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

A. J. Roberts*

Nov 2013 – February 25, 2026

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.

Contents

*School of Mathematical Sciences, Adelaide University, 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 $\mathcal{O}(\varepsilon^5)$.

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 $\mathcal{O}(\varepsilon^5)$ is here the same as error $\mathcal{O}(|\vec{s}|^6)$.

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

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

On this slow manifold the evolution is

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

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

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

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, with override of some name translations; 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). Let's make the algebra case sensitive: for *psl* we set `raise`; whereas for *csl* we set `lower`. Remember (currently, 2025) `off latex` automatically turns `raise` back on (but not `lower`).

```
12 off raise;  off lower;
```

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
```

```
    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 26 Feb 2026)"$
%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;
zz := tp adjvecs;
```

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

```
maxiter:=29$
```

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

```
trace:=0$
%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. Although seems OK for all other examples. Get case sensitive analysis by also `off raise,lower;`

```
on div,revpri; off allfac;
on rationalize;
```

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

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

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

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

Empty the output LaTeX file in case of error.

```
out "invarManReport.tex";
```

```
write "This empty document indicates error.";
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.

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

3 Dissect the linear part

Section contents

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

```
73 clear exp; operator exp;
74 let { df(exp(~u),t) => df(u,t)*exp(u)
75      , exp(~u)*exp(~v) => exp(u+v)
76      , exp(~u)^~p => exp(p*u)
77      };
```

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.

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

```
82                , sin(~u) => (exp(i*u)-exp(-i*u))/(2*i)
83                } );
```

Need function `conj_` to do parsimonious complex conjugation. Consequently, this procedure implicitly assumes that all parameter and variable names are real, except $\imath := \sqrt{-1}$.

```
84 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.

```
85 clear eval_;  array eval_(m);
86 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??

```
87 matrix aa_(m,m),dexp_(m,m),cexp_(m,m);
88 for j:=1:m do dexp_(j,j):=exp(eval_(j)*t);
89 for j:=1:m do cexp_(j,j):=exp(-conj_(eval_(j))*t);
90 aa_:=(tp map(conj_(~b),ee*dexp_)*zz*cexp_ )$
```

```
91 if trace then write aa_:=aa_;
92 write "Normalising the left-eigenvectors:";
93 aa_:=(aa_ where {exp(0)=>1, exp(~a)=>0 when a neq 0})$
94 if trace then write aa_:=aa_;
95 if det(aa_)=0 then << write
96     "ORTHOGONALITY ERROR IN EIGENVECTORS; I EXIT";
97     return>>;
98 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!?? In cases where there are slow-space variations we need to commute some derivatives one way, and commute other derivatives the other way—all a bit tricksy and unclear enough factors coded here. Also unclear is that `df()` appears to order its arguments in alphabetical order (A–Z,a–z): so 'space' variables before t-for-time may be in the wrong position. Maybe use `pdf()` for space derivative working throughout, instead of `df`??

```
99 clear d_; operator d_; linear d_;
100 let { d_(~a^~p,t,~dt)=>d_(a,t,dt)^p
101     , d_(~a*~b,t,~dt)=>d_(a,t,dt)*d_(b,t,dt)
102     , d_(exp(~a),t,~dt)=>exp(a)
103         *sub(t=-dt,cos(-i*a)+i*sin(-i*a))
104     , df(d_(~a,t,~dt),~b) => d_(df(a,b),t,dt) when not(b member s
105     , d_(df(~a,~b),t,~dt) => df(d_(a,t,dt),b) when (b member spac
106     , d_(df(~a,~b,~c),t,~dt) => df(d_(a,t,dt),b,c) when (b member
107     , d_(df(~a,~b,~c,~d),t,~dt) => df(d_(a,t,dt),b,c,d) when (b n
108     , d_(~a,t,0)=>a
109     , d_(d_(~a,t,~dta),t,~dtb)=>d_(a,t,dta+dtb)
110     };
```

