Equation-free computational homogenisation with Dirichlet boundaries

A. J. Roberts*

January 25, 2023

Contents

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

1 Eckhardt2210eg2: example of a 1D heterogeneous diffusion by simulation on small patches

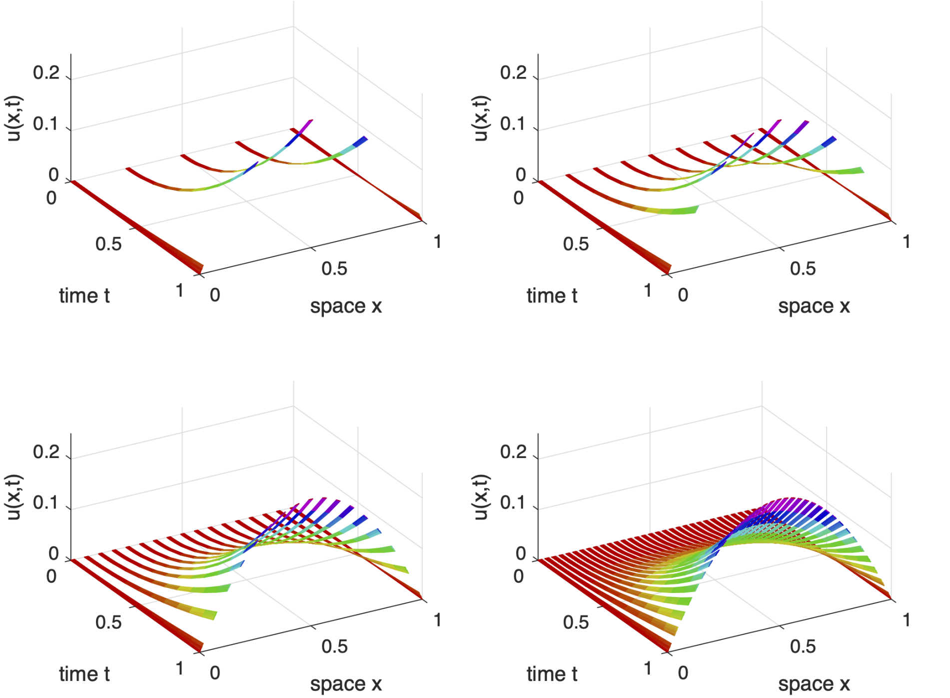
Plot an example simulation in time generated by the patch scheme applied to macroscale forced diffusion through a medium with microscale heterogeneity in space. This is more-or-less the second example of Eckhardt and Verfürth (2022) [§6.2.1].

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

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

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.

Figure 1: diffusion field $u(x,t)$ of the patch scheme applied to the forced heterogeneous diffusive (1). Simulate for $5, 9, 17, 33$ patches and compare to the full-domain simulation (65 patches, not shown).



Here use period $\epsilon = 1/130$ (so that computation completes in seconds). The patch scheme computes only on a fraction of the spatial domain, see Figure 1. 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 0.00128).

patch spacing H	0.25	0.12	0.06	0.03
sine-forcing error	0.0018	0.0009	0.0002	$1.6e{-}5$
parabolic-forcing error	$9.0e{-}9$	$3.7e{-}9$	$0.9e{-}9$	$0.06e{-}9$

The smooth sine-forcing leads to errors that appear due to patch scheme and its interpolation. The parabolic-forcing errors appear to be due to the integration errors of `ode15s` and not at all due to the patch scheme. In comparison, Eckhardt and Verfürth (2022) reported much larger errors in the range 0.001–0.1 (Figure 3).

1.1 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.

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

Set the spatial period ϵ, via integer $1/\epsilon$, and other parameters.

```
maxLog2Nx = 6
nPeriodsPatch = 2 % any integer
rEpsilon = nPeriodsPatch*(2^maxLog2Nx+1) % up to 200 say
dx = 1/(mPeriod*rEpsilon+1)
nSubP = nPeriodsPatch*mPeriod+2
tol=1e-9;
```

Loop to explore errors on various sized patches.

```
Us=[]; DXs=[]; % for storing results to compare
iPP=0; I=nan;
for log2Nx = 2:maxLog2Nx
nP = 2^log2Nx+1
```

Determine indices of patches that are common in various resolutions

```
if isnan(I), I=1:nP; else I=2*I-1; end
```

Establish the global data struct `patches` for the microscale heterogeneous lattice diffusion system (1) solved on domain $[0, 1]$, with `nP` patches, and say fourth order interpolation to provide the edge-values. 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).

```
global patches
ordCC = 4
configPatches1(@heteroDiffF,[0 1],'equispaced',nP ...
    ,ordCC,dx,nSubP,'EdgyInt',true,'hetCoeffs',a);
DX = mean(diff(squeeze(patches.x(1,1,1,:))))
DXs=[DXs;DX];
```

Set the forcing coefficients, either the original parabolic, or sinusoidal.

```
if 1 % given forcing
  patches.f1=2*( patches.x-patches.x.^2 );
  patches.f2=2*0.5+0*patches.x;
else% simple sine forcing
  patches.f1=sin(pi*patches.x);
  patches.f2=pi/2*sin(pi*patches.x);
end%if
```

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

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

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,nP) ,[2 1 3 4]) );
us = squeeze(us);
xs(end+1,:) = nan;  us(end+1,:,:) = nan;
uss=reshape(permute(us,[1 3 2]),[],length(ts));
```

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

```
iPP=iPP+1;
if iPP<=4 % only draw four subplots
  figure(1), if iPP==1, clf(), end
  subplot(2,2,iPP)
  mesh(ts,xs(:),uss)
  if iPP==1, uMax=ceil(max(uss(:))*100)/100, end
  view(60,40), colormap(0.8*hsv), zlim([0 uMax])
  xlabel('time t'), ylabel('space x'), zlabel('u(x,t)')
```

```
  drawnow
end%if
```

At the end of the log2Nx-loop, store field at the end-time from centre region of each patch for comparison.

