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

A. J. Roberts*

Nov 2013 – November 25, 2022

Abstract

This procedure constructs a specified invariant manifold for a specified system of ordinary differential equations or delay differential equations. 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

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

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

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(odefns, evals, evecs, adjvecs, toosmall);
```

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

- `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;

- `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), 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), and further extended to the projection of initial conditions, forcing, uncertainty via the innovations of 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, Example 2.1)

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

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

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

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.

1.2 Header of the procedure

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.

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

Usually seems best to use the `gcd` option, but sometimes worse, so set here in order for a user to optionally turn off before invoking the procedure.

```
on gcd,ezgcd;
```

Now define the procedure as an operator so we can define procedures internally, and may be flexible with its arguments. Section 8 marks the end of the procedure.

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

1.3 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;
```

Write an intro message.

```
write "Construct an invariant manifold (version 25 Nov 2022)"$
```

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";
```

1.4 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 no. of variables ",
n:=part(length(ee),1);
write "no. 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 Dissect the linear part

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.

```
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.

```
59 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.

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

2.1 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??

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

```
72      return>>;
73 zz:=zz*aa_^(-1);
```

2.2 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!?

```
74 clear d_; operator d_; linear d_;
75 let { d_(~a^~p,t,~dt)=>d_(a,t,dt)^p
76      , d_(~a*~b,t,~dt)=>d_(a,t,dt)*d_(b,t,dt)
77      , d_(exp(~a),t,~dt)=>exp(a)
78          *sub(t=-dt,cos(-i*a)+i*sin(-i*a))
79      , df(d_(~a,t,~dt),~b)=>d_(df(a,b),t,dt)
80      , d_(~a,t,0)=>a
81      , d_(d_(~a,t,~dta),t,~dtb)=>d_(a,t,dta+dtb)
82      };
```

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

```
83 if trace then write "setting somerules";
84 somerules:={}$
85 depend u1,t; somerules:=(u1(~dt)=d_(u1,t,dt)).somerules$
86 depend u2,t; somerules:=(u2(~dt)=d_(u2,t,dt)).somerules$
87 depend u3,t; somerules:=(u3(~dt)=d_(u3,t,dt)).somerules$
88 depend u4,t; somerules:=(u4(~dt)=d_(u4,t,dt)).somerules$
89 depend u5,t; somerules:=(u5(~dt)=d_(u5,t,dt)).somerules$
90 depend u6,t; somerules:=(u6(~dt)=d_(u6,t,dt)).somerules$
91 depend u7,t; somerules:=(u7(~dt)=d_(u7,t,dt)).somerules$
92 depend u8,t; somerules:=(u8(~dt)=d_(u8,t,dt)).somerules$
93 depend u9,t; somerules:=(u9(~dt)=d_(u9,t,dt)).somerules$
94 ff:=(ff where somerules)$
```

2.3 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));
end;
write "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);
```

2.4 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))*e
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";
write
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 >>;
```

2.5 Ameliorate the nonlinearity

Anything not in the linear operator gets multiplied by `small` to be treated as small in the analysis. The feature of the second alternative is that when a user invokes `small` then the power of smallness is not then changed; however, causes issues in the relative scaling of some terms, so restore to the original version. This might need reconsidering. The current `if` always chooses the first simple alternative.

```
139 somerules:=for j:=1:n collect
140   (d_(1,t,~dt)*mkid(u,j)=d_(mkid(u,j),t,dt))$
141 ll0_:=(ll*uvec where somerules)$
142 ff:=(if 1 then small*ff
143             else ff-(1-small)*sub(small=0,ff))
144     +(1-small)*ll0_$
```

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

```
145 %ff0:=(ff where uzero)$ % obliterates u1,... as operators
146 ff:=ff+(exp(0)-1)*sub(uzero,ff)$
```

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

```
147 rhsfn:=for i:=1:n sum e_(i,1)*ff(i,1)$
148 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.

```
149 rhsjact:=for i:=1:n sum for j:=1:n sum
150     e_(j,i)*df(ff(i,1),mkid(u,j))$
```

2.6 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.

```
151 clear eval_s,modes_;
152 array eval_s(m),modes_(m);
153 neval:=0$ eval_set:={}$
154 for j:=1:m do if not(eval_(j) member eval_set) then begin
155   neval:=neval+1;
156   eval_s(neval):=eval_(j);
157   eval_set:=eval_(j).eval_set;
158   modes_(neval):=for k:=j:m join
159     if eval_(j)=eval_(k) then {k} else {};
160 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.

