Computer algebra derives the slow manifold of macroscale holistic discretisations in two dimensions

Tony MacKenzie* A. J. Roberts†

March 2009

Abstract

Recent developments in dynamical systems theory provides new support for the discretisation of PDEs and other microscale systems. By systematically resolving subgrid microscale dynamics the new approach constructs asymptotically accurate, macroscale closures of the microscale dynamics of the PDE. Here we explore the methodology for problems with two spatial dimensions. The algebraic detail is enormous so we detail computer algebra procedures to handle the complexity. However, only low order models can be constructed purely algebraically; higher order models in 2D appear to require a mixed numerical and algebraic approach that is also detailed. Being based upon the computation of residuals, the procedures here may be simply adapted to a wide class of reaction-diffusion equations.

Contents

*Department of Mathematics and Computing, University of Southern Queensland, Toowoomba, Queensland 4352, Australia.

†Corresponding author: School of Mathematical Sciences, University of Adelaide, South Australia 5005, Australia. mailto:anthony.roberts@adelaide.edu.au

1 Introduction

We extend the dynamical systems, holistic, approach to the macroscale discrete modelling [6, 10, 9, e.g.] to two dimensional, homogeneous, nonlinear reaction-diffusion equations. As a particular example, this report considers in detail the real valued, two dimensional, Ginzburg–Landau equation

$$\frac{\partial u}{\partial t} = \nabla^2 u + \alpha(u - u^3)\,. \tag{1}$$

We choose this 2D real Ginzburg–Landau equation as a prototype PDE because it is well studied and its dynamics well understood [3, 5, e.g.]. This report details the construction by computer algebra of the macroscale discrete model of its dynamics in two spatial dimensions. The general theoretical support, the performance and the physical interpretation are detailed elsewhere.

Place the discrete modelling of two dimensional, reaction-diffusion equations within the purview of centre manifold theory by dividing the domain into overlapping square elements and introducing special interelement coupling conditions. In this initial study, divide the domain into a set of *overlapping*

square elements. Define a grid of points (x_i, y_j) with, for simplicity, constant spacing h. The i,jth element, $E_{i,j}$, is then centred upon (x_i, y_j) and of width $\Delta x = \Delta y = 2h$. Define that $v_{i,j}(x,y,t)$ denotes the field in the i,jth element and so evolves according to the Ginzburg–Landau PDE (1). Using the parameter γ to control the strength of the coupling, use coupling conditions around the i,jth element of

$$v_{i,j}(x_{i\pm1},y,t) = \gamma v_{i\pm1,j}(x_{i\pm1},y,t) + (1-\gamma)v_{i,j}(x_i,y,t), \quad |y-y_j|<h\,, \tag{2}$$

$$v_{i,j}(x,y_{j\pm1},t) = \gamma v_{i,j\pm1}(x,y_{j\pm1},t) + (1-\gamma)v_{i,j}(x,y_j,t), \quad |x-x_i|<h\,. \tag{3}$$

These are a natural extension to 2D of those established for 1D dynamics [9, e.g.]. Then centre manifold theory [1, 2, 4, e.g.] assures us of the existence, relevance and approximation of a slow manifold, macroscale model parametrised by a measure of the amplitude in each element and the coupling strength γ.

Section 2 gives computer algebra that satisfies the PDE (1) and IBCs (2)–(3) to residuals of $\mathcal{O}(\gamma^3+\alpha^3)$. The computer code gives subgrid fields, which are too complex to record here. The corresponding evolution of the grid values on the slow manifold are

$$\begin{aligned}\dot u_{i,j} = {}& \frac{\gamma}{h^2}\delta^2 u_{i,j} + \alpha\left(u_{i,j}-u_{i,j}^3\right)\\ & - \frac{\gamma^2}{12h^2}\delta^4 u_{i,j} + \alpha\gamma\left(\tfrac{1}{12}\delta^2 u_{i,j}^3 - \tfrac14 u_{i,j}^2\delta^2 u_{i,j}\right) + \mathcal{O}\left(\gamma^3+\alpha^3\right),\end{aligned} \tag{4}$$

