Couple microscale periodic patches to simulate macroscale emergent dynamics : Supplementary material

Hammad Alotaibi[1] Barry Cox[2] A. J. Roberts[3]

September 7, 2017

A Code for 3D atomistic simulation

This ancillary material provides the numerical code for simulating a microscale patch of atoms coupled over macroscale empty space to boundary values of temperature. It is included to document and potentially reproduce the results.

A.1 Main driver code

```
% coded 3d simulation of position and velocity of
% interacting atoms.  Uses ode23 to do time integration.
% HA 2015--6--19
global ll mucontrol TL TR ii nfns nAux hh Khsq

nAtom=64  % number of atoms
tEnd=3 % end time of simulation
ll=nAtom.^(1/3);   % length of periodic patch (inter-atom eq is at one)
mucontrol=0 % zero is no control, circa 30 seems best
TL=1, TR=0.5 % macroscale boundary values of temp
hh=ll  % macroscale BCs applied at +/-hh
rng('shuffle'); seed=422 %100+floor(900*rand); %random realisation seed
fileroot=['ctrlpatch' num2str(seed) 'N' num2str(nAtom)]
nAux=12; % number of auxillary variables computed
Khsq=0.5/(ll/2)^2; % coefficient of control

rng(seed);
% distribute atoms, randomly up to one per box
```

```
ns=ceil(ll);
i=linspace(-0.5,0.5,2*ns+1);
[j,i,p]=meshgrid(ll*i(2:2:end));
[~,k]=sort(rand(size(i(:)))); k=k(1:nAtom);
xq=[i(k) j(k) p(k) zeros(nAtom,3)]';
% add smallish, mean-zero, random position and velocity
zz=rand(6,nAtom)-0.5; zz=zz-repmat(mean(zz,2),1,nAtom);
xq=xq+0.3*diag([1,1,1,2,2,2])*zz; xq=xq(:);
% for imposing triple periodicity in space
ii=1:6:6*nAtom; ii=[ii ii+1 ii+2]+nAux;

% simulation in time from given ICs
nfns=0;
[ts,Txqs]=ode23(@ctrlhddtu3dode,[0 tEnd],[zeros(nAux,1);xq]);
% impose periodicity on the computed positions
xqs=Txqs(:,nAux+1:end);  ii=ii-nAux;
xqs(:,ii)=xqs(:,ii)-round(xqs(:,ii)/ll)*ll;
% auxillary quantities at middle of time steps
Tuvws=diff(Txqs(:,1:nAux))./repmat(diff(ts),1,nAux);
tsx=(ts(1:end-1)+ts(2:end))/2;
nFunctions=nfns

ctrlgraphs % draw graphical output
```

A.2 Interpose periodicity on positions

This function avoids MATLAB's `ode23` objecting to discontinuities as atoms move across edges of the periodic box.

```
function dxq=ctrlhddtu3dode(t,xq)
% Computes time derivative of position and velocity of
% interacting particles for Matlab integrator.
% HA Jan 2015 -- 2016
global  ll ii nfns
nfns=nfns+1;
% impose triple periodicity on positions
xq(ii)=xq(ii)-round(xq(ii)/ll)*ll;
dxq=ctrlhddtu3d(xq,t);
end
```