```
i=nPeriodsPatch/2*mPeriod+1+(-mPeriod/2+1:mPeriod/2);
Us(:,:,iPP)=squeeze(us(i,end,I));
Xs=squeeze(patches.x(i,1,1,I));
if iPP>1
   assert(norm(Xs-Xsp)<tol,'sampling error in space')
   end
Xsp=Xs;
end%for log2Nx
ifOurCf2eps(mfilename) %optionally save figure
```

Assess errors by comparing to the full-domain solution

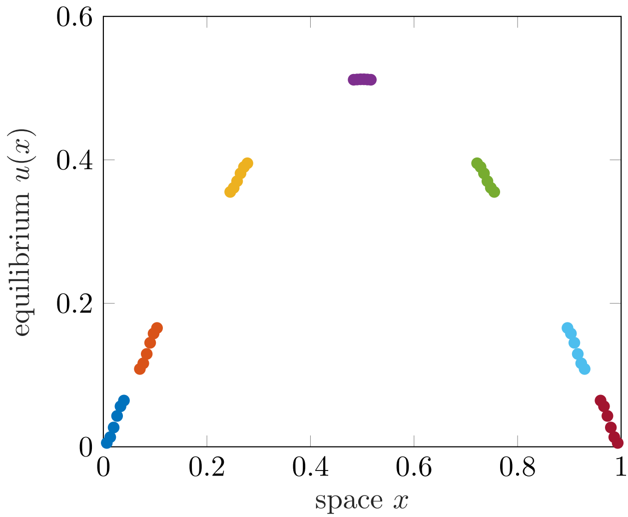
```
DXs=DXs
Uerr=squeeze(max(max(abs(Us-Us(:,:,end)))))
figure(2),clf,
loglog(DXs,Uerr,'o:')
xlabel('H'),ylabel('error')
```

1.2 heteroDiffF(): forced heterogeneous diffusion

This function codes the lattice heterogeneous diffusion inside the patches with forcing and with microscale boundary conditions on the macroscale boundaries. 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
  % macroscale Dirichlet BCs
  u( 1 ,:,:, 1 )=0; % left-edge of leftmost is zero
  u(end,:,:,end)=0; % right-edge of rightmost is zero
  % interior forced diffusion
  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
```

Figure 2: Equilibrium of the heterogeneous diffusion problem with forcing the same as that applied at time $t = 1$, and for relatively large $\epsilon = 0.04$ so we can see the patches. By default this code is for $\epsilon = 0.004$ where the microscale heterogeneity and patches are tiny.




```
  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
```

2 EckhartEquilib: find an equilibrium of a 1D heterogeneous diffusion via small patches

Sections 1 and 1.2 describe details of the problem and more details of the following configuration. The aim is to find the equilibrium, Figure 2, of the forced heterogeneous system with a forcing corresponding to that applied at time $t = 1$. Computational efficiency comes from only computing the microscale heterogeneity on small spatially sparse patches, potentially much smaller than those shown in Figure 2.

First configure the patch system Establish the microscale heterogeneity has micro-period `mPeriod` on the lattice, and coefficients to match Eckhardt2210.04536 §6.2.1.

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

Set the number of patches, the number of periods per patch, and the spatial period ϵ, via integer $1/\epsilon$.

```
nPatch = 7
nPeriodsPatch = 1 % any integer
rEpsilon = 250 % 25 for graphic, up to 2000 say
dx = 1/(mPeriod*rEpsilon+1)
nSubP = nPeriodsPatch*mPeriod+2
```

Establish the global data struct `patches` for the microscale heterogeneous lattice diffusion system (1) solved on domain $[0, 1]$, with Chebyshev-like distribution of patches, and say fourth order interpolation to provide the edge-values. Use 'edgy' interpolation.

```
global patches
ordCC = 4
configPatches1(@heteroDiffF,[0 1],'chebyshev',nPatch ...
    ,ordCC,dx,nSubP,'EdgyInt',true,'hetCoeffs',a);
```

Set the forcing coefficients, either the original parabolic, or sinusoidal. At time $t = 1$ the resultant forcing we actually apply here is simply the sum of the two components.

```
if 0 % given forcing
  patches.f1 = 2*( patches.x-patches.x.^2 );
  patches.f2 = 2*0.5+0*patches.x;
else% simple sine forcing
  patches.f1 = sin(pi*patches.x);
  patches.f2 = pi/2*sin(pi*patches.x);
end%if
```

Find equilibrium with fsolve We seek the equilibrium for the forcing that applies at time $t = 1$ (as if that specific forcing were applying for all time). Execute the function that invokes `fsolve`. For this linear problem, it is computationally quicker using a linear solver, but `fsolve` is quicker in human time, and generalises to nonlinear problems.

```
u = squeeze(execFsolve)
```

Then plot the equilibrium solution (Figure 2).

```
clf, plot(squeeze(patches.x),u,'.')
xlabel('space $x$'),ylabel('equilibrium $u(x)$')
```

Code to execute fsolve We code the function `execFsolve` to execute `fsolve` because easiest if a sub-function that computes the time derivatives has access to variables `u0` and `i`.

```
function [u,normRes] = execFsolve
global patches
```

Start the search from a zero field.

```
u0 = 0*patches.x;
```

But set patch-edge values to `Nan` in order to use `i` to index the interior sub-patch points as they are the variables.

```
u0([1 end],:,:,:) = nan;
i = find(~isnan(u0));
```

Seek the equilibrium, and report the norm of the residual.

```
[u0(i),res] = fsolve(@duidt,u0(i));
normRes = norm(res)
```

The aim is to zero the time derivatives `duidt` in the following function. First, insert the vector of variables into the patch-array of `u0`. Second, find the time derivatives via the patch scheme, and finally return a vector of those at the patch-internal points.

```
function res = duidt(ui)
  u = u0;   u(i) = ui;
  res = patchSys1(1,u);
  res = res(i);
end%function duidt
end%function execFsolve
```

Fin.

