A general invariant manifold construction procedure, including isochrons of slow manifolds

A. J. Roberts*

Nov 2013 – November 5, 2025

Abstract

This procedure constructs a specified invariant manifold for a specified system of ordinary differential equations or delay differential equations, or partial differential equations where the spatial gradients are small. The invariant manifold may be any of a centre manifold, a slow manifold, an un/stable manifold, a sub-centre manifold, a nonlinear normal form, any spectral submanifold, or indeed a normal form coordinate transform of the entire state space. Thus the procedure may be used to analyse pitchfork bifurcations, or oscillatory Hopf bifurcations, or any more complicated superposition. In the cases when the neglected spectral modes all decay, the constructed invariant manifold supplies a faithful large time model of the dynamics of the differential equations. Further, in the case of a slow manifold, this procedure now derives vectors defining the projection onto the invariant manifold along the isochrons: this projection is needed for initial conditions, forcing, system modifications, and uncertainty quantification.

Section contents

*School of Mathematical Sciences, University of Adelaide, South Australia 5005, Australia. https://profajroberts.github.io/

1 Introduction

Section contents

Installation Download and install the computer algebra package *Reduce* via http://www.reduce-algebra.com Download and unzip the folder https://profajroberts.github.io/InvariantManifold.zip Within the folder `InvariantManifold`, start-up *Reduce* and load the procedure by executing the command `in_tex "invariantManifold.tex"$` [1] Test your installation by then executing `exampleslowman();` (see Section 1.1).

Execution Thereafter, construct a specified invariant manifold of a specific dynamical system by executing the following command with specific values for the input parameters. See `diverseExamples.pdf` for many examples.

```
1 invariantmanifold(spacex, odefns, evals, evecs,
2     adjvecs, toosmall);
```

Inputs As in the example of the next Section 1.1, the input parameters to the procedure are the following:

- `spacex`: for ODEs provide the empty list "`{}`"; for PDEs with large-scale, gradual, variations in some 'space' variables then specify the list of space variables, for example `{x,y}` denotes that there are 'spatial' gradients in variables x, y, gradients that will be treated as small.
- `odefns`, a comma separated list within `mat((...))`, the RHS expressions of the ODEs/DDEs of the system, a system expressed in terms of variables `u1`, `u2`, ..., for time derivatives d`u1`$/dt$, d`u2`$/dt$, ...;

  any time delayed variables in the RHS are coded by the time-delay in parenthesises after the variable, as in the example `u1(pi/2)` to represent $u_1(t-\pi/2)$ in the DDEs;

  any 'spatial' derivatives *must* be specified using a `pdf()` operator (currently with at most five arguments)—examples are `pdf(u1,x)` $= \partial u_1/\partial x$, `pdf(u3,x,y)` $= \partial^2 u_3/\partial x\partial y$, `pdf(u2,x,4)` $= \partial^4 u_2/\partial x^4$.

[1] This script changes many internal settings of *Reduce*, so best done only when needed.

- `evals`, a comma separated list within `mat((...))`, the eigenvalues of the modes to be the basis for the invariant manifold—each eigenvalue may be complex-valued, of the form `a+b*i`;
- `evecs`, a comma separated list of vectors within `mat(...)`—each vector a comma separated list of components within `(...)`, the eigenvectors of the modes corresponding to the given eigenvalues of the invariant manifold basis—possibly complex-valued;
- `adjvecs`, a comma separated list of vectors within `mat(...)`, often the adjoint eigenvectors of the modes corresponding to the given eigenvalues of the invariant manifold basis;
- `toosmall`, an integer giving the desired order of error in the asymptotic approximation that is constructed. The procedure embeds the specified system in a family of systems parametrised by ε, and constructs an invariant manifold, and evolution thereon, of the embedding system to the asymptotic error $\mathcal{O}\bigl(\varepsilon^{\texttt{toosmall}}\bigr)$ (as $\varepsilon \to 0$). Often the introduced artificial ε has a useful physical meaning, but strictly you should evaluate the output at $\varepsilon = 1$ to recover results for the specified system, and then interpret the results in terms of actual 'small' parameters.

Outputs This procedure reports the specified system, the embedded system it actually analyses, the number of iterations taken, the invariant manifold approximation, the evolution on the invariant manifold, and optionally a basis for projecting onto the invariant manifold.

- A plain text report to the Terminal window in which `Reduce` is executing—the invariant manifold is parametrised by variables `s(1)`, `s(2)`, ..., and the dynamics by their evolution in time.
- A LaTeX source report written to the file `invarManReport.tex` (and `invarManReportSys.tex`)—the invariant manifold is parametrised by variables s_1, s_2, ..., and the dynamics by their evolution in time. Generate a pdf report by executing `pdflatex invarManReport`.
- Global variable `uu` gives the constructed invariant manifold such that

`coeffn(uu,e_(i,1),1)` gives the ith coordinate, `ui`, of the invariant manifold as a function of `s(j)`, s_j.

- Global variable `gg` gives the evolution on the invariant manifold, such that `coeffn(gg,e_(j,1),1)` gives the time derivative of `s(j)`, $\dot{s}_j$.
- Global variable `zs` (optional): in the case of a slow manifold (where all specified eigenvalues are zero), `zs` gives the normals to the isochrons at the slow manifold, such that `coeffn(zs,e_(i,j),1)` as a function of $\vec{s}$, is the ith component of the jth normal vector to the isochron.

One may change the appearance of the output somewhat. For example, it is often useful to execute `factor s;` before executing `invariantmanifold(...)` in order to group terms with the same powers of amplitudes/order-parameters/coarse-variables.

Background The theoretical support for the results of the analysis of this procedure is centre/stable/unstable manifold theory (e.g., Carr 1981, Haragus & Iooss 2011, Roberts 2015*b*), and an embryonic backwards theory (Roberts 2022). This particular procedure is developed from a coordinate-independent algorithm for constructing centre manifolds originally by Coullet & Spiegel (1983), adapted for human-efficient computer algebra by Roberts (1997), extended to invariant/inertial manifolds (Roberts 1989*b*, Foias et al. 1988), extended to the evolution of gradually varying patterns in space (Roberts 1988, 2015*a*, Bunder & Roberts 2021); and further extended to the projection of initial conditions, forcing, uncertainty, etc (Roberts 1989*a*, 2000).

We use the computer algebra package *Reduce* [`http://reduce-algebra.com/`] because it is both free and perhaps the fastest general purpose computer algebra system (Fateman 2003, e.g.).