where the centred difference operator applies in both spatial dimensions,

$$\begin{aligned}\delta^2 u_{i,j} &= u_{i+1,j}+u_{i-1,j}+u_{i,j+1}+u_{i,j-1}-4u_{i,j}\,,\\ \delta^4 u_{i,j} &= u_{i+2,j}+u_{i-2,j}+u_{i,j+2}+u_{i,j-2}\\ &\quad -4(u_{i+1,j}+u_{i-1,j}+u_{i,j+1}+u_{i,j-1})+12u_{i,j}\,,\end{aligned}$$

This model is the extension to two spatial dimensions of the $\mathcal{O}(\gamma^3+\alpha^3)$ holistic model of the 1D Ginzburg–Landau equation [7].

However, although low order accuracy models such as (4) can be constructed algebraically, it appears that higher order models cannot. Thus Section 3

details computer algebra to numerically solve for the microscopic subgrid scale field. Its application here to the discretisation of the Ginzburg–Landau PDE (1) serves as a proof of principle for applying the holistic method to general PDEs of two or more spatial dimensions.

2 Construct the algebraic slow manifold

Computer algebra code to generate *analytic* holistic discretisations of reaction-diffusion PDEs in 2D. Different PDEs are analysed by changing the nonlinear term. Higher order models are constructed by changing the order of neglected terms. Unfortunately, analytic construction can only be carried out to low order accuracy.

All code is written in the computer algebra package REDUCE.[1]

2.1 Initialisation

Set some parameters to improve printing of the results.

```
▷▷ casm2d ◁◁
% see cadsmmd2d.pdf for documentation
on div; off allfac; on revpri;
factor gam,h,x,y,alfa;
```

Subgrid variables The subgrid, intraelement, structures are functions of intraelement microscale variables $\texttt{xi} = (\texttt{x} - \texttt{x}_\texttt{i})/\texttt{h}$ and $\texttt{yi} = (\texttt{y} - \texttt{y}_\texttt{j})/\texttt{h}$.

```
▷▷ casm2d ◁◁+
depend xi,x;
let df(xi,x)=>1/h;
depend yi,y;
```

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

```
let df(yi,y)=>1/h;
```

Parametrise the slow manifold Parametrise the slow manifold, macroscale, evolution by the evolving grid values `u(i,j)` $= u_{i,j}$ such that $du_{i,j}/dt = g$.

```
                        ▷▷ casm2d ◁◁+
operator u;
depend u,t;
let df(u(~k,~m),t)=>sub({i=k,j=m},g);
```

The linear, slow subspace, approximation is that of piecewise constant fields and no evolution.

```
                        ▷▷ casm2d ◁◁+
v:=u(i,j);
g:=0;
```

Set asymptotic truncation Here scale nonlinearity parameter α with parameter γ so we truncate to residuals and errors of order $\mathcal{O}(\alpha^3 + \gamma^3)$.

```
                        ▷▷ casm2d ◁◁+
let gam^3=>0;
alpha:=alfa*gam;
```

General multinomial For the method of undetermined coefficients we set up a general multinomial solution with unknown coefficients `cc(m,n)`, up to order `o` in the intraelement variables. Collect the unknown coefficients in the set `cs`. Operator `ugh` makes it easier to equate coefficients of the multinomial.

```
                    ▷▷ casm2d ◁◁+
o:=6;
operator cc;
vv:=for m:=0:o sum for n:=0:o-m sum cc(m,n)*xi^m*yi^n$
cs:=for m:=0:o join for n:=0:o-m collect cc(m,n)$
operator ugh;
linear ugh;
depend xi,xy; depend yi,xy;
let ugh(xi^~m*yi^~n,xy)=>z^(n-2+(m+n-4)*(m+n-3)/2);
```

2.2 Iteratively construct the slow manifold

Now start the iteration, repeating corrections until all residuals are zero.

```
                    ▷▷ casm2d ◁◁+
it:=0$
repeat begin
```

