Equation-free computational homogenisation with Dirichlet boundaries

A. J. Roberts*

October 25, 2022

Contents

1 `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.

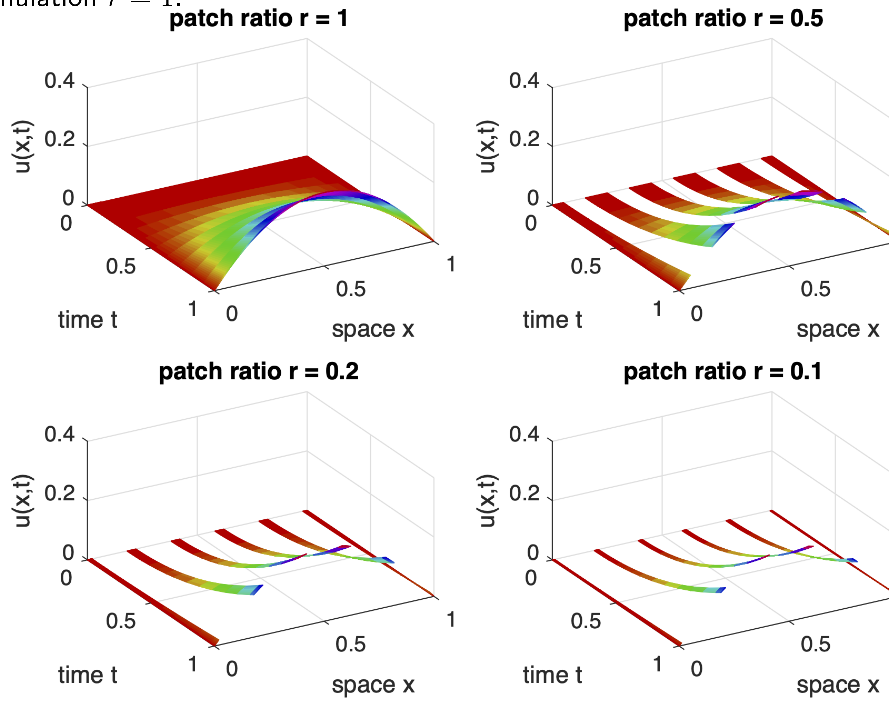
Suppose the spatial microscale lattice is at points x_i, with constant spacing dx. With dependent variables $u_i(t)$, simulate the microscale lattice forced diffusion system

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

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.

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

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



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 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 $1/1200$).

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

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. 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 (1) 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 ...
```

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

```
    ,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);
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
```

1.2 `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
```

2 `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

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

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.

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 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'.

2.1 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 (2) 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
```

2.2 `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
```