A.3 Time derivatives of position and velocity

```
function dTxq=ctrlhddtu3d(Txq,t)
% Computes time derivative of position and velocity of
% interacting particles.  Force triply periodic in
% space, periodicity ll. For the moment ignore that this
% dynamical system is sympletic.  HA Jan 2014 -- 2016
global  ll mucontrol TL TR nAux hh Khsq
% unpack positions
xq=Txq(nAux+1:end);
x=xq(1:6:end);
y=xq(2:6:end);
z=xq(3:6:end);
% d/dt position = velocity
dxq=nan(size(xq));
dxq(1:6:end)=xq(4:6:end);
dxq(2:6:end)=xq(5:6:end);
dxq(3:6:end)=xq(6:6:end);

% Assume triple images enough to capture all significant.
[xxt,xx,lls]=meshgrid(x,x,[-ll 0 ll]);
[yyt,yy,lls]=meshgrid(y,y,[-ll 0 ll]);
[zzt,zz,lls]=meshgrid(z,z,[-ll 0 ll]);
[ddx,px]=min((xx-xxt+lls).^2,[],3);
[ddy,py]=min((yy-yyt+lls).^2,[],3);
[ddz,pz]=min((zz-zzt+lls).^2,[],3);
ds=sqrt(ddx+ddy+ddz)+1e-8;
% forces as a function of distance (with extra / dist)
fs=ds.^(-7-1)-ds.^(-13-1);
fs=-min(100,-fs); % ad hoc limit on force
fx=(xxt(:,:,1)-xx(:,:,1)-(px-2).*ll).*fs;
fy=(yyt(:,:,1)-yy(:,:,1)-(py-2).*ll).*fs;
fz=(zzt(:,:,1)-zz(:,:,1)-(pz-2).*ll).*fs;
% d/dt velocities = sum of forces
dxq(4:6:end)=sum(fx,2);
dxq(5:6:end)=sum(fy,2);
dxq(6:6:end)=sum(fz,2);

% proportional controller of the patch temperature
halfcore=ll/8; % halfwidth of core and action regions
xaction=ll/4;  % action regions centred at quarter points
```

```
% Find which atoms are in each region
jl=(abs(x+xaction)<halfcore);
jc=(abs(x        )<halfcore);
jr=(abs(x-xaction)<halfcore);
% unpack velocities and KEs, to find regional temperatures
u=xq(4:6:end); v=xq(5:6:end); w=xq(6:6:end); ke=(u.^2+v.^2+w.^2)/2;
Tl=mean(ke(jl));
Tc=mean(ke(jc));
Tr=mean(ke(jr));
% Control towards specified environmental temperature
rlh=ll/8/hh;
T0=(Tc-(TR+TL)*rlh^2/6)/(1-rlh^2/3);
Tintl=T0-(TR-TL)*rlh+(TR-2*T0+TL)*rlh^2*13/6;
Tintr=T0+(TR-TL)*rlh+(TR-2*T0+TL)*rlh^2*13/6;
ratel=mucontrol*(Tintl-Tl)*Khsq/Tl/2;
rater=mucontrol*(Tintr-Tr)*Khsq/Tr/2;
% de/accelerate in dirn of velocity, propto control
dxq(4:6:end)=dxq(4:6:end)+(ratel*jl+rater*jr).*u;
dxq(5:6:end)=dxq(5:6:end)+(ratel*jl+rater*jr).*v;
dxq(6:6:end)=dxq(6:6:end)+(ratel*jl+rater*jr).*w;

% other auxillary quantities are mean velocity in regions
ul_av=mean(u(jl)); vl_av=mean(v(jl)); wl_av=mean(w(jl));
uc_av=mean(u(jc)); vc_av=mean(v(jc)); wc_av=mean(w(jc));
ur_av=mean(u(jr)); vr_av=mean(v(jr)); wr_av=mean(w(jr));
aux=[Tl;Tc;Tr;ul_av;vl_av;wl_av;uc_av;vc_av;wc_av;ur_av;vr_av;wr_av];

% return auxillary quantities and atomic time derivatives
dTxq=[aux;dxq];
end
```

B Computer algebra derives a slow manifold

The following computer algebra code constructs a macroscale slow manifold to the controlled periodic-patch mesoscale PDE (20) in the autonomous case $\sigma = 0$. We use the language Reduce[1] because it is both free and perhaps the fastest general purpose computer algebra system [1].

Make printing pretty.

[1] http://reduce-algebra.com/

```
on div; off allfac; on revpri;
factor gamma,mu,delta,hh,kk,alpha;
```

Scale space to $\xi = (x - X_j)/H$ where `hh` $= H$.

```
depend xi,x;
let df(xi,x)=>1/hh;
```

Parametrise solution in terms of amplitudes U_j such that $\partial U_j/\partial t = g(\vec{U}, t)$. Let's define U_j to be the average over an averaging region, then the interpolated values will just be the required averages in those regions: although there may be a glitch for physical boundary conditions.

```
operator uu; depend uu,t;
let df(uu(~k),t)=>sub(j=k,g);
```

Zeroth approximation field is constant in each element. Write the patch field in four quarters: the core $u_c(\xi, t)$ for $|\xi| < \frac{1}{4}r$; the left action region $u_l(\xi, t)$ for $-\frac{3}{4}r < \xi < \frac{1}{4}r$; the right action region $u_r(\xi, t)$ for $\frac{1}{4}r < \xi < \frac{3}{4}r$; and the 'back/buffer' field $u_b(\xi, t)$ for $\frac{3}{4}r < |\xi| \leq r$.

```
uc:=ul:=ur:=ub:=uu(j);
g:=0;
```

