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

A. J. Roberts*

Nov 2013 – October 15, 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.

Contents

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

1 Introduction

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`$/dt$, ...;

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

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 `diff()` operator (currently with at most three arguments)—examples are `diff(u1,x)` $= \partial u_1/\partial x$, `diff(u3,x,y)` $= \partial^2 u_3/\partial x\partial y$, `diff(u2,x,4)` $= \partial^4 u_2/\partial x^4$.

- `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 $\mathcal{O}\big(\varepsilon^{\texttt{toosmall}}\big)$ (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 version 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!),

$$u_1 = 3/8\varepsilon^3 s_1^4 - 1/2\varepsilon s_1^2 + s_1 + O(\varepsilon^4),$$
$$u_2 = -3/8\varepsilon^3 s_1^4 + 1/2\varepsilon s_1^2 + s_1 + O(\varepsilon^4).$$

On this slow manifold the evolution is

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

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

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

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

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 `diff` operator that has the same syntax as `df()` but we need separate name in order to delay Reduce processing.

```
12 operator diff;
```

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`.

```
13 on ezgcd;
```

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

```
operator invariantmanifold;
```

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

```
IMerrorMsg:="invariantmanifold() Should Have Six Arguments"$
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, diffzero;
```

Write an intro message.

```
write "Construct invariant manifold (version 15 Oct 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;
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. But seems OK for all other examples.

```
on div; off allfac; on revpri;
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 `diff` terms, currently coded up to five arguments.

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

3 Dissect the linear part

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??

```
88 matrix aa_(m,m),dexp_(m,m),cexp_(m,m);
89 for j:=1:m do dexp_(j,j):=exp(eval_(j)*t);
90 for j:=1:m do cexp_(j,j):=exp(-conj_(eval_(j))*t);
91 aa_:=(tp map(conj_(~b),ee*dexp_)*zz*cexp_ )$
92 if trace then write aa_:=aa_;
93 write "Normalising the left-eigenvectors:";
94 aa_:=(aa_ where {exp(0)=>1, exp(~a)=>0 when a neq 0})$
95 if trace then write aa_:=aa_;
96 if det(aa_)=0 then << write
97     "ORTHOGONALITY ERROR IN EIGENVECTORS; I EXIT";
98     return>>;
99 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!?

```
100 clear d_; operator d_; linear d_;
101 let { d_(~a^~p,t,~dt)=>d_(a,t,dt)^p
102     , d_(~a*~b,t,~dt)=>d_(a,t,dt)*d_(b,t,dt)
103     , d_(exp(~a),t,~dt)=>exp(a)
104         *sub(t=-dt,cos(-i*a)+i*sin(-i*a))
105     , df(d_(~a,t,~dt),~b)=>d_(df(a,b),t,dt)
106     , d_(~a,t,0)=>a
107     , d_(d_(~a,t,~dta),t,~dtb)=>d_(a,t,dta+dtb)
108     };
```

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

```
109 if trace then write "setting somerules";
110 somerules:={}$
111 somerules:=(u1(~dt)=d_(u1,t,dt)).somerules$
112 somerules:=(u2(~dt)=d_(u2,t,dt)).somerules$
113 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 diffzero );
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 $\mathcal{L}_{\rm new} := \mathcal{L} - \mathcal{L}_{\rm adj}$ 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);
ffnondiff:=(ff where diffzero);
procedure scalediff(z_);
    (z_ where {diff(~a,~b)=>small*df(a,b)
        , diff(~a,~b,~n)=>small^n*df(a,b,n) when fixp(n)
        , diff(~a,~b,~c)=>small^2*df(a,b,c) when not fixp(c)
        });
ffdiff:=map(scalediff(~z),ff-ffnondiff);
ff:=small*ffnondiff
    +(1-small)*ll0_
    +ffdiff;
```

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

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

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

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

```
182     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.

```
183 clear eval_s,modes_;
184 array eval_s(m),modes_(m);
185 neval:=0$ eval_set:={}$
186 for j:=1:m do if not(eval_(j) member eval_set) then begin
187   neval:=neval+1;
188   eval_s(neval):=eval_(j);
189   eval_set:=eval_(j).eval_set;
190   modes_(neval):=for k:=j:m join
191     if eval_(j)=eval_(k) then {k} else {};
192 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.

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

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

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

3.7 Precompute matrices for updates

Precompute matrices to solve for updates for each of the critical eigenvalues, and the general case $\mathtt{k} = 0$. The matrix

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

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

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

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

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

```
207   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.