Compute residuals of governing equations Compute the residual of the reaction-diffusion PDE in the general i,jth element. Could easily change the reaction term to any polynomial in the field v. Could also incorporate advection terms into the microscale PDE.

```
                    ▷▷ casm2d ◁◁+
de:=df(v,t)-df(v,x,2)-df(v,y,2)-alpha*(v-v^3);
```

Compute the residuals of the interelement coupling IBCss.

```
                    ▷▷ casm2d ◁◁+
bcr:=sub(xi=1,v)-sub(xi=0,v)
   -gam*(sub({xi=0,i=i+1},v)-sub(xi=0,v));
```

```
bcl:=sub(xi=-1,v)-sub(xi=0,v)
   -gam*(sub({xi=0,i=i-1},v)-sub(xi=0,v));
bct:=sub(yi=1,v)-sub(yi=0,v)
   -gam*(sub({yi=0,j=j+1},v)-sub(yi=0,v));
bcb:=sub(yi=-1,v)-sub(yi=0,v)
   -gam*(sub({yi=0,j=j-1},v)-sub(yi=0,v));
```

For information as to the progress of the iteration, print out the length of the residuals: when this is five, then all residuals are probably zero.

```
                    ▷▷ casm2d ◁◁+
write lengths:=length(de)+length(bcr)+length(bcl)
    +length(bct)+length(bcb);
```

Add the as yet unknown corrections To find the desired update in each iteration, first substitute the form of the update into the computed residuals.

```
                    ▷▷ casm2d ◁◁+
deq:=de+gd-df(vv,x,2)-df(vv,y,2);
rbcr:=bcr+sub(xi=1,vv)-sub(xi=0,vv);
lbcl:=bcl+sub(xi=-1,vv)-sub(xi=0,vv);
tbct:=bct+sub(yi=1,vv)-sub(yi=0,vv);
bbcb:=bcb+sub(yi=-1,vv)-sub(yi=0,vv);
```

Solve for the corrections Then extract equate coefficients of each power of the multinomial in the intraelement variables.

```
                    ▷▷ casm2d ◁◁+
eqns:=ugh(xi^2*yi^2*deq,xy);
eqns:=append(coeff(eqns,z),cc(0,0).
```

```
    append(coeff(rbcr,yi),append(coeff(lbcl,yi),
    append(coeff(tbct,xi),coeff(bbcb,xi)))));
sol:=solve(eqns,gd.cs);
```

Update Update the field and the evolution, assuming a solution was found (not true for higher orders).

```
                    ▷▷ casm2d ◁◁+
v:=v+sub(sol,vv);
g:=g+sub(sol,gd);
```

Terminate End the iteration when all residuals are zero, or too many iterations have been performed.

```
                    ▷▷ casm2d ◁◁+
showtime;
end until {de,bcr,bcl,bct,bcb}={0,0,0,0,0} or (it:=it+1)>10;
```

2.3 Scrounge an extra order of evolution using solvability

First define the linear integral operator `inthat(a,xi)` $= \int_{-1}^{1}(1-|\xi|)a\,d\xi$ in order to quickly apply the solvability condition.

```
                    ▷▷ casm2d ◁◁+
operator inthat;
linear inthat;
let { inthat(~a^~p,~b)=>0 when (a=b)and(not evenp(p))
, inthat(~a^~p,~b)=>2/(p+1)/(p+2) when (a=b)and evenp(p)
, inthat(~a,~b)=>0 when (a=b)
, inthat(1,~b)=>1
```

```
};
```

Set the requisite *next order* of truncation in the asymptotic expansion by finding the highest order currently retained in coupling γ, then setting to discard terms of two orders higher.

```
                    ▷▷ casm2d ◁◁+
o:=deg((1+gam)^9,gam)$
if o=1 then let gam^3=>0;
if o=2 then let gam^4=>0;
```