1.1 A simple example: `exampleslowman()`

Execute this example by invoking the command `exampleslowman();` The example system to analyse is specified to be (Roberts 2015*b*, Example 2.1)

$$\dot{u}_1 = -u_1 + u_2 - u_1^2\,, \quad \dot{u}_2 = u_1 - u_2 + u_2^2\,.$$

```
3 procedure exampleslowman;
4     invariantmanifold({},
5     mat((-u1+u2-u1^2,u1-u2+u2^2)),
6     mat((0)),
7     mat((1,1)),
8     mat((1,1)),
9     5)$
```

We seek the slow manifold so specify the eigenvalue zero. From the linearisation matrix $\left[\begin{smallmatrix}-1 & 1\\ 1 & -1\end{smallmatrix}\right]$ a corresponding eigenvector is $\vec{e}=(1,1)$, and a corresponding left-eigenvector is $\vec{z}=\vec{e}=(1,1)$, as specified. The last parameter specifies to construct the slow manifold to errors $\mathcal{O}\big(\varepsilon^5\big)$.

The procedure actually analyses the embedding system, the family of problems,

$$\dot{u}_1=-u_1+u_2-\varepsilon u_1^2\,,\quad \dot{u}_2=u_1-u_2+\varepsilon u_2^2\,.$$

Here the artificial parameter ε has a physical interpretation in that it counts the nonlinearity: a term in ε^p will be a $(p+1)$th order term in $\vec{u}=(u_1,u_2)$. Hence the specified error $\mathcal{O}\big(\varepsilon^5\big)$ is here the same as error $\mathcal{O}\big(|\vec{s}|^6\big)$.

The constructed slow manifold is, in terms of the parameter s_1 (and reverse ordering!),

$$\begin{aligned}u_1&=3/8\varepsilon^3s_1^4-1/2\varepsilon s_1^2+s_1+O\big(\varepsilon^4\big),\\ u_2&=-3/8\varepsilon^3s_1^4+1/2\varepsilon s_1^2+s_1+O\big(\varepsilon^4\big).\end{aligned}$$

On this slow manifold the evolution is

$$\dot{s}_1=-3/4\varepsilon^4s_1^5+\varepsilon^2s_1^3+O\big(\varepsilon^5\big)\,:$$

here the leading term in s_1^3 indicates the origin is unstable. To project initial conditions onto the slow manifold, or non-autonomous forcing, or modifications of the original system, or to quantify uncertainty, use the projection defined by the derived vector

$$\vec{z}_1=\begin{bmatrix}z_{11}\\ z_{12}\end{bmatrix}=\begin{bmatrix}3/2\varepsilon^4s_1^4+3/4\varepsilon^3s_1^3-1/2\varepsilon^2s_1^2-1/2\varepsilon s_1+1/2\\ 3/2\varepsilon^4s_1^4-3/4\varepsilon^3s_1^3-1/2\varepsilon^2s_1^2+1/2\varepsilon s_1+1/2\end{bmatrix}+O\big(\varepsilon^5\big).$$

Evaluate these at $\varepsilon=1$ to apply to the original specified system, or alternatively just interpret ε as a way to count the order of each term.

2 Start of the procedure code

Section contents

Need a couple of things established before defining the procedure: the `rlfi` package; and operator names for the variables of the dynamical system (in case they have delays)—currently code a max of nine variables.

```
10 load_package rlfi;
11 operator u1,u2,u3,u4,u5,u6,u7,u8,u9;
```

Allow 'space' derivatives, to be assumed small, via this `pdf` operator that has the same syntax as `df()` but we need to use separate name in order to delay Reduce processing: `pdf` defined by `rlfi` package. Usually seems best to use the `ezgcd` option—set here in order for a user to optionally turn off before invoking the procedure. This option has mysteriously clashed fatally with `gcd` in some examples (Jul 2025), so appears best without `gcd`. Unfortunately, `ezgcd` also crashes `psl` so leave it off for now (4 Nov 2025). If using *psl*-Reduce, then let's make the algebra case sensitive (as of 2025, *csl* does not seem to permit case sensitive). Remember (currently, 2025) `off latex` automatically turns `raise` on.

```
12 off raise;
```

Now define the procedure as an operator so we can define procedures internally, and may be flexible with its arguments.

```
13 operator invariantmanifold;
```

Four or fewer arguments is an error. If just five arguments, then assume no 'space' variables.

```
14 IMerrorMsg:="invariantmanifold() Should Have Six Arguments"$
15 for all a1,a2,a3,a4 let
16     invariantmanifold(a1,a2,a3,a4) = <<rederr(IMerrorMsg)>>;
```

```
for all a1,a2,a3,a4,a5 let
    invariantmanifold(a1,a2,a3,a4,a5) = begin
        write "**** Warning: ",IMerrorMsg;
        invariantmanifold({},a1,a2,a3,a4,a5);
        rederr(IMerrorMsg);
    end;
```

Define the current procedure with six arguments. Section 9 marks the end of the procedure.

```
for all spacex, odefns, evals, evecs, adjvecs, toosmall
    let invariantmanifold(spacex, odefns, evals, evecs,
        adjvecs, toosmall) = begin
```

2.1 Preamble to the procedure

Operators and arrays are always global, but we can make variables and matrices local, except for matrices that need to be declared `matrix`. So, move to implement all arrays and operators to have underscores, and almost all scalars and most matrices to be declared local here.

```
scalar ff, evalm, ee, zz, maxiter, trace, ll, uvec,
reslin, ok, rhsjact, jacadj, resd, resde, resz, rhsfn,
pp, est, eyem, m, pdfzero;
```

Write an intro message.

```
write "Construct invariant manifold (version 5 Nov 2025)"$
%write input_odefns:=odefns;
%write input_evals:=evals;
%write input_evecs:=evecs;
%write input_adjvecs:=adjvecs;
%write input_toosmall:=toosmall;
```

Transpose the defining matrices so that vectors are columns.

```
ff := tp odefns;
ee := tp evecs;
```

```
37 zz := tp adjvecs;
```

Define default parameters for the iteration: `maxiter` is the maximum number of allowed iterations. Specific problems may override this default.

```
38 maxiter:=29$
```

For optional trace printing of test cases: comment out second line when not needed.

```
39 trace:=0$
40 %trace:=1; maxiter:=5; %%%%
```

The `rationalize` switch makes code much faster with complex numbers. The switch `gcd` seems to wreck convergence for `doubleHofDDE`, so leave it off. But seems OK for all other examples.

```
41 on div; off allfac; on revpri;
42 on rationalize;
```

Use `e_` as basis vector for matrices and vectors. Declare it non-commutative so that multiplication does not commute.

```
43 clear e_; operator e_; noncom e_;
44 factor e_;
45 let { e_(~j,~k)*e_(~l,~p)=>0 when k neq l
46      , e_(~j,~k)*e_(~l,~p)=>e_(j,p) when k=l
47      , e_(~j,~k)^2=>0 when j neq k
48      , e_(~j,j)^2=>e_(j,j) };
```

Also need (once) a transpose operator: do complex conjugation explicitly when needed.

```
49 clear tpe_; operator tpe_; linear tpe_;
50 let tpe_(e_(~i,~j),e_)=>e_(j,i);
```

Empty the output LaTeX file in case of error.

```
51 out "invarManReport.tex";
52 write "This empty document indicates error.";
53 shut "invarManReport.tex";
```