3 `Eckhardt2210eg1`: example of 1D space-time heterogeneous diffusion via computational homogenisation with projective integration and small patches

An example simulation in time generated by projective integration allied with the patch scheme applied to forced diffusion in a medium with microscale heterogeneity in both space and time. This is more-or-less the first example of Eckhardt and Verfürth (2022) [§6.2].

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

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

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. Figure 3 shows an example patch simulation.

The approximate homogenised PDE is $U_t = A_0 U_{xx} + F$ 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 Eckhardt 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!).

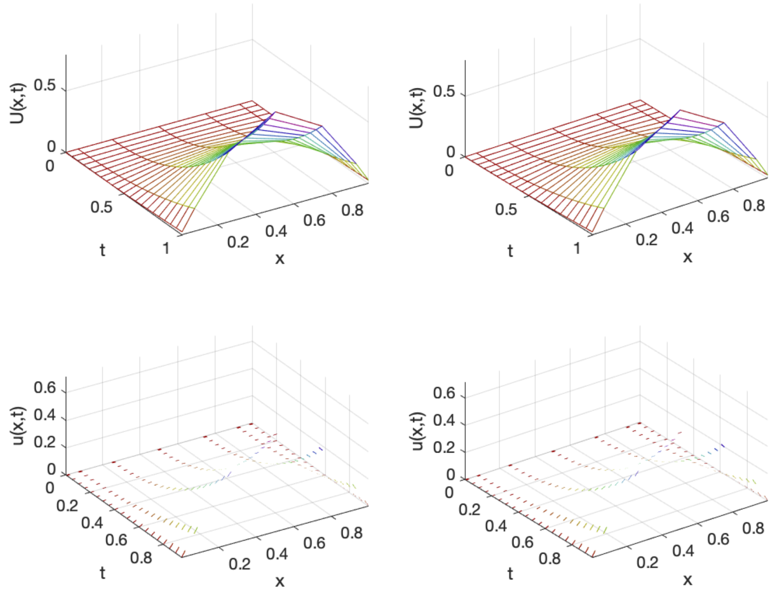
The highest wavenumber mode on the macro-grid of patches, spacing H, is the zig-zag mode on $\dot{U}_i = A_0(U_{I+1} - 2U_I + U_{I-1})/H^2 + F_I$ which evolves like $U_I = (-1)^I e^{-\alpha t}$ for the fastest 'slow rate' of $\alpha = 4A_0^2/H^2$. When $H = 0.2$ and $A_0 \approx 0.1$ this rate is $\alpha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

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 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

Figure 3: diffusion field $u(x,t)$ of the patch scheme applied to the forced space-time heterogeneous diffusive (2). Simulate for seven patches (with a 'Chebyshev' distribution): the top stereo pair is a mesh plot of a macroscale value at the centre of each spatial patch at each projective integration time-step; the bottom stereo pair shows the corresponding tiny space-time patches in which microscale computations were carried out.



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.

```
clear all
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.

```
nPeriodsPatch = 2 % any integer
```

```
rEpsilon = 100
dx = 1/(mPeriod*rEpsilon+1)
nSubP = nPeriodsPatch*mPeriod+2
tol=1e-9;
```

Set the time periodicity (global).

```
global microTimePeriod
microTimePeriod = 1/rEpsilon^2
```

Establish the global data struct **patches** for the microscale heterogeneous lattice diffusion system (2) solved on macroscale domain $[0, 1]$, with `nPatch` patches, and say fourth-order interpolation to provide the edge-values of the inter-patch coupling conditions. Distribute the patches either equispaced or chebyshev. 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).

```
nPatch = 7
ordCC = 4
Dom = 'chebyshev'
global patches
configPatches1(@heteroDiffF,[0 1],Dom,nPatch ...
    ,ordCC,dx,nSubP,'EdgyInt',true,'hetCoeffs',a);
DX = mean(diff(squeeze(patches.x(1,1,1,:))))
```

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

```
if 0 % given forcing
  patches.f1=2*( patches.x-patches.x.^2 );
  patches.f2=2*0.5+0*patches.x;
else% simple sine forcing
  patches.f1=sin(pi*patches.x);
  patches.f2=pi/2*sin(pi*patches.x);
end%if
```

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 $\beta \approx \pi^2 A_0/h^2 \approx 2000$ for patch width $h \approx 0.02$: use the formula from the Manual, with some extra factor, and rounded to the nearest multiple of the time micro-periodicity.