```
161 itisSlowMan_:=if eval_set={0} then 1 else 0$
162 if trace then write itisSlowMan_:=itisSlowMan_;
```

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

```
163 eval_s(0):=geneval$
164 modes_(0):={}$
```

2.7 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`.

```
165 matrix llzz_(n+m,n+m);
```

```
clear l_invs,g_invs,l1_invs,g1_invs,l2_invs,g2_invs;
array l_invs(neval), g_invs(neval), l1_invs(neval),
     g1_invs(neval), l2_invs(neval), g2_invs(neval);
clear sp_; operator sp_; linear sp_;
for k_:=0:neval do begin
  if trace then write "ITERATION ",k_;
```

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

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

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

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

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

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

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

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

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

```
188   for i:=1:length(modes_(k_)) do
189     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.

```
190   for i:=length(modes_(k_))+1:m do begin
191     for j:=1:n+i-1 do llzz_(n+i,j):=llzz_(j,n+i):=0;
192     llzz_(n+i,n+i):=1;
193   end;
194   if trace then write
195   "finished Add some trivial rows and columns";
```

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

```
196   if trace then write llzz_:=llzz_;
197   llzz_:=llzz_^(-1);
198   if trace then write llzz_:=llzz_;
199   l_invs(k_):=for i:=1:n sum for j:=1:n sum e_(i,j)*llzz_(i,j);
200   g_invs(k_):=for i:=1:length(modes_(k_)) sum
201     for j:=1:n sum e_(part(modes_(k_),i),j)*llzz_(i+n,j);
202   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.

```
203   l1_invs(k_) := for ii:=1:n sum for j:=1:n sum
204       e_(ii,j)*conj_(llzz_(j,ii));
205   l2_invs(k_) := for ii:=1:n sum
206       for j:=1:length(modes_(k_)) sum
207           e_(ii,part(modes_(k_),j))*conj_(llzz_(j+n,ii));
208   g1_invs(k_) := for ii:=1:length(modes_(k_)) sum
209       for j:=1:n sum
210           e_(part(modes_(k_),ii),j)*conj_(llzz_(j,ii+n));
```

```
211   g2_invs(k_) := for ii:=1:length(modes_(k_)) sum
212       for j:=1:length(modes_(k_)) sum
213           e_(part(modes_(k_),ii),part(modes_(k_),j))
214           *conj_(llzz_(j+n,ii+n));
215   if trace then write
216   "finished Unpack the conjugate transpose";
217 end;
```

2.8 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.

```
218 clear l_inv; operator l_inv; linear l_inv;
219 let l_inv(e_(~j,~k)*exp(~a),exp)=>l_invproc(a/t)*e_(j,k);
220 procedure l_invproc(a);
221   if a member eval_set
222   then << k_:=0;
223     repeat k_:=k_+1 until a=eval_s(k_);
224     l_invs(k_)*exp(a*t) >>
225   else sub(geneval=a,l_invs(0))*exp(a*t)$
```

Second for the evolution on the invariant manifold.

```
226 clear g_inv; operator g_inv; linear g_inv;
227 let g_inv(e_(~j,~k)*exp(~a),exp)=>ginv_proc(a/t)*e_(j,k);
228 procedure ginv_proc(a);
229   if a member eval_set
230   then << k_:=0;
231     repeat k_:=k_+1 until a=eval_s(k_);
232     g_invs(k_) >>
233   else sub(geneval=a,g_invs(0))$
```

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

procedure.

```
234 procedure inv_proc(a,invs);
235   if a member eval_set
236   then << k_:=0;
237     repeat k_:=k_+1 until a=eval_s(k_);
238     invs(k_)*exp(a*t) >>
239   else sub(geneval=a,invs(0))*exp(a*t)$
```

Then define operators that we use to update the projection.

```
240 clear l1_inv; operator l1_inv; linear l1_inv;
241 clear l2_inv; operator l2_inv; linear l2_inv;
242 clear g1_inv; operator g1_inv; linear g1_inv;
243 clear g2_inv; operator g2_inv; linear g2_inv;
244 let { l1_inv(e_(~j,~k)*exp(~a),exp)
245       => inv_proc(a/t,l1_invs)*e_(j,k)
246     , l2_inv(e_(~j,~k)*exp(~a),exp)
247       => inv_proc(a/t,l2_invs)*e_(j,k)
248     , g1_inv(e_(~j,~k)*exp(~a),exp)
249       => inv_proc(a/t,g1_invs)*e_(j,k)
250     , g2_inv(e_(~j,~k)*exp(~a),exp)
251       => inv_proc(a/t,g2_invs)*e_(j,k)
252     };
```

3 Initialise LaTeX output

Define the Greek alphabet with `small` as well.