2.2 Check the dimensionality of specified system

Extract dimension information from the parameters of the procedure: seek mD invariant manifold of an nD system.

```
write "total number of variables ",
n:=part(length(ee),1);
write "number of invariant modes ",
m:=part(length(ee),2);
if {length(evals),length(zz),length(ee),length(ff)}
  ={{1,m},{n,m},{n,m},{n,1}}
  then write "Input dimensions are OK"
  else <<write "INCONSISTENT INPUT DIMENSIONS, I EXIT";
      return>>;
```

For the moment limit to a maximum of nine components.

```
if n>9 then <<write "SORRY, MAX NUMBER ODEs IS 9, I EXIT";
    return>>;
```

Need an $m \times m$ identity matrix for normalisation of the isochron projection.

```
eyem:=for j:=1:m sum e_(j,j)$
```

2.3 Define time and optional space dependencies

```
if trace then write "creating depends upon spacex";
for j:=1:9 do begin
    depend mkid(u,j),t;
    foreach z_ in spacex do depend mkid(u,j),z_;
end;
```

Also set rules when needed to omit `pdf` terms, currently coded up to five arguments.

```
pdfzero:={ pdf(~a)=>0, pdf(~a,~b)=>0, pdf(~a,~b,~c)=>0
    , pdf(~a,~b,~c,~d)=>0, pdf(~a,~b,~c,~d,~e)=>0 };
```

3 Dissect the linear part

Section contents

Use the exponential `exp`$(u) = e^u$, but not with the myriad of inbuilt properties so clear it! Do not (yet) invoke the simplification of `exp`(0) as I want it to label modes of no oscillation, zero eigenvalue.

```
clear exp; operator exp;
let { df(exp(~u),t) => df(u,t)*exp(u)
    , exp(~u)*exp(~v) => exp(u+v)
    , exp(~u)^~p => exp(p*u)
    };
```

Also try mapping any user supplied sinusoids into this `exp()` so that we can handle harmonically forced systems. Only invoke on the supplied ODEs as delay differential equations use trig functions. But first test to see if there are any explicit time dependence: whether autonomous or nonautonomous.

```
tmp:=sub(t=t_,ff);  tmp:=df(tmp,t_);
itisAuto_:=1;
for j:=1:n do itisAuto_:=if tmp(j,1)=0 then itisAuto_ else 0;
ff:=(ff where { cos(~u) => (exp(i*u)+exp(-i*u))/2
              , sin(~u) => (exp(i*u)-exp(-i*u))/(2*i)
              } );
```

Need function `conj_` to do parsimonious complex conjugation.

```
procedure conj_(a)$ sub(i=-i,a)$
```

Make an array of eigenvalues for simplicity (`evals` not used hereafter). Substitute `small=0` in the eigenvalues just in case someone wants to detune eigenvalues in the analysis and supply the same parameter in the eigenvalues.

```
clear eval_;  array eval_(m);
for j:=1:m do eval_(j):=sub(small=0,evals(1,j));
```

3.1 Normalise the adjoint eigenvectors

When we include delay differential equations, then we need to account for the history of the eigenvector as well. Hence multiply each eigenvector by its oscillating factor, $e^{i\omega t}$, $e^{\lambda t}$, and then take the mean. This multiplication by its oscillating factor should not make any difference for non-delay equations by the natural orthogonality of left and right eigenvectors of different eigenvalues.

Note: the 'left eigenvectors' have to be the eigenvectors of the complex conjugate transpose, and for the complex conjugate eigenvalue. This seems best: for example, when the linear operator is $\left[\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right]$ then the adjoint and the right eigenvectors are the same.

For oscillations and un/stable manifolds we have to cope with imaginary and with real eigenvalues. Seems to need `zz` to have negative complex conjugated frequency so store in `cexp_`—cannot remember why this appears to work!? It may only work for pure real and for pure imaginary eigenvalues??

```
matrix aa_(m,m),dexp_(m,m),cexp_(m,m);
for j:=1:m do dexp_(j,j):=exp(eval_(j)*t);
for j:=1:m do cexp_(j,j):=exp(-conj_(eval_(j))*t);
aa_:=(tp map(conj_(~b),ee*dexp_)*zz*cexp_ )$
if trace then write aa_:=aa_;
write "Normalising the left-eigenvectors:";
aa_:=(aa_ where {exp(0)=>1, exp(~a)=>0 when a neq 0})$
if trace then write aa_:=aa_;
if det(aa_)=0 then << write
    "ORTHOGONALITY ERROR IN EIGENVECTORS; I EXIT";
    return>>;
zz:=zz*aa_^(-1);
```

3.2 Operator to represent delays

Introduce an operator to represent delay factors more conveniently for analysis. The `exp` rule probably only works for pure imaginary modes!?

```
clear d_; operator d_; linear d_;
let { d_(~a^~p,t,~dt)=>d_(a,t,dt)^p
    , d_(~a*~b,t,~dt)=>d_(a,t,dt)*d_(b,t,dt)
    , d_(exp(~a),t,~dt)=>exp(a)
        *sub(t=-dt,cos(-i*a)+i*sin(-i*a))
    , df(d_(~a,t,~dt),~b)=>d_(df(a,b),t,dt)
    , d_(~a,t,0)=>a
    , d_(d_(~a,t,~dta),t,~dtb)=>d_(a,t,dta+dtb)
    };
```

Now rewrite the (delay) factors in terms of this operator. For the moment limit to a maximum of nine ODEs.

```
if trace then write "setting somerules";
somerules:={}$
somerules:=(u1(~dt)=d_(u1,t,dt)).somerules$
somerules:=(u2(~dt)=d_(u2,t,dt)).somerules$
somerules:=(u3(~dt)=d_(u3,t,dt)).somerules$
somerules:=(u4(~dt)=d_(u4,t,dt)).somerules$
somerules:=(u5(~dt)=d_(u5,t,dt)).somerules$
somerules:=(u6(~dt)=d_(u6,t,dt)).somerules$
somerules:=(u7(~dt)=d_(u7,t,dt)).somerules$
somerules:=(u8(~dt)=d_(u8,t,dt)).somerules$
somerules:=(u9(~dt)=d_(u9,t,dt)).somerules$
ff:=(ff where somerules)$
```

3.3 Linearise at the origin

Assume the equilibrium is at the origin. Find the linear operator at the equilibrium. Include `small=0` as we notionally adjoin it in the list of variables. Do not need to here make any non-zero forcing small at the equilibrium as it gets multiplied by `small` later. (For some reason using `mkid(u,k)=>0` does

not resolve the mkid, but `mkid(u,k)=0` does; however, not clear if it is a problem.)

```
ll:=ee*(tp ee)*0; %zero nxn matrix
uzero:=(for k:=1:n collect (mkid(u,k)=0))$
equilibrium:=(small=0).uzero$
for j:=1:n do for k:=1:n do begin
  ll(j,k):=df(ff(j,1),mkid(u,k));
  ll(j,k):=( sub(equilibrium,ll(j,k)) where pdfzero );
end;
write "I find the linear operator is";
write ll:=ll;
```