Define averages over various regions.

```
operator meanc; linear meanc;
operator meanl; operator meanr;
qr:=r/4;
let { meanc(xi^~~p,xi)=>qr^p*(1+(-1)^p)/2/(p+1)
    , meanc(1,xi)=>1
    , meanr(~a,xi)=>meanc(sub(xi=+r/2+xi,a),xi)
    , meanl(~a,xi)=>meanc(sub(xi=-r/2+xi,a),xi)
    };
```

The iterative refinement seeks updates that are polynomial in the subpatch field. Here gather the coefficients for later reference, where `maxn` is the maximum order of the polynomial: increase for higher-order or more nonlinear problems.

```
maxn:=4;
operator cl,cc,cr,cb;
cvars:=cg.(for n:=0:maxn join {cl(n),cc(n),cr(n),cb(n)})$
czero:=part(solve(cvars,cvars),1)$
```

Use various differences of macroscale grid values for the interpolative coupling. `uud(p)` is either $\delta^p U$ or $\mu\delta^p U$ when p is even or odd respectively. We could analyse a low-order interpolation between patches to high-order in coupling parameter γ in order to demonstrate convergence (hopefully).

```
maxint:=floor((maxn+1)/2);
array uud(2*maxint);
```

```
23 uud(0):=uu(j)$ % core value
24 uud(1):=(uu(j+1)-uu(j-1))/2$ % mu*delta
25 for p:=2:2*maxint do uud(p):= % delta^2 of two orders lower
26     sub(j=j+1,uud(p-2))-2*uud(p-2)+sub(j=j-1,uud(p-2))$
```

Recalling the shift operator $\mathcal{E}^{\pm r/2} = (1 \pm \mu\delta + \frac{1}{2}\delta^2)^{r/2}$ so expand this form in a Taylor series and then evaluate.

```
27 f:=taylor((1+eps)^xi,eps,0,maxint)$
28 f:=taylortostandard(f);
29 epxi:=(sub(eps=+mu*delta+delta^2/2,f)
30     where mu^2=>1+delta^2/4)$
31 emxi:=(sub(eps=-mu*delta+delta^2/2,f)
32     where mu^2=>1+delta^2/4)$
33 intr:=sub(xi=r/2,epxi);
34 intl:=sub(xi=r/2,emxi);
```

Then express these formulas as interpolation of the macroscale grid values U_j.

```
35 let gam^2=>gamma*epsilon;
36 uuintr:=uud(0)+sub(delta=uud(1)*gam,(-1+intr
37     where {delta^~^p=>uud(p)*gam^p,mu=>gam}));
38 uuintl:=uud(0)+sub(delta=uud(1)*gam,(-1+intl
39     where {delta^~^p=>uud(p)*gam^p,mu=>gam}));
```

Loop to seek corrections until residuals are smaller than specified order of error. This particular algorithm uses a single order parameter, $\epsilon =$ `epsilon`, for all small parameters (as in above multiplication of `gamma`): it assumes that the solution at each and every order in ϵ is found correctly and so does not need any subsequent refinement. The algorithm proceeds to higher and higher order in ϵ, truncating to the specified orders of error in the parameters, until all residuals are zero. Because we do not truncate in ϵ then the residual calculation is exact to the specified order in parameters.

```
40 let { gamma^2=>0 , alpha^2=>0 };
41 for it:=1:99 do begin
```

Here we add a general form to the unknown fields. These coefficients are to be chosen to eliminate residuals in $\epsilon^{\texttt{it}}$.

```
42 g:=g+cg*epsilon^it;
43 ul:=ul+epsilon^it*(for n:=0:maxn sum cl(n)*xi^n);
44 uc:=uc+epsilon^it*(for n:=0:maxn sum cc(n)*xi^n);
45 ur:=ur+epsilon^it*(for n:=0:maxn sum cr(n)*xi^n);
46 ub:=ub+epsilon^it*(for n:=0:maxn sum cb(n)*xi^n);
```