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

```
111 if trace then write "setting somerules";
112 somerules:={}$
113 somerules:=(u1(~dt)=d_(u1,t,dt)).somerules$
114 somerules:=(u2(~dt)=d_(u2,t,dt)).somerules$
115 somerules:=(u3(~dt)=d_(u3,t,dt)).somerules$
116 somerules:=(u4(~dt)=d_(u4,t,dt)).somerules$
117 somerules:=(u5(~dt)=d_(u5,t,dt)).somerules$
118 somerules:=(u6(~dt)=d_(u6,t,dt)).somerules$
119 somerules:=(u7(~dt)=d_(u7,t,dt)).somerules$
120 somerules:=(u8(~dt)=d_(u8,t,dt)).somerules$
121 somerules:=(u9(~dt)=d_(u9,t,dt)).somerules$
122 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.)

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

We need a vector of unknowns for a little while.

```
132 uvec:=0*ff; %nx1 zero matrix
133 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$
```

```
159     for j:=1:n do for k:=1:m do
160         ok:=if reslin(j,k)=0 then ok else 0$
161     if ok then iter:=iter+1000;
162 end;
163 if not ok then << write
164     "OOPS, INCONSISTENT EIGENVALUES, EIGENVECTORS AND OPERATOR.
165     EMAIL ME; I EXIT";
166     return >>;
```

3.5 Ameliorate the nonlinearity

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

```
167 somerules:=for j:=1:n collect
168   (d_(1,t,~dt)*mkid(u,j)=d_(mkid(u,j),t,dt))$
169 ll0_:=(ll*uvec where somerules);
170 ffnonpdf:=(ff where pdfzero);
171 procedure scalepdf(z_);
172     (z_ where {pdf(~a,~b)=>small*df(a,b)
173         , pdf(~a,~b,~n)=>small^n*df(a,b,n) when fixp(n)
174         , pdf(~a,~b,~c)=>small^2*df(a,b,c) when not fixp(c)
175         });
176 ffpdf:=map(scalepdf(~z),ff-ffnonpdf);
177 ff:=small*ffnonpdf
178     +(1-small)*ll0_
179     +ffpdf;
```

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

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

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

```
181 rhsfn:=for i:=1:n sum e_(i,1)*ff(i,1)$
182 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.

```
rhsjact:=for i:=1:n sum for j:=1:n sum
    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.

```
clear eval_s,modes_;
array eval_s(m),modes_(m);
neval:=0$ eval_set:={}$
for j:=1:m do if not(eval_(j) member eval_set) then begin
  neval:=neval+1;
  eval_s(neval):=eval_(j);
  eval_set:=eval_(j).eval_set;
  modes_(neval):=for k:=j:m join
    if eval_(j)=eval_(k) then {k} else {};
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.

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

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

```
eval_s(0):=geneval$
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`.

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

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

```
206   for ii:=1:n do for jj:=1:n do llzz_(ii,jj):=(
207       -sub(small=0,ll(ii,jj)) where d_(1,t,~dt)
208       => 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.

```
209   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.

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

```
217                 });
```

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

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

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

```
222   for i:=1:length(modes_(k_)) do
223     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.

```
224   for i:=length(modes_(k_))+1:m do begin
225     for j:=1:n+i-1 do llzz_(n+i,j):=llzz_(j,n+i):=0;
226     llzz_(n+i,n+i):=1;
227   end;
228   if trace then write
229   "finished Add some trivial rows and columns";
```

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

```
230   if trace then write "llzz_= ",llzz_;
231   llzz_:=llzz_^(-1);
232   if trace then write "llzz_= ",llzz_;
233   l_invs(k_):=for i:=1:n sum for j:=1:n sum e_(i,j)*llzz_(i,j);
234   g_invs(k_):=for i:=1:length(modes_(k_)) sum
235     for j:=1:n sum e_(part(modes_(k_),i),j)*llzz_(i+n,j);
236   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.