We need a vector of unknowns for a little while.

```
uvec:=0*ff; %nx1 zero matrix
for j:=1:n do uvec(j,1):=mkid(u,j);
```

3.4 Eigen-check

Variable `aa_` appears here as the diagonal matrix of eigenvalues. Check that the eigenvalues and eigenvectors are specified correctly.

```
write "Check invariant subspace linearisation ";
for j:=1:m do for k:=1:m do aa_(j,k):=0;
for j:=1:m do aa_(j,j):=eval_(j);
% following maybe only for pure centre modes??
reslin:=(ll*(ee*dexp_)-(ee*dexp_)*aa_
    where exp(~a)*d_(1,t,~dt)
    =>sub(t=-dt,cos(-i*a)+i*sin(-i*a))*exp(a) )$
if trace then write reslin:=reslin;
ok:=1$
for j:=1:n do for k:=1:m do
    ok:=if reslin(j,k)=0 then ok else 0$
if ok then write "Linearisation is OK";
```

Try to find a correction of the linear operator that is ‘close’. Multiply by the

adjoint eigenvectors and then average over time: operator $\mathcal{L}_{\text{new}} := \mathcal{L} - \mathcal{L}_{\text{adj}}$ should now have zero residual. Lastly, correspondingly adjust the ODEs, since `lladj` does not involve delays we do not need delay operator transforms in the product.

```
if not ok then for iter:=1:2 do begin
    write "WARNING: I NEED TO ADJUST LINEAR OPERATOR";
    lladj:=reslin*tp map(conj_(~b),zz*cexp_);
    write
    lladj:=(lladj where {exp(0)=>1, exp(~a)=>0 when a neq 0});
    write
    ll:=ll-lladj;
    % following maybe only for pure centre modes??
    write
    reslin:=(ll*(ee*dexp_)-(ee*dexp_)*aa_
        where exp(~a)*d_(1,t,~dt)
        => sub(t=-dt,cos(-i*a)+i*sin(-i*a))*exp(a) );
    ok:=1$
    for j:=1:n do for k:=1:m do
        ok:=if reslin(j,k)=0 then ok else 0$
    if ok then iter:=iter+1000;
end;
if not ok then << write
    "OOPS, INCONSISTENT EIGENVALUES, EIGENVECTORS AND OPERATOR.
    EMAIL ME; I EXIT";
    return >>;
```

3.5 Ameliorate the nonlinearity

Anything not in the linear operator gets multiplied by `small` to be treated as small in the analysis.

```
somerules:=for j:=1:n collect
  (d_(1,t,~dt)*mkid(u,j)=d_(mkid(u,j),t,dt))$
ll0_:=(ll*uvec where somerules);
ffnonpdf:=(ff where pdfzero);
```

```
168 procedure scalepdf(z_);
169     (z_ where {pdf(~a,~b)=>small*df(a,b)
170         , pdf(~a,~b,~n)=>small^n*df(a,b,n) when fixp(n)
171         , pdf(~a,~b,~c)=>small^2*df(a,b,c) when not fixp(c)
172         });
173 ffpdf:=map(scalepdf(~z),ff-ffnonpdf);
174 ff:=small*ffnonpdf
175     +(1-small)*ll0_
176     +ffpdf;
```

Any constant term in the equations `ff` has to be multiplied by `exp(0)`.

```
177 ff:=ff+(exp(0)-1)*sub(uzero,ff)$
```

From the matrix versions of the equations, create algebraic form using the matrix basis.

```
178 rhsfn:=for i:=1:n sum e_(i,1)*ff(i,1)$
179 if trace then write "rhsfn= ",rhsfn;
```

Also, create the algebraic form of the jacobian transpose using the matrix basis: take the conjugate later when used.

```
180 rhsjact:=for i:=1:n sum for j:=1:n sum
181     e_(j,i)*df(ff(i,1),mkid(u,j));
```

3.6 Store invariant manifold eigenvalues

Extract all the eigenvalues in the invariant manifold, and the set of all the corresponding modes in the invariant manifold variables. The slow modes are accounted for as having zero eigenvalue. Remember the eigenvalue set is not in the 'correct' order. Array `modes_` stores the set of indices of all the modes of a given eigenvalue.

```
182 clear eval_s,modes_;
183 array eval_s(m),modes_(m);
184 neval:=0$ eval_set:={}$
185 for j:=1:m do if not(eval_(j) member eval_set) then begin
```

```
186   neval:=neval+1;
187   eval_s(neval):=eval_(j);
188   eval_set:=eval_(j).eval_set;
189   modes_(neval):=for k:=j:m join
190     if eval_(j)=eval_(k) then {k} else {};
191 end;
```

Set a flag for the case of a slow manifold when all eigenvalues are zero, as then we compute the isochron projection. The next challenge is to get this isochron code working for the case of non-slow invariant manifolds.

```
192 itisAutoSlowMan_:=if eval_set={0} then itisAuto_ else 0$
193 if trace then write "itisAutoSlowMan_= ",itisAutoSlowMan_;
```

Put in the non-singular general case as the zero entry of the arrays.

```
194 eval_s(0):=geneval$
195 modes_(0):={}$
```

3.7 Precompute matrices for updates

Precompute matrices to solve for updates for each of the critical eigenvalues, and the general case `k` = 0. The matrix

$$\texttt{llzz\_} = \begin{bmatrix} -\mathcal{L}+\partial_t & \mathcal{E}_0 \\ \mathcal{Z}_0^\dagger & 0 \end{bmatrix}$$

and then put its inverse in place. Subsequently, extract the blocks for the generalised inverses and solvability condition into `l_invs` and `g_invs`.

```
196 matrix llzz_(n+m,n+m);
197 clear l_invs,g_invs,l1_invs,g1_invs,l2_invs,g2_invs;
198 array l_invs(neval), g_invs(neval), l1_invs(neval),
199       g1_invs(neval), l2_invs(neval), g2_invs(neval);
200 clear sp_; operator sp_; linear sp_;
201 for k_:=0:neval do begin
202   if trace then write "ITERATION ",k_;
```

Code the operator $\mathcal{L}\hat{v}$ where the delay is to only act on the oscillation part.

```
for ii:=1:n do for jj:=1:n do llzz_(ii,jj):=(
    -sub(small=0,ll(ii,jj)) where d_(1,t,~dt)
    => cos(i*eval_s(k_)*dt)+i*sin(i*eval_s(k_)*dt));
```

Code the operator $\partial\hat{v}/\partial t$ where it only acts on the oscillation part.

```
for j:=1:n do llzz_(j,j):=eval_s(k_)+llzz_(j,j);
```