Compute exactly the same residuals of the PDE and the interelement coupling conditions.

```
                    ▷▷ casm2d ◁◁+
de:=df(v,t)-df(v,x,2)-df(v,y,2)-alpha*(v-v^3)$
bcr:=sub(xi=1,v)-sub(xi=0,v)
  -gam*(sub({xi=0,i=i+1},v)-sub(xi=0,v))$
bcl:=sub(xi=-1,v)-sub(xi=0,v)
  -gam*(sub({xi=0,i=i-1},v)-sub(xi=0,v))$
bct:=sub(yi=1,v)-sub(yi=0,v)
  -gam*(sub({yi=0,j=j+1},v)-sub(yi=0,v))$
bcb:=sub(yi=-1,v)-sub(yi=0,v)
  -gam*(sub({yi=0,j=j-1},v)-sub(yi=0,v))$
```

Compute the next correction to the evolution g by integrating the residual of the PDE over an element, and including the contributions from the boundary coupling residuals.

```
                    ▷▷ casm2d ◁◁+
gd:=inthat(de,xi)$
gd:=inthat(gd,yi)$
gd:=gd+inthat(bcr+bcl,yi)/h^2+inthat(bct+bcb,xi)/h^2;
g:=g-gd$
```

```
showtime;
```

2.4 Derive the finite difference form

For some reason this code section appears to clash with the code for the equivalent PDE, so only do these sections conditionally via 'if' statements like this.

```
                    ▷▷ casm2d ◁◁+
if 1 then begin
◁◁ finitediff ▷▷
end;
```

Define linear finite difference operators in the two different spatial directions: the spatial direction is indicated by the third letter in the operator name; the first two letters are `md` $= \mu\delta$ or `dd` $= \delta^2$.

```
                    ▷▷ finitediff ◁◁
operator mdx; operator mdy;
operator ddx; operator ddy;
```

These operators only act on the macroscale grid values $u_{i,j}$ so make them linear in the symbol `u`.

```
                    ▷▷ finitediff ◁◁+
linear mdx; linear mdy;
linear ddx; linear ddy;
```

These finite difference operators commute so define the cannonical form to be that the operators in a factor must appear in the order `mdx`, `mdy`, `ddx` and `ddy`. Insist on this order by the following commute rules.

```
                    ▷▷ finitediff ◁◁+
let { mdy(mdx(~a,u),u)=>mdx(mdy(a,u),u)
    , ddx(mdx(~a,u),u)=>mdx(ddx(a,u),u)
    , ddy(mdx(~a,u),u)=>mdx(ddy(a,u),u)
    , ddx(mdy(~a,u),u)=>mdy(ddx(a,u),u)
    , ddy(mdy(~a,u),u)=>mdy(ddy(a,u),u)
    , ddy(ddx(~a,u),u)=>ddx(ddy(a,u),u)
```

Also code the operator rule that $\mu^2 = 1 + \delta^2/4$.

```
                    ▷▷ finitediff ◁◁+
    , mdx(mdx(~a,u),u)=>ddx(a,u)+ddx(ddx(a,u),u)/4
    , mdy(mdy(~a,u),u)=>ddy(a,u)+ddy(ddy(a,u),u)/4
    };
```

Apply the transformation rule, step-by-step, that the spatial shift $E^{\pm 1} = 1 \pm \mu\delta + \delta^2/2$. First do this transformation on powers of variables.

```
                    ▷▷ finitediff ◁◁+
goo:=(g where {u(i,j)=>u
,u(i+~k,~j)^~p=>u(i+k-1,j)^p+mdx(u(i+k-1,j)^p,u)
    +1/2*ddx(u(i+k-1,j)^p,u) when k>0
,u(i+~k,~j)^~p=>u(i+k+1,j)^p-mdx(u(i+k+1,j)^p,u)
    +1/2*ddx(u(i+k+1,j)^p,u) when k<0
,u(~i,j+~k)^~p=>u(i,j+k-1)^p+mdy(u(i,j+k-1)^p,u)
    +1/2*ddy(u(i,j+k-1)^p,u) when k>0
,u(~i,j+~k)^~p=>u(i,j+k+1)^p-mdy(u(i,j+k+1)^p,u)
    +1/2*ddy(u(i,j+k+1)^p,u) when k<0
})$
```