```
237   l1_invs(k_) := for ii:=1:n sum for j:=1:n sum
238       e_(ii,j)*conj_(llzz_(j,ii));
239   if trace then write "l1_invs",k_," =",l1_invs(k_);
240   l2_invs(k_) := for ii:=1:n sum
241       for j:=1:length(modes_(k_)) sum
242           e_(ii,part(modes_(k_),j))*conj_(llzz_(j+n,ii));
243   if trace then write "l2_invs",k_," =",l2_invs(k_);
244   g1_invs(k_) := for ii:=1:length(modes_(k_)) sum
245       for j:=1:n sum
246           e_(part(modes_(k_),ii),j)*conj_(llzz_(j,ii+n));
247   if trace then write "g1_invs",k_," =",g1_invs(k_);
248   g2_invs(k_) := for ii:=1:length(modes_(k_)) sum
249       for j:=1:length(modes_(k_)) sum
250           e_(part(modes_(k_),ii),part(modes_(k_),j))
251           *conj_(llzz_(j+n,ii+n));
252   if trace then write "g2_invs",k_," =",g2_invs(k_);
253   if trace then write
254   "finished Unpack the conjugate transpose";
255 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.

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

```
262     tmp_:=l_invs(k_)*exp(a_*t) >>
263   else tmp_:=sub(geneval=a_,l_invs(0))*exp(a_*t)$
264   return tmp_;
265   end;
```

Second for the evolution on the invariant manifold.

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

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

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

Then define operators that we use to update the projection.

```
284 clear l1_inv; operator l1_inv; linear l1_inv;
285 clear l2_inv; operator l2_inv; linear l2_inv;
286 clear g1_inv; operator g1_inv; linear g1_inv;
287 clear g2_inv; operator g2_inv; linear g2_inv;
288 let { l1_inv(e_(~j,~k)*exp(~a_),exp)
289       => inv_proc(a_/t,l1_invs)*e_(j,k)
```

```
290      , l2_inv(e_(~j,~k)*exp(~a_),exp)
291        => inv_proc(a_/t,l2_invs)*e_(j,k)
292      , g1_inv(e_(~j,~k)*exp(~a_),exp)
293        => inv_proc(a_/t,g1_invs)*e_(j,k)
294      , g2_inv(e_(~j,~k)*exp(~a_),exp)
295        => inv_proc(a_/t,g2_invs)*e_(j,k)
296      };
```

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.

```
297 operator diff;
298 if spacex neq {} then
299 let { df(~a,~x,~b)=>diff(x) when a=b
300      , df(~a,~x,~n,~b)=>diff(x)^n when a=b and fixp(n)
301      , df(~a,~x,~y,~b)=>diff(x)*diff(y)
302          when a=b and not fixp(y)
303      , df(~a,~x,~n,~y,~b)=>diff(x)^n*diff(y)
304          when a=b and fixp(n) and not fixp(y)
305      , df(~a,~x,~y,~n,~b)=>diff(x)*diff(y)^n
306          when a=b and not fixp(y) and fixp(n)
307      , df(~a,~x,~y,~z,~b)=>diff(x)*diff(y)*diff(z)
308          when a=b and not fixp(y) and not fixp(z)
309      };
```