Now code the part leading to the solvability condition which arises from allowing the (oscillation) amplitude to evolve. Use operator `sp_` to extract the delay parts that subtly affect the updates of the evolution.

```
for j:=1:length(modes_(k_)) do
  for ii:=1:n do llzz_(ii,n+j):=ee(ii,part(modes_(k_),j))
   +(for jj:=1:n sum
     sp_(ll(ii,jj)*ee(jj,part(modes_(k_),j)),d_)
     where { sp_(1,d_)=>0
           , sp_(d_(1,t,~dt),d_)=>dt*(
             cos(i*eval_s(k_)*dt)+i*sin(i*eval_s(k_)*dt))
           });
```

Force the updates to be orthogonal to the left-eigenvectors in the complex conjugate transpose adjoint.

```
for ii:=1:length(modes_(k_)) do for j:=1:n do
    llzz_(n+ii,j):=conj_(zz(j,part(modes_(k_),ii)));
if trace then write
"finished Force the updates to be orthogonal";
```

Set the bottom-right corner of the matrix to zero.

```
for i:=1:length(modes_(k_)) do
  for j:=1:m do llzz_(n+i,n+j):=0;
```

Add some trivial rows and columns to make the matrix up to the same size for all eigenvalues.

```
for i:=length(modes_(k_))+1:m do begin
  for j:=1:n+i-1 do llzz_(n+i,j):=llzz_(j,n+i):=0;
  llzz_(n+i,n+i):=1;
```

```
224   end;
225   if trace then write
226   "finished Add some trivial rows and columns";
```

Invert the matrix and unpack into arrays ready for use by the inversion operators.

```
227   if trace then write "llzz_= ",llzz_;
228   llzz_:=llzz_^(-1);
229   if trace then write "llzz_= ",llzz_;
230   l_invs(k_):=for i:=1:n sum for j:=1:n sum e_(i,j)*llzz_(i,j);
231   g_invs(k_):=for i:=1:length(modes_(k_)) sum
232     for j:=1:n sum e_(part(modes_(k_),i),j)*llzz_(i+n,j);
233   if trace then write "finished Invert the matrix and unpack";
```

Unpack the conjugate transpose for inverse operators used for the isochrons. A difference here is that the orthogonality condition is non-trivial (in the slow manifold we assumed amplitudes were exactly orthogonal to the left-eigenvectors), so we need to remember more parts of the inverse of the matrix.

```
234   l1_invs(k_) := for ii:=1:n sum for j:=1:n sum
235       e_(ii,j)*conj_(llzz_(j,ii));
236   if trace then write "l1_invs",k_," =",l1_invs(k_);
237   l2_invs(k_) := for ii:=1:n sum
238       for j:=1:length(modes_(k_)) sum
239           e_(ii,part(modes_(k_),j))*conj_(llzz_(j+n,ii));
240   if trace then write "l2_invs",k_," =",l2_invs(k_);
241   g1_invs(k_) := for ii:=1:length(modes_(k_)) sum
242       for j:=1:n sum
243           e_(part(modes_(k_),ii),j)*conj_(llzz_(j,ii+n));
244   if trace then write "g1_invs",k_," =",g1_invs(k_);
245   g2_invs(k_) := for ii:=1:length(modes_(k_)) sum
246       for j:=1:length(modes_(k_)) sum
247           e_(part(modes_(k_),ii),part(modes_(k_),j))
248           *conj_(llzz_(j+n,ii+n));
249   if trace then write "g2_invs",k_," =",g2_invs(k_);
```

```
250   if trace then write
251   "finished Unpack the conjugate transpose";
252 end;
```

3.8 Define operators that invoke these inverses

Decompose residuals into parts, and operate on each. First for the invariant manifold. But making `e_` non-commutative means that it does not get factored out of these linear operators: must post-multiply by `e_` because the linear inverse is a premultiply.

```
253 clear l_inv; operator l_inv; linear l_inv;
254 let l_inv(e_(~j,~k)*exp(~a_),exp) => l_invproc(a_/t)*e_(j,k);
255 procedure l_invproc(a_);  begin scalar tmp_;
256   if a_ member eval_set
257   then << k_:=0;
258     repeat k_:=k_+1 until a_=eval_s(k_);
259     tmp_:=l_invs(k_)*exp(a_*t) >>
260   else tmp_:=sub(geneval=a_,l_invs(0))*exp(a_*t)$
261   return tmp_;
262   end;
```

Second for the evolution on the invariant manifold.

```
263 clear g_inv; operator g_inv; linear g_inv;
264 let g_inv(e_(~j,~k)*exp(~a_),exp)=>ginv_proc(a_/t)*e_(j,k);
265 procedure ginv_proc(a_);  begin scalar tmp_;
266   if a_ member eval_set
267   then << k_:=0;
268     repeat k_:=k_+1 until a_=eval_s(k_);
269     tmp_:=g_invs(k_) >>
270   else tmp_:=sub(geneval=a_,g_invs(0))$
271   return tmp_;
272   end;
```

Copy and adjust the above for the projection. But first define the generic procedure.

```
273 procedure inv_proc(a_,invs);  begin scalar tmp_;
274   if a_ member eval_set
275   then << k_:=0;
276     repeat k_:=k_+1 until a_=eval_s(k_);
277     tmp_:=invs(k_)*exp(a_*t) >>
278   else tmp_:=sub(geneval=a_,invs(0))*exp(a_*t)$
279   return tmp_;
280   end;
```

Then define operators that we use to update the projection.

```
281 clear l1_inv; operator l1_inv; linear l1_inv;
282 clear l2_inv; operator l2_inv; linear l2_inv;
283 clear g1_inv; operator g1_inv; linear g1_inv;
284 clear g2_inv; operator g2_inv; linear g2_inv;
285 let { l1_inv(e_(~j,~k)*exp(~a_),exp)
286       => inv_proc(a_/t,l1_invs)*e_(j,k)
287     , l2_inv(e_(~j,~k)*exp(~a_),exp)
288       => inv_proc(a_/t,l2_invs)*e_(j,k)
289     , g1_inv(e_(~j,~k)*exp(~a_),exp)
290       => inv_proc(a_/t,g1_invs)*e_(j,k)
291     , g2_inv(e_(~j,~k)*exp(~a_),exp)
292       => inv_proc(a_/t,g2_invs)*e_(j,k)
293     };
```

3.9 Projection onto slow manifold for PDE case

Now that we have finished with `pdf`, repurpose it for the bare differentiation operator needed for the projection vectors. This seems to work for 1-D space PDEs, but needs checking.

```
294 operator diff;
295 if spacex neq {} then
296 let { df(~a,~x,~b)=>diff(x) when a=b
297     , df(~a,~x,~n,~b)=>diff(x)^n when a=b and fixp(n)
298     , df(~a,~x,~y,~b)=>diff(x)*diff(y)
```

```
299         when a=b and not fixp(y)
300     , df(~a,~x,~n,~y,~b)=>diff(x)^n*diff(y)
301         when a=b and fixp(n) and not fixp(y)
302     , df(~a,~x,~y,~n,~b)=>diff(x)*diff(y)^n
303         when a=b and not fixp(y) and fixp(n)
304     , df(~a,~x,~y,~z,~b)=>diff(x)*diff(y)*diff(z)
305         when a=b and not fixp(y) and not fixp(z)
306     };
```