Second do the same shift transformation on the simple variables to give the operator form of the macroscale discrete model in goo.

```
                    ▷▷ finitediff ◁◁+
write goo:=(goo where {u(i,j)=>u
,u(i+~k,~j)=>u(i+k-1,j)+mdx(u(i+k-1,j),u)
    +1/2*ddx(u(i+k-1,j),u) when k>0
,u(i+~k,~j)=>u(i+k+1,j)-mdx(u(i+k+1,j),u)
    +1/2*ddx(u(i+k+1,j),u) when k<0
,u(~i,j+~k)=>u(i,j+k-1)+mdy(u(i,j+k-1),u)
    +1/2*ddy(u(i,j+k-1),u) when k>0
,u(~i,j+~k)=>u(i,j+k+1)-mdy(u(i,j+k+1),u)
    +1/2*ddy(u(i,j+k+1),u) when k<0
});
```

Finish with the compute time.

```
                    ▷▷ finitediff ◁◁+
showtime;
```

2.5 Obtain the equivalent PDE at full coupling

For some reason this code section appears to clash with the code in the previous section, so only do these sections conditionally via 'if' statements like this.

```
                    ▷▷ casm2d ◁◁+
if 0 then begin
◁◁ equivpde ▷▷
end;
```

Print the right-hand side of the slow manifold discretisation $\mathrm{du}_{i,j}/\mathrm{dt} = \mathrm{gg}$ for full coupling.

```
                    ▷▷ equivpde ◁◁
write gg:=sub(gam=1,g);
```

Write the equivalent PDE in terms of a symbol `uu(p,q)` to denote the derivative $\partial^{p+q}u/\partial x^p \partial_y^q$.

```
                    ▷▷ equivpde ◁◁+
operator uu;
```

Replace all grid shifts of `u` by derivatives of `uu`. Compute to order `o` which hopefully is of high enough order to show everything of interest.

```
                    ▷▷ equivpde ◁◁+
o:=8$
gpde:=(gg where u(~k,~l)=> uu(0,0)
+(for q:=1:o sum uu(0,q)*(l-j)^q*h^q/factorial(q))
+(for p:=1:o sum uu(p,0)*(k-i)^p*h^p/factorial(p))
+(for p:=1:o sum for q:=1:o-p sum
  uu(p,q)*(k-i)^p*(l-j)^q*h^(p+q)/factorial(p)/factorial(q))
)$
```

Write out the PDE, omitting high powers of grid spacing h. This truncation and the order `o` have to be changed together.

```
                    ▷▷ equivpde ◁◁+
write gpde:=(gpde where h^7=>0);
```

Finish with the compute time.

```
                    ▷▷ equivpde ◁◁+
showtime;
```

Give the overall grand final 'end' statement, then output all the code.

```
                                ▷▷ casm2d ◁◁+
end;
```

3 Numerically construct the slow manifold

Instead of trying to solve analytically for the subgrid scale structure, here solve for the structure numerically.

3.1 Initialisation

Improve the format of printing.

```
                                ▷▷ ncsm2d ◁◁
% see cadsmmd2d.pdf for documentation
on div; off allfac; on revpri;
factor h,alfa,gam;
```

Load routines to do the LU decomposition, Section 3.3, and subsequent back substitutions, Section 3.4.

```
                                ▷▷ ncsm2d ◁◁+
in "lu_decomp.red"$
in "lu_backsub.red"$
```

The subgrid, microscale, numerical resolution is improved by increasing `n`, needs to be even. The subgrid grid step is `dx` $= h/n$ and the number of equations and unknowns is `nn` $= N = (2n+1)^2 + 1$.

```
                                ▷▷ ncsm2d ◁◁+
n:=2$
```

```
dx:=h/n$
nn:=(2*n+1)^2+1$
```

Define the matrices used to store the subgrid, and to represent and solve the equations. The scope of matrices are global in Reduce.