```
253 defid small,name="\eps";%varepsilon;
254 defid alpha,name=alpha;
255 defid beta,name=beta;
256 defid gamma,name=gamma;
257 defid delta,name=delta;
258 defid epsilon,name=epsilon;
259 defid varepsilon,name=varepsilon;
```

```
260 defid zeta,name=zeta;
261 defid eta,name=eta;
262 defid theta,name=theta;
263 defid vartheta,name=vartheta;
264 defid iota,name=iota;
265 defid kappa,name=kappa;
266 defid lambda,name=lambda;
267 defid mu,name=mu;
268 defid nu,name=nu;
269 defid xi,name=xi;
270 defid pi,name=pi;
271 defid varpi,name=varpi;
272 defid rho,name=rho;
273 defid varrho,name=varrho;
274 defid sigma,name=sigma;
275 defid varsigma,name=varsigma;
276 defid tau,name=tau;
277 defid upsilon,name=upsilon;
278 defid phi,name=phi;
279 defid varphi,name=varphi;
280 defid chi,name=chi;
281 defid psi,name=psi;
282 defid omega,name=omega;
283 defid Gamma,name=Gamma;
284 defid Delta,name=Delta;
285 defid Theta,name=Theta;
286 defid Lambda,name=Lambda;
287 defid Xi,name=Xi;
288 defid Pi,name=Pi;
289 defid Sigma,name=Sigma;
290 defid Upsilon,name=Upsilon;
291 defid Phi,name=Phi;
292 defid Psi,name=Psi;
293 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.

```
294 defindex e_(down,down);
295 defid e_,name="e";
296 defindex d_(arg,down,down);
297 defid d_,name="D";
298 defindex u(down);
299 defid u1,name="u\sb1";
300 defid u2,name="u\sb2";
301 defid u3,name="u\sb3";
302 defid u4,name="u\sb4";
303 defid u5,name="u\sb5";
304 defid u6,name="u\sb6";
305 defid u7,name="u\sb7";
306 defid u8,name="u\sb8";
307 defid u9,name="u\sb9";
308 defindex s(down);
309 defid exp,name="\exp";
310 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.

```
311 clear tmp_,tmp_s,tmp_z;
312 array tmp_(n),tmp_s(m),tmp_z(m);
313 defindex tmp_(down);
314 defindex tmp_s(down);
315 defindex tmp_z(down);
316 defid tmp_,name="\dot u";
317 defid tmp_s,name="\vec e";
318 defid tmp_z,name="\vec z";
319 rhs_:=rhsfn$
320 for k:=1:m do tmp_s(k):=
321     {for j:=1:n collect ee(j,k),exp(eval_(k)*t)};
322 for k:=1:m do tmp_z(k):=
```

```
323     {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.

```
324 out "scratchfile.red";
325 write "off echo;"$  % do not understand why needed in 2021??
326 write "write ""\)
327 \paragraph{The specified dynamical system}
328 \("";";
329 for j:=1:n do write "tmp_(",j,"):=
330     coeffn(rhs_,e_(",j,",1),1);";
331 write "write ""\)
332 \paragraph{Invariant subspace basis vectors}
333 \("";";
334 for j:=1:m do write "tmp_s(",j,"):=tmp_s(",j,");";
335 for j:=1:m do write "tmp_z(",j,"):=tmp_z(",j,");";
336 write "end;";
337 shut "scratchfile.red";
```

Now print the dynamical system to the LaTeX sub-file.

```
338 write "Ignore the following 13 lines of LaTeX"$
339 on latex$
340 out "invarManReportSys.tex"$
341 in "scratchfile.red"$
342 shut "invarManReportSys.tex"$
343 off latex$
```

4 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.

```
344 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.

```
345 clear gg;
346 clear s; operator s; depend s,t;
347 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.

```
348 procedure manifold_(uu,n);
349     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.

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

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

```
354 procedure matify_(a,m,n)$
355   begin matrix z(m,n);
356     for i:=1:m do for j:=1:n do z(i,j):=coeffn(a,e_(i,j),1);
357     return (z where {exp(0)=>1,small=>s});
358     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.

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

5 Iteratively construct the invariant manifold

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

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

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

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

Iteratively construct the invariant manifold.