```
ts = linspace(0,1,21)
h=(nSubP-1)*dx;
beta = pi^2*A0/h^2 % slowest rate of fast modes
burstT = 2.5*log(beta*diff(ts(1:2)))/beta
burstT = max(10,round(burstT/microTimePeriod))*microTimePeriod +1e-12
addpath('../../ProjInt')
```

Time the projective integration simulation.

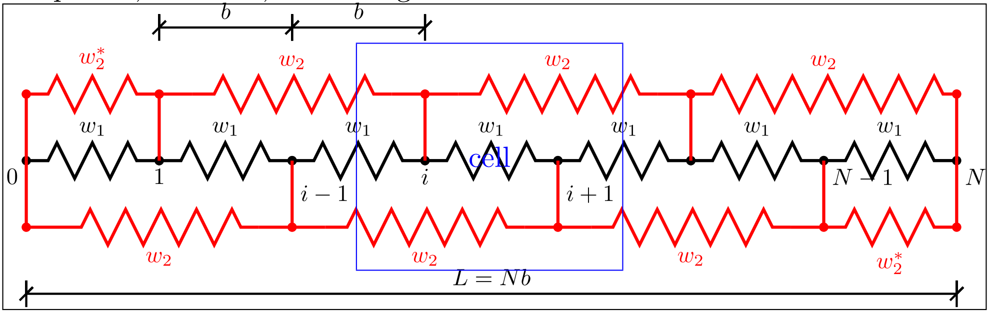
```
tic
[us,tss,uss] = PIRK2(@heteroBurstF, ts, u0(:), burstT);
cputime=toc
```

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

```
xs=squeeze(patches.x);
Xs=mean(xs);
Us=squeeze(mean( reshape(us,length(ts),[],nPatch), 2,'omitnan'));
figure(1),clf
for k = 1:2, subplot(2,2,k)
  mesh(ts,Xs(:),Us')
  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.

Figure 4: 1D arrangement of non-linear springs with connections to (a) next-to-neighbour node (Combescure 2022, Fig. 3(a)). The blue box is one cell of one period, width $2b$, containing an odd and an even i.




```
xs([1 end],:) = nan;
for k = 1:2, subplot(2,2,2+k)
  surf(tss,xs(:),uss',  '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
```

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

Started changing for BCs but nowhere near complete.

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

Suppose the spatial microscale lattice is at rest at points x_i, with constant spacing b (Figure 4). 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 4) and for every i,

$$\epsilon_i^p := \frac{1}{pb}(u_{i+p/2} - u_{i-p/2}), \quad \sigma_i^p := w_p'(\epsilon_i^p), \quad \frac{\partial^2 u_i}{\partial t^2} = \sum_{p=1}^{2} \frac{1}{pb?}(\sigma_{i+p/2}^p - \sigma_{i-p/2}^p). \tag{3}$$

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 $M_2 := 3$.

4.1 Configure heterogeneous toy elasticity systems

Set some physical parameters.

```
clear all
global b M vis i0 iN
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??
% see end-heteroToyE for function dLdt of prescribed end movement
vis = 0.01
tEnd = 130
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, nPatch = 5 % gives ratio=1 for full-domain
%nSubP = 4, nPatch = 3
%H=L/nPatch
%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 (3). Solved with `nPatch` patches, and high-order interpolation to provide the edge-values of the inter-patch coupling conditions.

```
global patches
configPatches1(@heteroToyE,[0 L],'equispaced',nPatch ...
    ,0,b,nSubP,'EdgyInt',edgyInt);
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
```

4.2 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
```

4.3 Simulate in time

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 4, 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, tEnd*linspace(0,1,41), 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,nPatch);
xs(end+1,:) = nan;
uvs = reshape( permute( reshape(ust ...
      ,length(ts),nSubP,4,1,nPatch) ,[2 3 1 4 5]) ,nSubP,[],1,nPatch)
uvs = reshape( patchEdgeInt1(uvs) ,nSubP,4,[],nPatch);
% extract displacements
us = reshape( permute( uvs(:,1:2,:,:) ...
     ,[2 1 4 3]) ,2*nSubP,nPatch,[]);
us(end+1,:,:) = nan;
us = reshape(us,[],length(ts));
% extract velocities
vs = reshape( permute( uvs(:,3:4,:,:) ...
     ,[2 1 4 3]) ,2*nSubP,nPatch,[]);
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 displacements 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
```

Similarly plot velocities

```
  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
```

4.4 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
```

Separate state vector into displacement and velocity fields: u_{ijI} is the displacement at the jth point in the ith 2-cell in the Ith patch; similarly for velocity v_{ijI}. That is, physically neighbouring points have different j, whereas physical next-to-neighbours have i different by one.

```
  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.

```
if ~isempty(i0), uvt(i0)=0; end
if ~isempty(iN), uvt(iN(3:4))=dLdt(t); end% vel=d/dt of end displacem
end% function
```

4.5 `dLdt()`: prescribed movement of length

```
function Ld=dLdt(t)
Ld=-0.03*cos(t/20);
end
```

5 `monoscaleDiffEquil2`: equilibrium of a 2D monoscale heterogeneous diffusion via small patches

Here we find the steady state $u(x,y)$ to the heterogeneous PDE(inspired by Freese et al.[1] §5.2)

$$u_t = A(x,y)\vec\nabla\vec\nabla u - f,$$

on domain $[-1,1]^2$ with Dirichlet BCs, for coefficient 'diffusion' matrix

$$A := \begin{bmatrix} 2 & a \\ a & 2 \end{bmatrix} \quad \text{with } a := \operatorname{sign}(xy) \text{ or } a := \sin(\pi x)\sin(\pi y),$$

and for forcing $f(x,y)$ such that the exact equilibrium is

$$u = x\big(1-e^{1-|x|}\big)y\big(1-e^{1-|y|}\big).$$

But for simplicity, let's do $u = x(1-x^2)y(1-y^2)$ for which we code f later—as determined by this computer algebra.

[1] http://arxiv.org/abs/2211.13731

```
on gcd; factor sin;
%let { df(sign(~x),~x)=>0
%     , df(abs(~x),~x)=>sign(x)
%     , abs(~x)^2=>abs(x), sign(~x)^2=>1 };
%u:=x*(1-exp(1-abs(x)))*y*(1-exp(1-abs(y)));
u:=x*(1-x^2)*y*(1-y^2);
a:=sin(pi*x)*sin(pi*y);
f:=2*df(u,x,x)+2*a*df(u,x,y)+2*df(u,y,y);
```

Clear, and initiate globals.