▷▷ ncsm2d ◁◁+

```
matrix eqns(nn,1);
matrix indx(nn,1);
matrix vv(nn,1);
matrix lu(nn,nn);
matrix v(2*n+1,2*n+1)$
```

Parametrise the slow manifold The slow manifold of the macroscale discretisation is to be parametrised by the evolution of $u_{i,j}$ = `u(i,j)`. The evolution $du_{i,j}/dt = g_{i,j}$ = `g`$/h^2$ and unknown updates are stored in `gd`.

▷▷ ncsm2d ◁◁+

```
operator u;
depend u,t;
let df(u(~k,~m),t)=>sub({i=k,j=m},(g+gd)/h^2);
```

Initialise with constant field in `v` $= v_{i,j}$ and no evolution `g` $= 0$. Also add in the unknown update field `ud`.

▷▷ ncsm2d ◁◁+

```
g:=0$
operator ud;
matrix vd(2*n+1,2*n+1)$
for ii:=1:(2*n+1) do for jj:=1:(2*n+1) do begin
    v(ii,jj):=u(i,j)$
    vd(ii,jj):=ud(ii,jj)$
```

```
end;
v:=v+vd$
```

Set asymptotic truncation Truncate the asymptotic expansion in powers of the coupling parameter γ = `gam`. Here scale the nonlinearity parameter α = `a` with γ to most easily control the truncation. Also ignore the 'updates' when multiplied by the small parameter so that the equations remain linear in the updates.

```
                    ▷▷ ncsm2d ◁◁+
let { gam^4=>0, ud(~i,~j)*gam=>0};
a:=alfa*gam;
```

3.2 Iteratively construct the slow manifold

Repetitively find updates to the subgrid microscale field until all residuals are zero.

```
                    ▷▷ ncsm2d ◁◁+
iter:=0$
repeat begin
iter:=iter+1;
```

For convenience define the fields in the neighbouring elements.

```
                    ▷▷ ncsm2d ◁◁+
vr:=sub(i=i+1,v);
vl:=sub(i=i-1,v);
vt:=sub(j=j+1,v);
vb:=sub(j=j-1,v);
```

In the first iteration we find the equations and then the LU factorisation; in later iterations we just find the residuals. First the amplitude condition that $u_{i,j}$ is the field at he central point of each element.

```
                    ▷▷ ncsm2d ◁◁+
if iter=1 then eqns(1,1):=ud(n+1,n+1) else eqns(1,1):=0;
```

Then set the very corner of the subgrid field to zero as they are not used.

```
                    ▷▷ ncsm2d ◁◁+
if iter=1 then
<< eqns(2,1):=ud(1,1); eqns(3,1):=ud(1,2*n+1);
eqns(4,1):=ud(2*n+1,1); eqns(5,1):=ud(2*n+1,2*n+1) >>
else << eqns(2,1):=0; eqns(3,1):=0;
eqns(4,1):=0; eqns(5,1):=0 >>;
```

Now adjoin the residuals of the interelement coupling conditions, using `eq_count` to count the number of equations.

```
                    ▷▷ ncsm2d ◁◁+
eq_count:=6;
for ll:=2:(2*n) do begin
   eqns(eq_count,1):=v(2*n+1,ll)-v(n+1,ll)
      -gam*(vr(n+1,ll)-v(n+1,ll));
   eq_count:=eq_count+1;
   eqns(eq_count,1):=v(1,ll)-v(n+1,ll)
      -gam*(vl(n+1,ll)-v(n+1,ll));
   eq_count:=eq_count+1;
   eqns(eq_count,1):=v(ll,2*n+1)-v(ll,n+1)
      -gam*(vt(ll,n+1)-v(ll,n+1));
   eq_count:=eq_count+1;
   eqns(eq_count,1):=v(ll,1)-v(ll,n+1)
      -gam*(vb(ll,n+1)-v(ll,n+1));
   eq_count:=eq_count+1;
```

```
end;
```