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

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

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

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

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

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

```
if itisSlowMan_ then begin
```

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

Compute residual of the normalisation of the projection.

```
380     resz:=est*zs-eyem*exp(0);
381     if trace then write "resz=",matify_(resz,m,m);
382 end 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).

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

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

```
384 uu:=uu+l_inv(resde,exp);
385 gg:=gg+g_inv(resde,exp);
386 if trace then write "gg=",matify_(gg,m,1);
387 if trace then write "uu=",matify_(uu,n,1);
```

Now update the isochron projection, with normalisation.

```
388 if itisSlowMan_ then begin
389 zs:=zs+l1_inv(resd,exp)-l2_inv(resz,exp);
390 pp:=pp-g1_inv(resd,exp)+youshouldnotseethis*g2_inv(resz,exp);
391 if trace then write "zs=",matify_(zs,n,m);
392 if trace then write "pp=",matify_(pp,m,m);
393 end;
```

Terminate the iteration loop once residuals are zero.

```
394 showtime;
395 if {resde,resd,resz}={0,0,0} then write iter:=iter+10000;
396 end;
```

Only proceed to print if terminated successfully.

```
397 if {resde,resd,resz}={0,0,0}
```

```
398   then write "SUCCESS: converged to an expansion"
399   else <<write "FAILED TO CONVERGE; I EXIT";
400     return; >>;
```

6 Output text version of results

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

```
401 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.

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

Optionally write the projection vectors.

```
408 if itisSlowMan_ then begin write "
409 The normals to the isochrons at the slow manifold.
410 Use these vectors: to project initial conditions
411 onto the slow manifold; to project non-autonomous
412 forcing onto the slow evolution; to predict the
413 consequences of modifying the original system; in
414 uncertainty quantification to quantify effects on
415 the model of uncertainties in the original system.";
416   for j:=1:m do write "z",j," = ",
417     for i:=1:n collect coeffn(zs,e_(i,j),1);
418 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 itisSlowMan_ 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;
```

7 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~\(s\sb j\).
\("";";
for j:=1:n do write "tmp_(",j,"):=coeffn(small*uu,e_(",j,
        ",1),1)/small +order_(varepsilon^",toosmall-1,");";
if length(gg)>30 then begin
write "on rounded; print_precision 4$"$
for j:=1:n do write "tmp_(",j,"):=coeffn(small*uu,e_(",j,
        ",1),1)/small +order_(varepsilon^",toosmall-1,");";
write "off rounded;"$
end;
```

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

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

```
end;
```

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

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

Finish the scratchfile.

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

Execute the scratchfile with the required commands, with output to the main invariant manifold LaTeX file.

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

8 Fin

That's all folks, so end the procedure.

```
496 return Finished_constructing_invariant_manifold_of_system$
497 end$
```

9 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.

```
498 %load_package rlfi; %must be loaded early
499 deflist('((!( !\!b!i!g!() (!) !\!b!i!g!)) (!P!I !\!p!i! )
500          (!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.

```
501 symbolic procedure prinlaend;
502 <<terpri();
503   prin2t "\)\par";
504   if !*verbatim then
505       <<prin2t "\begin{verbatim}";
506         prin2t "REDUCE Input:">>;
507   ncharspr!*:=0;
508   if ofl!* then linelength(car linel!*)
509     else laline!*:=cdr linel!*;
510   nochar!*:=append(nochar!*,nochar1!*);
511   nochar1!*:=nil >>$
512   %
```

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

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

```
symbolic procedure latexon;
<<!*!*a2sfn:='texaeval;
  !*raise:=nil;
  prin2t "\documentclass[11pt,a5paper]{article}";
  prin2t "\usepackage[a5paper,margin=13mm]{geometry}";
  prin2t "\usepackage{parskip,time} \raggedright";
  prin2t "\def\eps{\varepsilon} \def\_{_}";
  prin2t "\title{Invariant manifold of your dynamical system}";
  prin2t "\author{A. J. Roberts, University of Adelaide\\";
  prin2t "\texttt{http://orcid.org/0000-0001-8930-1552}}";
  prin2t "\date{\now, \today}";
  prin2t "\begin{document}";
  prin2t "\maketitle";
  prin2t "Throughout and generally: the lowest order, most";
  prin2t "important, terms are near the end of each expression.";
  prin2t "\input{invarManReportSys}";
  if !*verbatim then
      <<prin2t "\begin{verbatim}";
        prin2t "REDUCE Input:">>;
  put('tex,'rtypefn,'(lambda(x) 'tex)) >>$
```

End the file when read by Reduce

```
541 end;
```

References

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. (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), *Model emergent dynamics in complex systems*, SIAM, Philadelphia.
`http://bookstore.siam.org/mm20/`

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