4 Initialise LaTeX output

Define the Greek alphabet with `small` as well.

```
310 defid small,name="\eps";%varepsilon;
311 defid alpha,name=alpha;
312 defid beta,name=beta;
```

```
313 defid gamma,name=gamma;
314 defid delta,name=delta;
315 defid epsilon,name=epsilon;
316 defid varepsilon,name=varepsilon;
317 defid zeta,name=zeta;
318 defid eta,name=eta;
319 defid theta,name=theta;
320 defid vartheta,name=vartheta;
321 defid iota,name=iota;
322 defid kappa,name=kappa;
323 defid lambda,name=lambda;
324 defid mu,name=mu;
325 defid nu,name=nu;
326 defid xi,name=xi;
327 defid pi,name=pi;
328 defid varpi,name=varpi;
329 defid rho,name=rho;
330 defid varrho,name=varrho;
331 defid sigma,name=sigma;
332 defid varsigma,name=varsigma;
333 defid tau,name=tau;
334 defid upsilon,name=upsilon;
335 defid phi,name=phi;
336 defid varphi,name=varphi;
337 defid chi,name=chi;
338 defid psi,name=psi;
339 defid omega,name=omega;
340 defid Gamma,name=Gamma;
341 defid Delta,name=Delta;
342 defid Theta,name=Theta;
343 defid Lambda,name=Lambda;
344 defid Xi,name=Xi;
345 defid Pi,name=Pi;
346 defid Sigma,name=Sigma;
347 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"; %AJR, 2026-01-16: delete?
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"$
```

```
403 shut "invarManReportSys.tex"$
404 off latex$  off raise,lower$
```

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.

```
405 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.

```
406 clear gg;
407 clear s; operator s; depend s,t;
408 foreach z_ in spacex do depend s,z_;
409 let { df(s(~j),t)=>coeffn(gg,e_(j,1),1)
410 , df(s(~j),t,~b)=> df(coeffn(gg,e_(j,1),1),b) when not fixp(b)
411 , df(s(~j),t,~b)=> df(coeffn(gg,e_(j,1),1),t,b-1) when (fixp(b) a
412 , df(s(~j),t,~b,~c)=> df(coeffn(gg,e_(j,1),1),b,c) when not fixp(
413 , df(s(~j),t,~b,~c)=> df(coeffn(gg,e_(j,1),1),t,b-1,c) when (fixp
414 , df(s(~j),t,~b,~c,~d)=> df(coeffn(gg,e_(j,1),1),b,c,d) when not
415 , df(s(~j),t,~b,~c,~d)=> df(coeffn(gg,e_(j,1),1),t,b-1,c,d) when
416 };
```

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

```
417 procedure manifold_(uu,n);
418     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.

```
419 uu:=for j:=1:m sum s(j)*exp(eval_(j)*t)
420   *(for k:=1:n sum e_(k,1)*ee(k,j));
421 gg:=0;
422 if trace then write uu:=uu;
```

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

```
423 procedure matify_(a,m,n)$
424   begin matrix z_(m,n);
425     for i:=1:m do for j:=1:n do z_(i,j):=coeffn(a,e_(i,j),1);
426     return (z_ where {exp(0)=>1});
427     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.

```
428 zs:=for j:=1:m sum exp(eval_(j)*t)
429   *(for k:=1:n sum e_(k,j)*zz(k,j))$
430 pp:=0$
```

6 Iteratively construct the invariant manifold

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

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

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

```
433 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_
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).

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

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

```
454 uu:=uu+l_inv(resde,exp);
455 if trace then write "uu=",matify_(uu,n,1);
456 gg:=gg+g_inv(resde,exp);
457 if trace then write "gg=",matify_(gg,m,1);
```

Now update the isochron projection, with normalisation.

```
458 if itisAutoSlowMan_ then begin
459 if trace then write "starting zs evaluation";
460 zs:=zs+l1_inv(resd,exp)-l2_inv(resz,exp);
461 if trace then write "zs=",matify_(zs,n,m);
462 pp:=pp-g1_inv(resd,exp)+youshouldnotseethis*g2_inv(resz,exp);
463 if trace then write "pp=",matify_(pp,m,m);
464 end;%if itisAutoSlowMan_
```

Terminate the iteration loop once residuals are zero.

```
465 if {resde,resd,resz}={0,0,0} then write iter:=iter+10000;
466 end;
467 showtime;
468 if not itisAutoSlowMan_ then clear zs;
```

Only proceed to print if terminated successfully.

```
469 if {resde,resd,resz}={0,0,0}
470   then write "SUCCESS: converged to an expansion"
471   else <<write "FAILED TO CONVERGE; I EXIT";
472     return; >>;
```

7 Output text report of results

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

```
473 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.

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

Optionally write the projection vectors.