Adjoin the residuals of the interior subgrid equations which are a simple centred difference approximation to the PDE.

```
                    ▷▷ ncsm2d ◁◁+
for ii:=2:(2*n) do begin
   for jj:=2:(2*n) do begin
      eqns(eq_count,1):=h^2*(df(v(ii,jj),t)
         -(v(ii+1,jj)+v(ii-1,jj)+v(ii,jj+1)
         +v(ii,jj-1)-4*v(ii,jj))/(dx^2)
         -a*v(ii,jj)+a*v(ii,jj)^3);
         eq_count:=eq_count+1;
   end;
end;
```

In the first iteration, the above residuals will incorporate the unknown updates symbolically. Extract these unknowns and assign their coefficients to the `lu` matrix. Then perform the LU decomposition for later use. Lastly remove the symbolic unknowns.

```
                    ▷▷ ncsm2d ◁◁+
if iter=1 then begin
   for ii:=1:nn do begin
      temp:=coeff((-eqns(ii,1)+art*xx^nn where
         {ud(~k,~m)=>xx^(k+(m-1)*(2*n+1)),
         gd=>xx^nn,u(~k,~m)=>0}),xx);
      for jj:=1:nn do
         lu(ii,jj):=part(temp,jj+1);
   end;
   lu:=(lu where art=>0);
   lu_decomp();
   eqns:=(eqns where {ud(~i,~j)=>0});
   gd:=0;
```

```
   v:=(v where ud(~i,~j)=>0);
end;
```

Use the factorisation stored in `lu` to solve for the updates, and assign to the relevant g and v.

```
                    ▷▷ ncsm2d ◁◁+
lu_backsub();
g:=g+eqns(nn,1);
for ii:=1:(2*n+1) do
   for jj:=1:(2*n+1) do
      v(ii,jj):=v(ii,jj)+eqns((jj-1)*(2*n+1)+ii,1);
```

Terminate the loop when all residuals are zero. Check if all the residuals are zero to the specified order of error, by clearing `zero_res` if any residual is non-zero.

```
                    ▷▷ ncsm2d ◁◁+
zero_res:=1;
for ii:=1:nn do if not(eqns(ii,1)=0) then zero_res:=0;
showtime;
end until zero_res;
```

Lastly, restore the factor of h^2 in the evolution.

```
                    ▷▷ ncsm2d ◁◁+
g:=g/h^2;
```

Conditionally derive the finite difference form of the evolution.

```
                    ▷▷ ncsm2d ◁◁+
if 0 then begin
◁◁ finitediff ▷▷
```

```
end;
```

Conditionally derive the equivalent PDE for this model.

```
                    ▷▷ ncsm2d ◁◁+
if 1 then begin
◁◁ equivpde ▷▷
end;
```

Give the grand final 'end' statement, then output all the code.

```
                    ▷▷ ncsm2d ◁◁+
end;
```

3.3 LU decomposition

This LU decomposition is adapted from Numerical recipes in Fortran 77 [8]. Requires matrices `lu(n,n)`, `indx(n,1)` and `eqns(n,1)` to be predefined. Matrices are global in scope in Reduce so we do not bother passing parameters.

```
                    ▷▷ lu_decomp ◁◁
% see cadsmmd2d.pdf for documentation, from Press et al.
procedure lu_decomp;
begin scalar n,np,i,j,k,imax,dd,aamax,dum,sum,tiny;
  tiny:=1.0e-20;
  n:=first(length(lu));
  dd:=1;
  for i:=1:n do begin
    aamax:=abs(lu(i,1));
    for j:=2:n do aamax:=max(aamax,abs(lu(i,j)));
    vv(i,1):=1/aamax;
```

```
    end;
    for j:=1:n do begin
      for i:=1:(j-1) do
        lu(i,j):=lu(i,j)-(for k:=1:(i-1) sum lu(i,k)*lu(k,j));
      aamax:=0;
      for i:=j:n do begin
        lu(i,j):=lu(i,j)-(for k:=1:(j-1) sum lu(i,k)*lu(k,j));
        dum:=vv(i,1)*abs(lu(i,j));
        if dum>=aamax then begin
          imax:=i;
          aamax:=dum;
        end;
      end;
      if not(j=imax) then begin
        for k:=1:n do begin
          dum:=lu(imax,k);
          lu(imax,k):=lu(j,k);
          lu(j,k):=dum;
        end;
        dd:=-dd;
        vv(imax,1):=vv(j,1);
      end;
      indx(j,1):=imax;
      if lu(j,j)=0 then lu(j,j):=tiny;
      if not(j=n) then begin
        dum:=1/lu(j,j);
        for i:=(j+1):n do lu(i,j):=lu(i,j)*dum;
      end;
    end;
  end;
  end;
```