4 Initialise LaTeX output

Define the Greek alphabet with `small` as well.

```
307 defid small,name="\eps";%varepsilon;
308 defid alpha,name=alpha;
309 defid beta,name=beta;
310 defid gamma,name=gamma;
311 defid delta,name=delta;
312 defid epsilon,name=epsilon;
313 defid varepsilon,name=varepsilon;
314 defid zeta,name=zeta;
315 defid eta,name=eta;
316 defid theta,name=theta;
317 defid vartheta,name=vartheta;
318 defid iota,name=iota;
319 defid kappa,name=kappa;
320 defid lambda,name=lambda;
321 defid mu,name=mu;
322 defid nu,name=nu;
323 defid xi,name=xi;
324 defid pi,name=pi;
325 defid varpi,name=varpi;
326 defid rho,name=rho;
327 defid varrho,name=varrho;
328 defid sigma,name=sigma;
```

```
defid varsigma,name=varsigma;
defid tau,name=tau;
defid upsilon,name=upsilon;
defid phi,name=phi;
defid varphi,name=varphi;
defid chi,name=chi;
defid psi,name=psi;
defid omega,name=omega;
defid Gamma,name=Gamma;
defid Delta,name=Delta;
defid Theta,name=Theta;
defid Lambda,name=Lambda;
defid Xi,name=Xi;
defid Pi,name=Pi;
defid Sigma,name=Sigma;
defid Upsilon,name=Upsilon;
defid Phi,name=Phi;
defid Psi,name=Psi;
defid Omega,name=Omega;
```

For the variables names I use, as operators, define how they appear in the LaTeX, and also define that their arguments appear as subscripts.

```
defindex diff(down);
defid diff,name="\diff";
defindex e_(down,down);
defid e_,name="e";
defindex d_(arg,down,down);
defid d_,name="D";
defindex u(down);
defid u1,name="u_1";
defid u2,name="u_2";
defid u3,name="u_3";
defid u4,name="u_4";
defid u5,name="u_5";
defid u6,name="u_6";
```

```
defid u7,name="u_7";
defid u8,name="u_8";
defid u9,name="u_9";
defindex s(down);
defid exp,name="\exp";
defindex exp(arg);
```

Change printing of all derivatives to partials (hacked from `rlfi.red`).

```
put('df,'lapop,"\partial ")$
```

Can we write the system? Not in matrices apparently. So define a dummy array `tmp_` that we use to get the correct symbol typeset.

```
clear tmp_,tmp_s,tmp_z;
array tmp_(n),tmp_s(m),tmp_z(m);
defindex tmp_(down);
defindex tmp_s(down);
defindex tmp_z(down);
defid tmp_,name="\dot u";
defid tmp_s,name="\vec e";
defid tmp_z,name="\vec z";
rhs_:=rhsfn$
for k:=1:m do tmp_s(k):=
    {for j:=1:n collect ee(j,k),exp(eval_(k)*t)};
for k:=1:m do tmp_z(k):=
    {for j:=1:n collect zz(j,k),exp(eval_(k)*t)};
```

We have to be shifty here because `rlfi` does not work inside a loop: so write the commands to a file, and then input the file. Unclear why `off echo` needed after 2021.

```
out "scratchfile.red";
write "off echo;"$
write "write ""\)
\IfFileExists{cmsys.red}{\verbatiminput{cmsys.red}}{}
\paragraph{The specified dynamical system}
\("";";
```

```
for j:=1:n do write "tmp_(",j,"):=
    coeffn(rhs_,e_(",j,",1),1);";
write "write ""\)
\paragraph{Invariant subspace basis vectors}
\("";";
for j:=1:m do write "tmp_s(",j,"):=tmp_s(",j,");";
for j:=1:m do write "tmp_z(",j,"):=tmp_z(",j,");";
write "end;";
shut "scratchfile.red";
```

Now print the dynamical system to the LaTeX sub-file. Switch `raise` back off.

```
write "Ignore the following 13 lines of LaTeX"$
on latex$
out "invarManReportSys.tex"$
in "scratchfile.red"$
shut "invarManReportSys.tex"$
off latex$  off raise$
```

5 Linear approximation to the invariant manifold

But first, write out the possibly adjusted nonlinear right-hand side function. According to the manual, this will append to the earlier output to the file.

```
write "Analyse ODE/DDE system du/dt = ",ff;
```

Parametrise the invariant manifold in terms of these amplitudes. For this substitution to work, `gg` *cannot* be declared scalar as then it gets replaced by zero here and throughout. Let `gg` be global so a user can access the time derivative expressions afterwards, similarly for `uu` the constructed invariant manifold.

```
clear gg;
clear s; operator s; depend s,t;
foreach z_ in spacex do depend s,z_;
let df(s(~j),t)=>coeffn(gg,e_(j,1),1);
```

Invoke the following procedure to substitute whatever the current approximation is into (nonlinear) expressions.

```
407 procedure manifold_(uu,n);
408     for j:=1:n collect mkid(u,j)=coeffn(uu,e_(j,1),1)$
```

The linear approximation to the invariant manifold must be the following corresponding to the eigenvalues down the diagonal (even if zero). The amplitudes s_j are slowly evolving as they are either slow modes, or the complex amplitudes of oscillating modes.

```
409 uu:=for j:=1:m sum s(j)*exp(eval_(j)*t)
410   *(for k:=1:n sum e_(k,1)*ee(k,j));
411 gg:=0;
412 if trace then write uu:=uu;
```

For some temporary trace printing, where for simplicity `small` is replaced by `s`.

```
413 procedure matify_(a,m,n)$
414   begin matrix z_(m,n);
415     for i:=1:m do for j:=1:n do z_(i,j):=coeffn(a,e_(i,j),1);
416     return (z_ where {exp(0)=>1,small=>s});
417     end$
```

For the isochron may need to do something different with eigenvalues, but this should work as the inner product is complex conjugate transpose. The `pp` matrix is proposed to place the projection residuals in the range of the isochron.

```
418 zs:=for j:=1:m sum exp(eval_(j)*t)
419   *(for k:=1:n sum e_(k,j)*zz(k,j))$
420 pp:=0$
```

6 Iteratively construct the invariant manifold

But first establish the Taylor series in any delay factors of slow amplitudes.

```
let d_(s(~k),t,~dt)=>s(k)+(for n:=1:toosmall sum
        (-dt)^n*df(s(k),t,n)/factorial(n));
```

Truncate expansions to specified order of error (via loop index trick).

```
for j:=toosmall:toosmall do let small^j=>0;
```