```
clear all
global patches i
```

Patch configuration Initially use 7×7 patches in the square $(-1, 1)^2$. For continuous forcing we may have small patches of any reasonable microgrid spacing—here the microgrid error dominates.

```
nPatch = 7
nSubP = 5
dx = 0.03
```

Specify some order of interpolation.

```
configPatches2(@monoscaleDiffForce2,[-1 1 -1 1],'equispace' ...
    ,nPatch ,4 ,dx ,nSubP ,'EdgyInt',true );
```

Compute the time-constant coefficient and time-constant forcing, and store them in struct `patches` for access by the microcode of Section 5.1.

```
  x=patches.x;  y=patches.y;
  patches.A = sin(pi*x).*sin(pi*y);
  patches.fu = ...
    +2*patches.A.*(9*x.^2.*y.^2-3*x.^2-3*y.^2+1) ...
    +12*x.*y.*(x.^2+y.^2-2);
```

By construction, the PDE has analytic solution

```
uAnal = x.*(1-x.^2).*y.*(1-y.^2);
```

Solve for steady state Set initial guess of zero, with NaN to indicate patch-edge values. Index i are the indices of patch-interior points, and the number of unknowns is then its length.

```
u0 = zeros(nSubP,nSubP,1,1,nPatch,nPatch);
u0([1 end],:,:) = nan;  u0(:,[1 end],:) = nan;
i = find(~isnan(u0));
nVars = numel(i)
```

Solve by iteration. Use fsolve for simplicity and robustness (using optimoptions to omit its trace information).

```
tic;
uSoln = fsolve(@theRes,u0(i) ...
        ,optimoptions('fsolve','Display','off'));
solnTime = toc
```

Store the solution into the patches, and give magnitudes.

```
u0(i) = uSoln;
normSoln = norm(uSoln)
normResidual = norm(theRes(uSoln))
errors = uAnal(i)-uSoln;
normError = norm(errors)
```

Draw solution profile First reshape arrays to suit 2D space surface plots.

```
figure(1), clf, colormap(hsv)
x = squeeze(patches.x); y = squeeze(patches.y);
u = reshape(permute(squeeze(u0),[1 3 2 4]), [numel(x) numel(y)]);
```

Draw the patch solution surface, with edge-values omitted as already NaN by not bothering to interpolate them.

```
surf(x(:),y(:),u'); view(60,55)
xlabel('x'), ylabel('y'), zlabel('u(x,y)')
```

5.1 monoscaleDiffForce2(): microscale discretisation inside patches of forced diffusion PDE

This function codes the lattice heterogeneous diffusion of the PDE inside the patches. For 6D input arrays u, x, and y, computes the time derivative at each point in the interior of a patch, output in ut.

```
function ut = monoscaleDiffForce2(t,u,patches)
  dx = diff(patches.x(2:3));  % x space step
  dy = diff(patches.y(2:3));  % y space step
  ix = 2:size(u,1)-1; % x interior points in a patch
  iy = 2:size(u,2)-1; % y interior points in a patch
  ut = nan+u;         % preallocate output array
```

Set Dirichlet boundary value of zero around the square domain, or code some function variation.

```
u( 1 ,:,:,:, 1 ,:)=0; % left edge of left patches
u( 1 ,:,:,:, 1 ,:)=(1+patches.y)/2; % or code function of y
u(end,:,:,:,end,:)=0; % right edge of right patches
u(:, 1 ,:,:,:, 1 )=0; % bottom edge of bottom patches
u(:,end,:,:,:,end)=0; % top edge of top patches
u(:,end,:,:,:,end)=1; % or code function of x
```

Compute the time derivatives via stored forcing and coefficients. Easier to code by conflating the last four dimensions into the one `,:`.

```
  ut(ix,iy,:) ...
  = 2*diff(u(:,iy,:),2,1)/dx^2 +2*diff(u(ix,:,:),2,2)/dy^2 ...
   +2*patches.A(ix,iy,:).*( u(ix+1,iy+1,:) -u(ix-1,iy+1,:) ...
     -u(ix+1,iy-1,:) +u(ix-1,iy-1,:) )/(4*dx*dy) ...
   -patches.fu(ix,iy,:);
end%function monoscaleDiffForce2
```

5.2 `theRes()`: function to zero

This functions converts a vector of values into the interior values of the patches, then evaluates the time derivative of the system, and returns the vector of patch-interior time derivatives.

```
function f=theRes(u)
  global patches i
  v=nan(size(patches.x+patches.y));
  v(i)=u;
  f=patchSys2(0,v(:),patches);
  f=f(i);
end%function theRes
```

Fin.

6 `twoscaleDiffEquil2`: equilibrium of a 2D twoscale heterogeneous diffusion via small patches

Here we find the steady state $u(x,y)$ to the heterogeneous PDE (inspired by Freese et al.[2] §5.3.1)

$$u_t = A(x,y)\vec\nabla\vec\nabla u - f,$$

on domain $[-1,1]^2$ with Dirichlet BCs, for coefficient 'diffusion' matrix, varying with period 2ϵ on the microscale $\epsilon = 2^{-7}$, of

$$A := \begin{bmatrix} 2 & a \\ a & 2 \end{bmatrix} \quad \text{with } a := \sin(\pi x/\epsilon)\sin(\pi y/\epsilon),$$

and for forcing $f := (x + \cos 3\pi x)y^3$.

Clear, and initiate globals.