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

```
215                 });
```

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

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

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

```
220   for i:=1:length(modes_(k_)) do
221     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.

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

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

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

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

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

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

Second for the evolution on the invariant manifold.

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

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

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

Then define operators that we use to update the projection.

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

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

3.9 Projection onto slow manifold for PDE case

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

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

4 Initialise LaTeX output

Define the Greek alphabet with `small` as well.

```
306 defid small,name="\eps";%varepsilon;
307 defid alpha,name=alpha;
308 defid beta,name=beta;
309 defid gamma,name=gamma;
310 defid delta,name=delta;
```

```
311 defid epsilon,name=epsilon;
312 defid varepsilon,name=varepsilon;
313 defid zeta,name=zeta;
314 defid eta,name=eta;
315 defid theta,name=theta;
316 defid vartheta,name=vartheta;
317 defid iota,name=iota;
318 defid kappa,name=kappa;
319 defid lambda,name=lambda;
320 defid mu,name=mu;
321 defid nu,name=nu;
322 defid xi,name=xi;
323 defid pi,name=pi;
324 defid varpi,name=varpi;
325 defid rho,name=rho;
326 defid varrho,name=varrho;
327 defid sigma,name=sigma;
328 defid varsigma,name=varsigma;
329 defid tau,name=tau;
330 defid upsilon,name=upsilon;
331 defid phi,name=phi;
332 defid varphi,name=varphi;
333 defid chi,name=chi;
334 defid psi,name=psi;
335 defid omega,name=omega;
336 defid Gamma,name=Gamma;
337 defid Delta,name=Delta;
338 defid Theta,name=Theta;
339 defid Lambda,name=Lambda;
340 defid Xi,name=Xi;
341 defid Pi,name=Pi;
342 defid Sigma,name=Sigma;
343 defid Upsilon,name=Upsilon;
344 defid Phi,name=Phi;
345 defid Psi,name=Psi;
```

```
346 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.

```
347 defindex diff(down);
348 defid diff,name="\diff";
349 defindex e_(down,down);
350 defid e_,name="e";
351 defindex d_(arg,down,down);
352 defid d_,name="D";
353 defindex u(down);
354 defid u1,name="u_1";
355 defid u2,name="u_2";
356 defid u3,name="u_3";
357 defid u4,name="u_4";
358 defid u5,name="u_5";
359 defid u6,name="u_6";
360 defid u7,name="u_7";
361 defid u8,name="u_8";
362 defid u9,name="u_9";
363 defindex s(down);
364 defid exp,name="\exp";
365 defindex exp(arg);
```

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

```
366 clear tmp_,tmp_s,tmp_z;
367 array tmp_(n),tmp_s(m),tmp_z(m);
368 defindex tmp_(down);
369 defindex tmp_s(down);
370 defindex tmp_z(down);
371 defid tmp_,name="\dot u";
372 defid tmp_s,name="\vec e";
373 defid tmp_z,name="\vec z";
374 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.

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

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.

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

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

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

```
procedure matify_(a,m,n)$
  begin matrix z_(m,n);
    for i:=1:m do for j:=1:n do z_(i,j):=coeffn(a,e_(i,j),1);
```

```
414     return (z_ where {exp(0)=>1,small=>s});
415     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.

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

6 Iteratively construct the invariant manifold

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

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

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

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

Iteratively construct the invariant manifold.

```
422 write "Start iterative construction of invariant manifold";
423 for iter:=1:maxiter do begin
424 if trace then write "
425 ITERATION = ",iter,"
426 -------------";
```

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

```
427 resde:=-df(uu,t)+sub(manifold_(uu,n),rhsfn);
428 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.

```
429 est:=tpe_(for j:=1:m sum df(uu,s(j))*e_(1,j),e_);
```

```
430 est:=conj_(est);
431 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.