Iteratively construct the invariant manifold.

```
write "Start iterative construction of invariant manifold";
for iter:=1:maxiter do begin
if trace then write "
ITERATION = ",iter,"
-------------";
```

Compute residual vector (matrix) of the dynamical system Roberts (1997).

```
resde:=-df(uu,t)+sub(manifold_(uu,n),rhsfn);
if trace then write "resde=",matify_(resde,n,1);
```

Get the local directions of the coordinate system on the curving manifold: store transpose as $m \times n$ matrix.

```
est:=tpe_(for j:=1:m sum df(uu,s(j))*e_(1,j),e_);
est:=conj_(est);
if trace then write "est=",matify_(est,m,n);
```

Compute residual matrix for the isochron projection Roberts (1989*a*, 2000). But for the moment, only do it if the `eval_set` is for slow manifolds.

```
if itisAutoSlowMan_ then begin
    jacadj:=conj_(sub(manifold_(uu,n),rhsjact));
    if trace then write "jacadj=",matify_(jacadj,n,n);
    resd:=df(zs,t)+jacadj*zs+zs*pp;
    if trace then write "resd=",matify_(resd,n,m);
```

Compute residual of the normalisation of the projection.

```
    resz:=est*zs-eyem*exp(0);
    if trace then write "resz=",matify_(resz,m,m);
end %if itisAutoSlowMan_
```

```
442 else resd:=resz:=0; % for when not slow manifold
```

Write lengths of residuals as a trace print (remember that the expression 0 has length one).

```
443 write lengthRes:=map(length(~a),{resde,resd,resz});
```

Solve for updates—all the hard work is already encoded in the operators.

```
444 uu:=uu+l_inv(resde,exp);
445 if trace then write "uu=",matify_(uu,n,1);
446 gg:=gg+g_inv(resde,exp);
447 if trace then write "gg=",matify_(gg,m,1);
```

Now update the isochron projection, with normalisation.

```
448 if itisAutoSlowMan_ then begin
449 if trace then write "starting zs evaluation";
450 zs:=zs+l1_inv(resd,exp)-l2_inv(resz,exp);
451 if trace then write "zs=",matify_(zs,n,m);
452 pp:=pp-g1_inv(resd,exp)+youshouldnotseethis*g2_inv(resz,exp);
453 if trace then write "pp=",matify_(pp,m,m);
454 end;%if itisAutoSlowMan_
```

Terminate the iteration loop once residuals are zero.

```
455 %showtime;
456 if {resde,resd,resz}={0,0,0} then write iter:=iter+10000;
457 end;
458 showtime;
459 if not itisAutoSlowMan_ then clear zs;
```

Only proceed to print if terminated successfully.

```
460 if {resde,resd,resz}={0,0,0}
461   then write "SUCCESS: converged to an expansion"
462   else <<write "FAILED TO CONVERGE; I EXIT";
463     return; >>;
```

7 Output text report of results

Once construction is finished, simplify `exp(0)`.

```
464 let exp(0)=>1;
```

Invoking switch `complex` improves some of the output of the complex numbers, but wrecks other parts of the output. Best left off.

Write text results.

```
465 write "The invariant manifold is (to one order lower)";
466 for j:=1:n do write "u",j," = ",
467   coeffn(small*uu,e_(j,1),1)/small;
468 write "The evolution of the real/complex amplitudes";
469 for j:=1:m do write "ds(",j,")/dt = ",
470   coeffn(gg,e_(j,1),1);
```

Optionally write the projection vectors.

```
471 if itisAutoSlowMan_ then begin write "
472 The normals to the isochrons at the slow manifold.
473 Use these vectors: to project initial conditions
474 onto the slow manifold; to project non-autonomous
475 forcing onto the slow evolution; to predict the
476 consequences of modifying the original system; in
477 uncertainty quantification to quantify effects on
478 the model of uncertainties in the original system.";
479   for j:=1:m do write "z",j," = ",
480     for i:=1:n collect coeffn(zs,e_(i,j),1);
481 end;
```

Write text results numerically evaluated when expressions are long.

```
482 if length(gg)>30 then begin
483 on rounded; print_precision 4$
484 write "Numerically, the invariant manifold is (to one order lowe
485 for j:=1:n do write "u",j," = ",
486   coeffn(small*uu,e_(j,1),1)/small;
```

```
write "Numerically, the evolution of the real/complex amplitudes'
for j:=1:m do write "ds(",j,")/dt = ",
  coeffn(gg,e_(j,1),1);
if itisAutoSlowMan_ then begin
  write "Numerically, normals to isochrons at slow manifold.";
  for j:=1:m do write "z",j," = ",
    for i:=1:n collect coeffn(zs,e_(i,j),1);
end;
off rounded;
end;
```

8 Output LaTeX report of results

Change the printing of temporary arrays.

```
clear tmp_zz; array tmp_zz(m,n);
defid tmp_,name="u";
defid tmp_s,name="\dot s";
defid tmp_z,name="\vec z";
operator zs_;%(m,n);
defid zs_,name="z";
defindex zs_(down,down);
```

Gather complicated result

```
for k:=1:m do for j:=1:n do
    tmp_zz(k,j):=(1*coeffn(zs,e_(j,k),1));
```

Include order of error to make printing more robust. But we cannot use `small^toosmall` in the following as that is set to zero (for the asymptotics), so we hard code that `small` appears as `varepsilon` ε.

```
clear order_;  operator order_;
defid order_,name="O";
defindex order_(arg);
```

Write to a file the commands needed to write the LaTeX expressions. Write

the invariant manifold to one order lower than computed.

```
out "scratchfile.red";
write "off echo;"$ % do not understand why needed in 2021??
write "write ""\)
\paragraph{The invariant manifold}
These give the location of the invariant manifold in
terms of parameters~\(s_ j\).
\("";";
for j:=1:n do write "tmp_(",j,"):=coeffn(small*uu,e_(",j,
        ",1),1)/small +order_(varepsilon^",toosmall-1,");";
if length(gg)>30 then begin
write "on rounded; print_precision 4$"$
for j:=1:n do write "tmp_(",j,"):=coeffn(small*uu,e_(",j,
        ",1),1)/small +order_(varepsilon^",toosmall-1,");";
write "off rounded;"$
end;
```

Write the commands to write the ODEs on the invariant manifold.

```
write "write ""\)
\paragraph{Invariant manifold ODEs}
The system evolves on the invariant manifold such
that the parameters evolve according to these ODEs.
\("";";
for j:=1:m do write "tmp_s(",j,"):=1*coeffn(gg,e_(",j
           ,",1),1)+order_(varepsilon^",toosmall,");";
if length(gg)>30 then begin
write "on rounded; print_precision 4$"$
for j:=1:m do write "tmp_s(",j,"):=1*coeffn(gg,e_(",j
           ,",1),1)+order_(varepsilon^",toosmall,");";
write "off rounded;"$
end;
```