```
clear all
global patches i
```

First establish the microscale heterogeneity has micro-period `mPeriod` on the spatial lattice. Set the phase of the heterogeneity so that each patch centre is a point of symmetry of the diffusivity. Then `configPatches2` replicates the heterogeneity to fill each patch.

```
mPeriod = 6
z = (0.5:mPeriod)'/mPeriod;
A = sin(2*pi*z).*sin(2*pi*z');
```

Set the periodicity, via ϵ, and other microscale parameters.

```
nPeriodsPatch = 1 % any integer
epsilon = 2^(-5) % so we can see patches
dx = (2*epsilon)/mPeriod
nSubP = nPeriodsPatch*mPeriod+2 % for edgy int
```

Patch configuration Say use 7×7 patches in $(-1,1)^2$, fourth order interpolation, and either 'equispace' or 'chebyshev':

```
nPatch = 7
configPatches2(@twoscaleDiffForce2,[-1 1],'equispace' ...
    ,nPatch ,4 ,dx ,nSubP ,'EdgyInt',true ,'hetCoeffs',A );
```

[2] http://arxiv.org/abs/2211.13731

Compute the time-constant forcing, and store in struct `patches` for access by the microcode of Section 6.1.

```
patches.fu = 100*(patches.x+cos(3*pi*patches.x)).*patches.y.^3;
```

Solve for steady state Set initial guess of zero, with `NaN` to indicate patch-edge values. Index `i` are the indices of patch-interior points, and the number of unknowns is then its length.

```
u0 = zeros(nSubP,nSubP,1,1,nPatch,nPatch);
u0([1 end],:,:) = nan;  u0(:,[1 end],:) = nan;
i = find(~isnan(u0));
nVariables = numel(i)
```

Solve by iteration. Use `fsolve` for simplicity and robustness (and using `optimoptions` to omit trace information).

```
tic;
uSoln = fsolve(@theRes,u0(i) ...
        ,optimoptions('fsolve','Display','off'));
solnTime = toc
```

Store the solution into the patches, and give magnitudes.

```
u0(i) = uSoln;
normSoln = norm(uSoln)
normResidual = norm(theRes(uSoln))
```

Draw solution profile First reshape arrays to suit 2D space surface plots.

```
figure(1), clf, colormap(hsv)
x = squeeze(patches.x); y = squeeze(patches.y);
u = reshape(permute(squeeze(u0),[1 3 2 4]), [numel(x) numel(y)]);
```

Draw the patch solution surface, with edge-values omitted as already `NaN` by not bothering to interpolate them.

```
surf(x(:),y(:),u'); view(60,55)
xlabel('x'), ylabel('y'), zlabel('u(x,y)')
```

6.1 `twoscaleDiffForce2()`: microscale discretisation inside patches of forced diffusion PDE

This function codes the lattice heterogeneous diffusion of the PDE inside the patches. For 6D input arrays `u`, `x`, and `y`, computes the time derivative at each point in the interior of a patch, output in `ut`.

```
function ut = twoscaleDiffForce2(t,u,patches)
  dx = diff(patches.x(2:3));  % x space step
  dy = diff(patches.y(2:3));  % y space step
  ix = 2:size(u,1)-1; % x interior points in a patch
  iy = 2:size(u,2)-1; % y interior points in a patch
  ut = nan+u;         % preallocate output array
```

Set Dirichlet boundary value of zero around the square domain.

```
  u( 1 ,:,:,:, 1 ,:)=0; % left edge of left patches
  u(end,:,:,:,end,:)=0; % right edge of right patches
  u(:, 1 ,:,:,:, 1 )=0; % bottom edge of bottom patches
  u(:,end,:,:,:,end)=0; % top edge of top patches
```

Compute the time derivatives via stored forcing and coefficients. Easier to code by conflating the last four dimensions into the one `,:`.

```
  ut(ix,iy,:) ...
  = 2*diff(u(:,iy,:),2,1)/dx^2 +2*diff(u(ix,:,:),2,2)/dy^2 ...
   +2*patches.cs(ix,iy).*( u(ix+1,iy+1,:) -u(ix-1,iy+1,:) ...
     -u(ix+1,iy-1,:) +u(ix-1,iy-1,:) )/(4*dx*dy) ...
   -patches.fu(ix,iy,:);
end%function twoscaleDiffForce2
```

6.2 `theRes()`: function to zero

This functions converts a vector of values into the interior values of the patches, then evaluates the time derivative of the system, and returns the vector of patch-interior time derivatives.

```
function f=theRes(u)
  global patches i
  v=nan(size(patches.x+patches.y));
  v(i)=u;
  f=patchSys2(0,v(:),patches);
```

```
  f=f(i);
end%function theRes
```

Fin.

7 abdulleDiffEquil2: equilibrium of a 2D twoscale heterogeneous diffusion via small patches

Here we find the steady state $u(x,y)$ to the heterogeneous PDE (inspired by Abdulle, Arjmand, and Paganoni 2020, §5.1)

$$u_t = \vec{\nabla}\cdot[a(x,y)\vec{\nabla}u] + 10,$$

on square domain $[0,1]^2$ with zero-Dirichlet BCs, for coefficient 'diffusion' matrix, varying with period ϵ of (their (45))

$$a := \frac{2+1.8\sin 2\pi x/\epsilon}{2+1.8\cos 2\pi y/\epsilon} + \frac{2+\sin \pi y/\epsilon}{2+1.8\cos 2\pi x/\epsilon}.$$

The solution shows some nice little microscale wiggles.

Clear, and initiate globals.

```
clear all
global patches i
```

First establish the microscale heterogeneity has micro-period `mPeriod` on the spatial micro-grid lattice. Then `configPatches2` replicates the heterogeneity to fill each patch. (These diffusion coefficients should really recognise the half-grid-point shifts, but let's not bother.)

