP), 2,'omitnan'));
I=nP:2*nP;
figure(1),clf
for k = 1:2, subplot(2,2,k)
```

```
  mesh(ts,Xs(I),Us(:,I)')
  ylabel('x'), xlabel('t'), zlabel('U(x,t)')
  colormap(0.8*hsv), axis tight, view(62-4*k,45)
end
```

Second, plot a surface detailing the microscale bursts—stereo pair. Do not bother with the patch-edge values.

```
i=i(1):i(2);
xs([1 end],:) = nan;
for k = 1:2, subplot(2,2,2+k)
  surf(tss,xs(i),uss(:,i)',  'EdgeColor','none')
  ylabel('x'), xlabel('t'), zlabel('u(x,t)')
  colormap(0.7*hsv), axis tight, view(62-4*k,45)
end
```

3.2 `heteroBurstF()`: a burst of heterogeneous diffusion

This code integrates in time the derivatives computed by `heteroDiff` from within the patch coupling of `patchSys1`. Try `ode23`, although `ode45` may give smoother results. Sample every period of the microscale time fluctuations (or, at least, close to the period).

```
function [ts, ucts] = heteroBurstF(ti, ui, bT)
    global microTimePeriod
        [ts,ucts] = ode45( @patchSys1,ti+(0:microTimePeriod:bT),ui(:)
end
```

References

Combescure, Christelle (Nov. 2022). "Selecting Generalized Continuum Theories for Nonlinear Periodic Solids Based on the Instabilities of the Underlying Microstructure". In: *Journal of Elasticity*. ISSN: 1573-2681. DOI: 10.1007/s10659-022-09949-6 (cit. on p. 2).