3.4 LU back substitution

This LU back substitution is adapted from Numerical recipes in Fortran 77 [8]. Requires matrices `lu(n,n)`, `indx(n,1)` and `eqns(n,1)` to be predefined: the first two must be obtained from procedure `lu_decomp`; and the last may be algebraic. Matrices are global in scope in Reduce so we do not bother passing parameters.

▷▷ lu_backsub ◁◁

```
% see cadsmmd2d.pdf for documentation, from Press et al.
procedure lu_backsub;
begin scalar n,i,j,ii,ll,sum;
  n:=first(length(lu));
  ii:=0;
  for i:=1:n do begin
    ll:=indx(i,1);
    sm:=eqns(ll,1);
    eqns(ll,1):=eqns(i,1);
    if not(ii=0) then
       sm:=sm-(for j:=ii:(i-1) sum lu(i,j)*eqns(j,1))
    else if not(sm=0) then ii:=i;
    eqns(i,1):=sm;
  end;
  for i:=n step -1 until 1 do eqns(i,1):=(eqns(i,1)
     -(for j:=(i+1):n sum lu(i,j)*eqns(j,1)))/lu(i,i);
end;
end;
```

Acknowledgement The Australian Research Council Discovery Project grants DP0774311 and DP0988738 helped support this research.

References

[1] J. Carr. *Applications of centre manifold theory*, volume 35 of *Applied Math Sci.* Springer-Verlag, 1981.

[2] J. Carr and R.G. Muncaster. The application of centre manifold theory to amplitude expansions. II. infinite dimensional problems. *J. Diff. Eqns*, 50:280–288, 1983.

[3] J.D. Gibbon. Weak and strong turbulence in the complex Ginzburg-Landau equation. In G.R. Sell, C. Foais, and R. Temam, editors, *Turbulence in fluid flows–A dynamical systems approach*, volume 55 of *The IMA volumes in mathematics and its applications*, pages 33–48. Springer-Verlag, 1993.

[4] Y. A. Kuznetsov. *Elements of applied bifurcation theory*, volume 112 of *Applied Mathematical Sciences*. Springer–Verlag, 1995.

[5] C.D. Levermore and M. Oliver. The complex Ginzburg-Landau equation as a model problem. In P. Deift, C.D. Levermore, and C.E. Wayne, editors, *Dynamical systems and probabilistic methods in partial differential equations*, volume 35 of *Lectures in Applied Mathematics*, pages 141–190. American Mathematical Society, 1996.

[6] T. Mackenzie and A. J. Roberts. Holistic finite differences accurately model the dynamics of the Kuramoto–Sivashinsky equation. *ANZIAM J.*, 42(E):C918–C935, 2000. http://anziamj.austms.org.au/V42/CTAC99/Mack.

[7] Tony MacKenzie. *Create accurate numerical models of complex spatio-temporal dynamical systems with holistic discretisation*. PhD thesis, University of Southern Queensland, 2005.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in FORTRAN. The art of scientific computing*. CUP, 2nd edition, 1992. http://www.library.cornell.edu/nr/cbookfpdf.html.

[9] A. J. Roberts. A holistic finite difference approach models linear dynamics consistently. *Mathematics of Computation*, 72:247–262, 2002. http://www.ams.org/mcom/2003-72-241/S0025-5718-02-01448-5.

[10] A.J. Roberts. Holistic discretisation ensures fidelity to Burgers' equation. *Applied Numerical Mathematics*, 37:371–396, 2001.