```
mPeriod = 6
x = (0.5:mPeriod)'/mPeriod;  y=x';
a = (2+1.8*sin(2*pi*x))./(2+1.8*sin(2*pi*y)) ...
   +(2+    sin(2*pi*y))./(2+1.8*sin(2*pi*x));
```

Set the periodicity, via ϵ, and other microscale parameters.

```
nPeriodsPatch = 1 % any integer
epsilon = 2^(-4) % not tiny, so we can see patches
dx = epsilon/mPeriod
nSubP = nPeriodsPatch*mPeriod+2 % when edgy int
```

Patch configuration Choose either Dirichlet (default) or Neumann on the left boundary in coordination with micro-code in Section 7.1

```
Dom.bcOffset = zeros(2);
if 1, Dom.bcOffset(1)=0.5; end% left Neumann
```

Say use 7×7 patches in $(0,1)^2$, fourth order interpolation, and either 'equispace' or 'chebyshev':

```
nPatch = 7
Dom.type='equispace';
configPatches2(@abdulleDiffForce2,[0 1],Dom ...
    ,nPatch ,4 ,dx ,nSubP ,'EdgyInt',true ,'hetCoeffs',a );
```

Solve for steady state Set initial guess of zero, with `NaN` to indicate patch-edge values. Index `i` are the indices of patch-interior points, and the number of unknowns is then its length.

```
u0 = zeros(nSubP,nSubP,1,1,nPatch,nPatch);
u0([1 end],:,:) = nan;  u0(:,[1 end],:) = nan;
i = find(~isnan(u0));
nVariables = numel(i)
```

Solve by iteration. Use `fsolve` for simplicity and robustness (and using `optimoptions` to omit trace information).

```
tic;
uSoln = fsolve(@theRes,u0(i) ...
        ,optimoptions('fsolve','Display','off'));
solnTime = toc
```

Store the solution into the patches, and give magnitudes.

```
u0(i) = uSoln;
normSoln = norm(uSoln)
normResidual = norm(theRes(uSoln))
```

Draw solution profile First reshape arrays to suit 2D space surface plots.

```
figure(1), clf, colormap(hsv)
x = squeeze(patches.x); y = squeeze(patches.y);
u = reshape(permute(squeeze(u0),[1 3 2 4]), [numel(x) numel(y)]);
```

Draw the patch solution surface, with edge-values omitted as already NaN by not bothering to interpolate them.

```
surf(x(:),y(:),u'); view(60,55)
xlabel('x'), ylabel('y'), zlabel('u(x,y)')
```

7.1 abdulleDiffForce2(): microscale discretisation inside patches of forced diffusion PDE

This function codes the lattice heterogeneous diffusion of the PDE inside the patches. For 6D input arrays u, x, and y, computes the time derivative at each point in the interior of a patch, output in ut.

```
function ut = abdulleDiffForce2(t,u,patches)
  dx = diff(patches.x(2:3));  % x space step
  dy = diff(patches.y(2:3));  % y space step
  ix = 2:size(u,1)-1; % x interior points in a patch
  iy = 2:size(u,2)-1; % y interior points in a patch
  ut = nan+u;         % preallocate output array
```

Set Dirichlet boundary value of zero around the square domain, but also cater for zero Neumann condition on the left boundary.

```
  u( 1 ,:,:,:, 1 ,:)=0; % left edge of left patches
  u(end,:,:,:,end,:)=0; % right edge of right patches
  u(:, 1 ,:,:,:, 1 )=0; % bottom edge of bottom patches
  u(:,end,:,:,:,end)=0; % top edge of top patches
  if 1, u(1,:,:,:,1,:)=u(2,:,:,:,1,:); end% left Neumann
```

Compute the time derivatives via stored forcing and coefficients. Easier to code by conflating the last four dimensions into the one ,:.

```
  ut(ix,iy,:) = diff(patches.cs(:,iy).*diff(u(:,iy,:)))/dx^2 ...
      + diff(patches.cs(ix,:).*diff(u(ix,:,:),1,2),1,2)/dy^2 ...
      + 10;
end%function abdulleDiffForce2
```

7.2 theRes(): function to zero

This functions converts a vector of values into the interior values of the patches, then evaluates the time derivative of the system, and returns the vector of patch-interior time derivatives.

```
function f=theRes(u)
  global patches i
  v=nan(size(patches.x+patches.y));
  v(i)=u;
  f=patchSys2(0,v(:),patches);
  f=f(i);
end%function theRes
```

Fin.

8 `randAdvecDiffEquil2`: equilibrium of a 2D random heterogeneous advection-diffusion via small patches

Here we find the steady state $u(x, y)$ of the heterogeneous PDE (inspired by Bonizzoni et al.[3] §6.2)

$$u_t = \mu_1 \nabla^2 u - (\cos\mu_2, \sin\mu_2) \cdot \vec{\nabla} u - u + f\,,$$

on domain $[0, 1]^2$ with Neumann BCs, for microscale random diffusion and advection coefficients, $\mu_1 \in [0.01, 0.1]$ and $\mu_2 \in [0, 2\pi)$, and for forcing

$$f := \exp\left[-\frac{(x-\mu_3)^2 + (x-\mu_4)^2}{\mu_5^2}\right],$$

smoothly varying in space for fixed $\mu_3, \mu_4 \in [0.25, 0.75]$ and $\mu_5 \in [0.1, 0.25]$. The above system is dominantly diffusive for lengths scales $\ell < 0.01 = \min\mu_1$.

Clear, and initiate globals.

```
clear all
global patches i
```

First establish the microscale heterogeneity has micro-period `mPeriod` on the spatial lattice. Then `configPatches2` replicates the heterogeneity to fill each patch.