```
if itisAutoSlowMan_ then begin write "
The normals to the 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.";
  for j:=1:m do write "z",j," = ",
    for i:=1:n collect coeffn(zs,e_(i,j),1);
end;
```

Write text results numerically evaluated when expressions are long.

```
if length(gg)>30 then begin
on rounded; print_precision 4$
write "Numerically, the invariant manifold is (to one order lowe
for j:=1:n do write "u",j," = ",
  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;
```

```
505 end;
```

8 Output LaTeX report of results

Change the printing of temporary arrays.

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

Gather complicated result

```
513 for k:=1:m do for j:=1:n do
514     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.

```
515 clear order_;  operator order_;
516 defid order_,name="O";
517 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.

```
518 out "scratchfile.red";
519 write "off echo;"$ % do not understand why needed in 2021??
520 write "write ""\)
521 \paragraph{The invariant manifold}
522 These give the location of the invariant manifold in
523 terms of parameters~\(s_ j\).
524 \("";";
525 for j:=1:n do write "tmp_(",j,"):=coeffn(small*uu,e_(",j,
```

```
526         ",1),1)/small +order_(varepsilon^",toosmall-1,");";
527 if length(gg)>30 then begin
528 write "on rounded; print_precision 4$"$
529 for j:=1:n do write "tmp_(",j,"):=coeffn(small*uu,e_(",j,
530         ",1),1)/small +order_(varepsilon^",toosmall-1,");";
531 write "off rounded;"$
532 end;
```

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

```
533 write "write ""\)
534 \paragraph{Invariant manifold ODEs}
535 The system evolves on the invariant manifold such
536 that the parameters evolve according to these ODEs.
537 \(""";";
538 for j:=1:m do write "tmp_s(",j,"):=1*coeffn(gg,e_(",j
539             ",1),1)+order_(varepsilon^",toosmall,");";
540 if length(gg)>30 then begin
541 write "on rounded; print_precision 4$"$
542 for j:=1:m do write "tmp_s(",j,"):=1*coeffn(gg,e_(",j
543             ",1),1)+order_(varepsilon^",toosmall,");";
544 write "off rounded;"$
545 end;
```

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

```
546 if itisAutoSlowMan_ then begin
547   write "write ""\)
548 \paragraph{Normals to isochrons at the slow manifold}
549 Use these vectors: to project initial conditions
550 onto the slow manifold; to project non-autonomous
551 forcing onto the slow evolution; to predict the
552 consequences of modifying the original system; in
553 uncertainty quantification to quantify effects on
554 the model of uncertainties in the original system.
555 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,lower$
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.

```
deflist('((!( !\!b!i!g!() (!) !\!b!i!g!))
         (!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.

```
572 symbolic procedure prinlaend;
573 <<terpri();
574   prin2t "\)\par";
575   if !*verbatim then
576       <<prin2t "\begin{verbatim}";
577         prin2t "REDUCE Input:">>;
578   ncharspr!*:=0;
579   if ofl!* then linelength(car linel!*)
580     else laline!*:=cdr linel!*;
581   nochar!*:=append(nochar!*,nochar1!*);
582   nochar1!*:=nil >>$
583   %
```

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

```
584 symbolic procedure prinlabegin;
585 <<if !*verbatim then
586       <<terpri();
587         prin2t "\end{verbatim}">>;
588   linel!*:=linelength nil . laline!*;
589   if ofl!* then linelength(laline!* + 2)
590     else laline!*:=car linel!* - 2;
591   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.

```
592 symbolic procedure latexon;
593 <<!*!*a2sfn:='texaeval;
594   !*raise:=nil;
```

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

End the file when read by Reduce

```
614 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].