Optionally write the commands to write the projection vectors on the slow manifold.

```
if itisAutoSlowMan_ then begin
  write "write ""\)
\paragraph{Normals to isochrons at the slow manifold}
Use these vectors: to project initial conditions
onto the slow manifold; to project non-autonomous
forcing onto the slow evolution; to predict the
consequences of modifying the original system; in
uncertainty quantification to quantify effects on
the model of uncertainties in the original system.
The normal vector \(\vec z_j:=(z_{j1},\ldots,z_{jn})\)
\("";";
  for i:=1:m do for j:=1:n do
  write "zs_(",i,",",j,"):=tmp_zz(",i,",",j
       ,")+order_(varepsilon^",toosmall,");";
end;%if itisAutoSlowMan_
```

Finish the scratchfile.

```
write ";end;";
shut "scratchfile.red";
```

Execute the scratchfile with the required commands, with output to the main invariant manifold LaTeX file. `revpri` makes no difference to the LaTeX output! Switch `raise` back off.

```
out "invarManReport.tex"$
on latex$
in "scratchfile.red"$
off latex$ off raise$
shut "invarManReport.tex"$
```

9 Fin

That's all folks, so end the procedure.

```
return Finished_constructing_invariant_manifold_of_system$
end$
```

10 Override some system procedures

Bad luck if these interfere with anything else a user might try to do afterwards!

First define how various tokens get printed. Package `rlfi` must already be loaded.

```
561 deflist('((!( !\!b!i!g!() (!) !\!b!i!g!)) (!P!I !\!p!i! )
562          (!p!i !\!p!i! ) (!E !e) (!I !i) (e !e) (i !i)),'name)$
```

Override the procedure that prints annoying messages about multicharacter symbols. It ends the output of one expression. This is just a copy from `rlfi.red` with the appropriate if-statement deleted. While interfering, hardcode that the mathematics is in inline mode.

```
563 symbolic procedure prinlaend;
564 <<terpri();
565   prin2t "\)\par";
566   if !*verbatim then
567       <<prin2t "\begin{verbatim}";
568         prin2t "REDUCE Input:">>;
569   ncharspr!*:=0;
570   if ofl!* then linelength(car linel!*)
571     else laline!*:=cdr linel!*;
572   nochar!*:=append(nochar!*,nochar1!*);
573   nochar1!*:=nil >>$
574   %
```

Similarly, hardcode at the beginning of expression output that the mathematics is in inline mode.

```
575 symbolic procedure prinlabegin;
576 <<if !*verbatim then
577       <<terpri();
578         prin2t "\end{verbatim}">>;
579   linel!*:=linelength nil . laline!*;
580   if ofl!* then linelength(laline!* + 2)
581     else laline!*:=car linel!* - 2;
```

```
582   prin2 "\(" >>$
```

Override the procedure that outputs the LaTeX preamble upon the command `on latex`. Modified from that in `rlfi.red`. Use it to write a decent header that we use for one master file.

```
583 symbolic procedure latexon;
584 <<!*!*a2sfn:='texaeval;
585   !*raise:=nil;
586   prin2t "\documentclass[11pt,a5paper]{article}";
587   prin2t "\usepackage[a5paper,margin=13mm]{geometry}";
588   prin2t "\usepackage{parskip,time,verbatim} \def\_{_}";
589   prin2t "\raggedright \def\eps{\varepsilon}";
590   prin2t "\def\diff{\partial\makebox[0pt]{\ $\cdot$}}";
591   prin2t "\title{Invariant manifold of your dynamical system}";
592   prin2t "\author{A. J. Roberts, University of Adelaide\\";
593   prin2t "\texttt{http://orcid.org/0000-0001-8930-1552}}";
594   prin2t "\date{\now, \today}";
595   prin2t "\begin{document}";
596   prin2t "\maketitle";
597   prin2t "Throughout and generally: the lowest order, most";
598   prin2t "important, terms are near the end of each expression.";
599   prin2t "\par\leftskip=2em  \parindent=-2em";
600   prin2t "\input{invarManReportSys}";
601   if !*verbatim then
602       <<prin2t "\begin{verbatim}";
603         prin2t "REDUCE Input:">>;
604   put('tex,'rtypefn,'(lambda(x) 'tex)) >>$
```

End the file when read by Reduce

```
605 end;
```

References

Bunder, J. E. & Roberts, A. J. (2021), 'Nonlinear emergent macroscale PDEs, with error bound, for nonlinear microscale systems', *Springer*

Nature Applied Sciences **3**(703), 1–28.

Carr, J. (1981), *Applications of centre manifold theory*, Vol. 35 of *Applied Math. Sci.*, Springer–Verlag.
`http://books.google.com.au/books?id=93BdN7btysoC`

Coullet, P. H. & Spiegel, E. A. (1983), 'Amplitude equations for systems with competing instabilities', *SIAM J. Appl. Math.* **43**, 776–821.

Fateman, R. (2003), 'Comparing the speed of programs for sparse polynomial multiplication', *ACM SIGSAM Bulletin* **37**(1), 4–15.
`http://www.cs.berkeley.edu/~fateman/papers/fastmult.pdf`

Foias, C., Jolly, M. S., Kevrekidis, I. G., Sell, G. R. & Titi, E. S. (1988), 'On the computation of inertial manifolds', *Physics Letters A* **131**, 433–436.

Haragus, M. & Iooss, G. (2011), *Local Bifurcations, Center Manifolds, and Normal Forms in Infinite-Dimensional Dynamical Systems*, Springer.

Roberts, A. J. (1988), 'The application of centre manifold theory to the evolution of systems which vary slowly in space', *J. Austral. Math. Soc. B* **29**, 480–500.

Roberts, A. J. (1989*a*), 'Appropriate initial conditions for asymptotic descriptions of the long term evolution of dynamical systems', *J. Austral. Math. Soc. B* **31**, 48–75.

Roberts, A. J. (1989*b*), 'The utility of an invariant manifold description of the evolution of a dynamical system', *SIAM J. Math. Anal.* **20**, 1447–1458.

Roberts, A. J. (1997), 'Low-dimensional modelling of dynamics via computer algebra', *Computer Phys. Comm.* **100**, 215–230.

Roberts, A. J. (2000), 'Computer algebra derives correct initial conditions for low-dimensional dynamical models', *Computer Phys. Comm.* **126**(3), 187–206.

Roberts, A. J. (2015*a*), 'Macroscale, slowly varying, models emerge from the microscale dynamics in long thin domains', *IMA Journal of Applied Mathematics* **80**(5), 1492–1518.

Roberts, A. J. (2015*b*), *Model emergent dynamics in complex systems*, SIAM, Philadelphia.
`https://epubs.siam.org/doi/10.1137/1.9781611973563`

Roberts, A. J. (2022), Backwards theory supports modelling via invariant manifolds for non-autonomous dynamical systems, Technical report, [`http://arxiv.org/abs/1804.06998`].