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

Compute residual of the normalisation of the projection.

```
437     resz:=est*zs-eyem*exp(0);
438     if trace then write "resz=",matify_(resz,m,m);
439 end %if itisAutoSlowMan_
440 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).

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

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

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

Now update the isochron projection, with normalisation.

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

Terminate the iteration loop once residuals are zero.

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

Only proceed to print if terminated successfully.

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

7 Output text version of results

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

```
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;
end;
```

8 Output LaTeX version 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
```

```
525 that the parameters evolve according to these ODEs.
526 \("";";
527 for j:=1:m do write "tmp_s(",j,"):=1*coeffn(gg,e_(",j
528            ,",1),1)+order_(varepsilon^",toosmall,");";
529 if length(gg)>30 then begin
530 write "on rounded; print_precision 4$"$
531 for j:=1:m do write "tmp_s(",j,"):=1*coeffn(gg,e_(",j
532            ,",1),1)+order_(varepsilon^",toosmall,");";
533 write "off rounded;"$
534 end;
```

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

```
535 if itisAutoSlowMan_ then begin
536   write "write ""\)
537 \paragraph{Normals to isochrons at the slow manifold}
538 Use these vectors: to project initial conditions
539 onto the slow manifold; to project non-autonomous
540 forcing onto the slow evolution; to predict the
541 consequences of modifying the original system; in
542 uncertainty quantification to quantify effects on
543 the model of uncertainties in the original system.
544 The normal vector \(\vec z_j:=(z_{j1},\ldots,z_{jn})\)
545 \("";";
546   for i:=1:m do for j:=1:n do
547   write "zs_(",i,",",j,"):=tmp_zz(",i,",",j
548         ,")+order_(varepsilon^",toosmall,");";
549 end;%if itisAutoSlowMan_
```

Finish the scratchfile.

```
550 write ";end;";
551 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!

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

```
%load_package rlfi; %must be loaded early
deflist('((!( !\!b!i!g!() (!) !\!b!i!g!)) (!P!I !\!p!i! )
          (!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.

```
symbolic procedure prinlaend;
<<terpri();
  prin2t "\)\par";
  if !*verbatim then
      <<prin2t "\begin{verbatim}";
        prin2t "REDUCE Input:">>;
```

```
568   ncharspr!*:=0;
569   if ofl!* then linelength(car linel!*)
570     else laline!*:=cdr linel!*;
571   nochar!*:=append(nochar!*,nochar1!*);
572   nochar1!*:=nil >>$
573   %
```

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

```
574 symbolic procedure prinlabegin;
575 <<if !*verbatim then
576       <<terpri();
577         prin2t "\end{verbatim}">>;
578   linel!*:=linelength nil . laline!*;
579   if ofl!* then linelength(laline!* + 2)
580     else laline!*:=car linel!* - 2;
581   prin2 "\(" >>$
```

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

```
582 symbolic procedure latexon;
583 <<!*!*a2sfn:='texaeval;
584   !*raise:=nil;
585   prin2t "\documentclass[11pt,a5paper]{article}";
586   prin2t "\usepackage[a5paper,margin=13mm]{geometry}";
587   prin2t "\usepackage{parskip,time,verbatim} \def\_{_}";
588   prin2t "\raggedright \def\eps{\varepsilon}";
589   prin2t "\def\diff{\partial\makebox[0pt]{\ $\cdot$}}";
590   prin2t "\title{Invariant manifold of your dynamical system}";
591   prin2t "\author{A. J. Roberts, University of Adelaide\\";
592   prin2t "\texttt{http://orcid.org/0000-0001-8930-1552}}";
593   prin2t "\date{\now, \today}";
594   prin2t "\begin{document}";
595   prin2t "\maketitle";
```

```
596   prin2t "Throughout and generally: the lowest order, most";
597   prin2t "important, terms are near the end of each expression.";
598   prin2t "\par\leftskip=2em  \parindent=-2em";
599   prin2t "\input{invarManReportSys}";
600   if !*verbatim then
601       <<prin2t "\begin{verbatim}";
602         prin2t "REDUCE Input:">>;
603   put('tex,'rtypefn,'(lambda(x) 'tex)) >>$
```

End the file when read by Reduce

```
604 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].