Compute the thirteen residuals of the advection-diffusion PDE in the four regions, the C^1 continuity conditions, and the patch amplitude condition. In the control use the action regions averages, and their difference with interpolation of core-averages from neighbouring patches. The control may be

easiest to see in terms of its reciprocal, `rmu` $= 1/\mu$.

```
mu:=1/rmu;
ampl:=meanc(uc,xi)-uu(j);
pdel:=-df(ul,t)+kk*df(ul,x,x)-alpha*epsilon*df(ul,x)
    +mu*kk/r^2/hh^2*(uuintl-meanl(ul,xi));
pdec:=-df(uc,t)+kk*df(uc,x,x)-alpha*epsilon*df(uc,x);
pder:=-df(ur,t)+kk*df(ur,x,x)-alpha*epsilon*df(ur,x)
    +mu*kk/r^2/hh^2*(uuintr-meanr(ur,xi));
pdeb:=-df(ub,t)+kk*df(ub,x,x)-alpha*epsilon*df(ub,x);
c0bl:=sub(xi=-3*qr,ul)-sub(xi=5*qr,ub);
c0lc:=sub(xi=-qr,uc-ul);
c0cr:=sub(xi=+qr,ur-uc);
c0rb:=sub(xi=+3*qr,ub-ur);
c1bl:=sub(xi=-3*qr,df(ul,x))-sub(xi=5*qr,df(ub,x));
c1lc:=sub(xi=-qr,df(uc,x)-df(ul,x));
c1cr:=sub(xi=+qr,df(ur,x)-df(uc,x));
c1rb:=sub(xi=+3*qr,df(ub,x)-df(ur,x));
ress:={ampl, pdel, pdec, pder, pdeb, c0bl,
    c0lc, c0cr, c0rb, c1bl, c1lc, c1cr, c1rb};
```

Monitor the progress of the residuals.

```
write lengthress:=map(length(~a),sub(czero,ress));
```

Solve for updates of order $\epsilon^{\mathtt{it}}$ to the field and evolution.

```
eqns:=(foreach res in ress join
    coeff(coeffn(res,epsilon,it),xi));
write lengtheqnsvars:={length(eqns),length(cvars)};
soln:=solve(eqns,cvars)$
if length(soln)=0 then rederr("******* no solution found");
g:=sub(soln,g);
ul:=sub(soln,ul);
uc:=sub(soln,uc);
ur:=sub(soln,ur);
ub:=sub(soln,ub);
```

Exit the loop when all residuals are zero.

```
showtime;
if sub(czero,ress)={0,0,0,0,0,0,0,0,0,0,0,0,0}
   then write it:=it+100000;
end;
```

The single ordering parameter ϵ is no longer needed.

```
epsilon:=1$
```

Find the equivalent PDE to the discretisation where `du(n)` denotes the nth spatial derivative.

```
operator du;
factor hh,r;
erro:=deg((1+gamma)^9,gamma)*2+1;
equivpde:=(g where uu(~k)=>(du(0)+for m:=1:erro+2 sum
                  ((k-j)*hh)^m*du(m)/factorial(m)) )$
for m:=erro:erro do let hh^m=>0;
equivpde:=equivpde;
```

Draw an example of the subpatch fields.

```
load_package gnuplot;
r:=1; rmu:=1/30; kk:=hh:=gamma:=1; alpha:=1;
procedure u(x);
 (if x>3*qr then sub(xi=x,ub)
 else if x>qr then sub(xi=x,ur)
 else if x>-qr then sub(xi=x,uc)
 else sub(xi=x,ul));
operator myu;
let myu(~x,~p)=> coeffn(u(x),uu(j+p),1) when numberp x;
plot(myu(x,0),myu(x,1),myu(x,-1),x=(-3*qr .. 5*qr)
    ,size="0.7,0.5",title="",xlabel="",ylabel=""
    ,terminal="postscript color",output="mmdcmppsX.eps");
```

Fin.

```
end;
```

References

[1] Richard Fateman. "Comparing the speed of programs for sparse polynomial multiplication". In: *ACM SIGSAM Bulletin* 37.1 (2003), pp. 4–15. DOI: 10.1145/844076.844080. URL: http://www.cs.berkeley.edu/~fateman/papers/fastmult.pdf (cit. on p. 4).

Author addresses

1. **Hammad Alotaibi**, School of Mathematical Sciences, University of Adelaide, South Australia.
   mailto:hammad.alotaibi@adelaide.edu.au
   orcid:0000-0002-4798-7119

2. **Barry Cox**, School of Mathematical Sciences, University of Adelaide, South Australia

mailto:barry.cox@adelaide.edu.au
orcid:0000-0002-0662-7037

3. **A. J. Roberts**, School of Mathematical Sciences, University of Adelaide, South Australia
mailto:anthony.roberts@adelaide.edu.au
orcid:0000-0001-8930-1552