```
mPeriod = 4
mu1 = 10.^(-1-rand(mPeriod))
mu2 = 2*pi*rand(mPeriod)
cs = cat(3,mu1,cos(mu2),sin(mu2));
meanDiffAdvec=squeeze(mean(mean(cs)))
```

[3] http://arxiv.org/abs/2211.15221

Set the periodicity, ϵ, and other microscale parameters.

```
nPeriodsPatch = 1 % any integer
epsilon = 2^(-4) % so we can see patches
dx = epsilon/mPeriod
nSubP = nPeriodsPatch*mPeriod+2 % for edgy int
```

Patch configuration Say use 7×7 patches in $(0,1)^2$, fourth order interpolation, either 'equispace' or 'chebyshev', and the offset for Neumann boundary conditions:

```
nPatch = 7
Dom.type= 'equispace';
Dom.bcOffset = 0.5;
configPatches2(@randAdvecDiffForce2,[0 1],Dom ...
    ,nPatch ,4 ,dx ,nSubP ,'EdgyInt',true ,'hetCoeffs',cs );
```

Compute the time-constant forcing, and store in struct `patches` for access by the microcode of Section 8.1.

```
mu = [ 0.25+0.5*rand(1,2) 0.1+0.15*rand ]
patches.fu = exp(-((patches.x-mu(1)).^2+(patches.y-mu(2)).^2)/mu(3)^2
```

Solve for steady state Set initial guess of zero, with `NaN` to indicate patch-edge values. Index `i` are the indices of patch-interior points, and the number of unknowns is then its length.

```
u0 = zeros(nSubP,nSubP,1,1,nPatch,nPatch);
u0([1 end],:,:) = nan;  u0(:,[1 end],:) = nan;
i = find(~isnan(u0));
nVariables = numel(i)
```

Solve by iteration. Use `fsolve` for simplicity and robustness (and using `optimoptions` to omit trace information).

```
tic;
uSoln = fsolve(@theRes,u0(i) ...
        ,optimoptions('fsolve','Display','off'));
solnTime = toc
```

Store the solution into the patches, and give magnitudes.

```
u0(i) = uSoln;
normSoln = norm(uSoln)
normResidual = norm(theRes(uSoln))
```

Draw solution profile First reshape arrays to suit 2D space surface plots.

```
figure(1), clf, colormap(hsv)
x = squeeze(patches.x); y = squeeze(patches.y);
u = reshape(permute(squeeze(u0),[1 3 2 4]), [numel(x) numel(y)]);
```

Draw the patch solution surface, with edge-values omitted as already `NaN` by not bothering to interpolate them.

```
surf(x(:),y(:),u'); view(60,55)
xlabel('x'), ylabel('y'), zlabel('u(x,y)')
```

8.1 `randAdvecDiffForce2()`: microscale discretisation inside patches of forced diffusion PDE

This function codes the lattice heterogeneous diffusion of the PDE inside the patches. For 6D input arrays `u`, `x`, and `y`, computes the time derivative at each point in the interior of a patch, output in `ut`.

```
function ut = randAdvecDiffForce2(t,u,patches)
  dx = diff(patches.x(2:3));  % x space step
  dy = diff(patches.y(2:3));  % y space step
  ix = 2:size(u,1)-1; % x interior points in a patch
  iy = 2:size(u,2)-1; % y interior points in a patch
  ut = nan+u;         % preallocate output array
```

Set Neumann boundary condition of zero derivative around the square domain.

```
  u( 1 ,:,:,:, 1 ,:)=u(  2  ,:,:,:, 1 ,:); % left edge of left patche
  u(end,:,:,:,end,:)=u(end-1,:,:,:,end,:); % right edge of right patc
  u(:, 1 ,:,:,:, 1 )=u(:,  2  ,:,:,:, 1 ); % bottom edge of bottom pa
  u(:,end,:,:,:,end)=u(:,end-1,:,:,:,end); % top edge of top patches
```

Compute the time derivatives via stored forcing and coefficients. Easier to code by conflating the last four dimensions into the one `,:`.

```
  ut(ix,iy,:) ...
  = patches.cs(ix,iy,1).*(diff(u(:,iy,:),2,1)/dx^2 ...
                         +diff(u(ix,:,:),2,2)/dy^2)...
   -patches.cs(ix,iy,2).*(u(ix+1,iy,:)-u(ix-1,iy,:))/(2*dx) ...
   -patches.cs(ix,iy,3).*(u(ix,iy+1,:)-u(ix,iy-1,:))/(2*dy) ...
   -u(ix,iy,:) +patches.fu(ix,iy,:);
end%function randAdvecDiffForce2
```

8.2 `theRes()`: function to zero

This functions converts a vector of values into the interior values of the patches, then evaluates the time derivative of the system, and returns the vector of patch-interior time derivatives.

```
function f=theRes(u)
  global patches i
  v=nan(size(patches.x+patches.y));
  v(i)=u;
  f=patchSys2(0,v(:),patches);
  f=f(i);
end%function theRes
```

Fin.

References

Abdulle, Assyr, Doghonay Arjmand, and Edoardo Paganoni (2020). *A parabolic local problem with exponential decay of the resonance error for numerical homogenization*. Tech. rep. Institute of Mathematics, École Polytechnique Fédérale de Lausanne (cit. on p. 27).

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 pp. 14, 15).

Eckhardt, Daniel and Barbara Verfürth (Oct. 2022). *Fully discrete Heterogeneous Multiscale Method for parabolic problems with multiple spatial and temporal scales*. Tech. rep. http://arxiv.org/abs/2210.04536 (cit. on pp. 2, 3, 10).