

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

A. J. Roberts*

Nov 2013 – June 22, 2021

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

1	Introduction	2
1.1	A simple example: <code>exampleslowman()</code>	5
1.2	Header of the procedure	6

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

1	<i>Introduction</i>	2
1.3	Preamble to the procedure	7
1.4	Check the dimensionality of specified system	8
2	Dissect the linear part	9
2.1	Normalise the adjoint eigenvectors	10
2.2	Operator to represent delays	10
2.3	Linearise at the origin	11
2.4	Eigen-check	12
2.5	Ameliorate the nonlinearity	13
2.6	Store invariant manifold eigenvalues	14
2.7	Precompute matrices for updates	15
2.8	Define operators that invoke these inverses	17
3	Initialise LaTeX output	19
4	Linear approximation to the invariant manifold	22
5	Iteratively construct the invariant manifold	23
6	Output text version of results	25
7	Output LaTeX version of results	27
8	Fin	29
9	Override some system procedures	29

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

¹This script changes many internal settings of *Reduce*, so best to do 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`, \dots , for time derivatives $du1/dt$, $du2/dt$, \dots ;
any time delayed variables in the RHS are coded by the time-delay in parentheses 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(...)`, usually 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 $\mathcal{O}(\varepsilon^{\text{toosmall}})$ (as $\varepsilon \rightarrow 0$). 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)`, \dots , 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, \dots , 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 i th coordinate, `ui`, of the invariant manifold as a function of `s(j)`, s_j .
- Global variable `ff_` gives the evolution on the invariant manifold, such that `coeffn(ff,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 i th component of the j th 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 2019). This particular procedure is developed from a coordinate-independent algorithm for constructing centre manifolds originally by Couillet & Spiegel (1983), adapted for human-efficient computer algebra by Roberts (1997), extended to invariant/inertial manifolds (Roberts 1989b, Foias et al. 1988), and further extended to the projection of initial conditions, forcing, uncertainty via the innovations of Roberts (1989a, 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 $\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$ 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!),

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

On this slow manifold the evolution is

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

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

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

Evaluate these at $\varepsilon = 1$ to apply to the original specified system, or here 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.

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

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

```

11 operator invariantmanifold;
12 for all odefns, evals, evecs, adjvecs, toosmall let
13   invariantmanifold(odefns, evals, evecs, adjvecs, toosmall)
14   = 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.

```

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

```

Write an intro message.

```

18 write "Construct an invariant manifold (version 21 Jun 2021)"$

```

Transpose the defining matrices so that vectors are columns.

```

19 ff := tp odefns;
20 ee := tp evecs;
21 zz := tp adjvecs;

```

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

```

22 maxiter:=29$
23 factor small;

```

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

```

24 trace:=0$
25 %trace:=1; maxiter:=5;

```

The `rationalize` switch makes code much faster with complex numbers. The switch `gcd` seems to wreck convergence, so leave it off.

```
26 on div; off allfac; on revpri;
27 on rationalize;
```

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

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

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

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

Empty the output LaTeX file in case of error.

```
36 out "invarManReport.tex";
37 write "This empty document indicates error.";
38 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.

```
39 write "total no. of variables ",
40 n:=part(length(ee),1);
41 write "no. of invariant modes ",
42 m:=part(length(ee),2);
43 if {length(evals),length(zz),length(ee),length(ff)}
```



```

44   ={{1,m},{n,m},{n,m},{n,1}}
45   then write "Input dimensions are OK"
46   else <<write "INCONSISTENT INPUT DIMENSIONS, I EXIT";
47       return>>;

```

For the moment limit to a maximum of nine components.

```

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

```

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

```

50 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.

```

51 clear exp; operator exp;
52 let { df(exp(~u),t) => df(u,t)*exp(u)
53       , exp(~u)*exp(~v) => exp(u+v)
54       , exp(~u)^^p => exp(p*u)
55       };

```

Need function `conj_` to do parsimonious complex conjugation.

```

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

```

Make an array of eigenvalues for simplicity (`evals` not used hereafter).

```

57 clear eval_; array eval_(m);
58 for j:=1:m do eval_(j):=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 $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ 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??

```

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

```

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

```

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

```

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

```

92 ll:=ee*(tp ee)*0; %zero nxn matrix

```

```

93 uzero:=(for k:=1:n collect (mkid(u,k)=0))$
94 equilibrium:=(small=0).uzero$
95 for j:=1:n do for k:=1:n do begin
96   ll(j,k):=df(ff(j,1),mkid(u,k));
97   ll(j,k):=sub(equilibrium,ll(j,k));
98 end;
99 write "Find the linear operator is";
100 write ll:=ll;

```

We need a vector of unknowns for a little while.

```

101 uvec:=0*ff; %nx1 zero matrix
102 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.

```

103 write "Check invariant subspace linearisation ";
104 for j:=1:m do for k:=1:m do aa_(j,k):=0;
105 for j:=1:m do aa_(j,j):=eval_(j);
106 % following maybe only for pure centre modes??
107 reslin:=(ll*(ee*dexp_)-(ee*dexp_)*aa_
108   where exp(~a)*d_(1,t,~dt)=>sub(t=-dt,cos(-i*a)+i*sin(-i*a))*
109 if trace then write reslin:=reslin;
110 ok:=1$
111 for j:=1:n do for k:=1:m do
112   ok:=if reslin(j,k)=0 then ok else 0$
113 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.

```

114 if not ok then for iter:=1:2 do begin
115   write "WARNING: I NEED TO ADJUST LINEAR OPERATOR";
116   write
117   lladj:=reslin*tp map(conj_(~b),zz*cexp_);
118   write
119   lladj:=(lladj where {exp(0)=>1, exp(~a)=>0 when a neq 0});
120   write
121   ll:=ll-lladj;
122   % following maybe only for pure centre modes??
123   write
124   reslin:=(ll*(ee*dexp_)-(ee*dexp_)*aa_
125     where exp(~a)*d_(1,t,~dt)=>sub(t=-dt,cos(-i*a)+i*sin(-i*a))*
126   ok:=1$
127   for j:=1:n do for k:=1:m do
128     ok:=if reslin(j,k)=0 then ok else 0$
129   if ok then iter:=iter+1000;
130   end;
131   if not ok then << write
132     "OOPS, INCONSISTENT EIGENVALUES, EIGENVECTORS AND OPERATOR.
133     EMAIL ME; I EXIT";
134   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.

```

135 somerules:=for j:=1:n collect
136   (d_(1,t,~dt)*mkid(u,j)=d_(mkid(u,j),t,dt))$
137 ll0_-:=(ll*uvec where somerules)$
138 ff:=(if 1 then small*ff

```

```

139             else ff-(1-small)*sub(small=0,ff))
140         +(1-small)*l10_$

```

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

```

141 %ff0:=(ff where uzero)$ % obliterates u1,... as operators
142 ff:=ff+(exp(0)-1)*sub(uzero,ff)$

```

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

```

143 rhsfn:=for i:=1:n sum e_(i,1)*ff(i,1)$
144 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.

```

145 rhsjact:=for i:=1:n sum for j:=1:n sum
146     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.

```

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

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

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

```
159 eval_s(0):=geneval$
160 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

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

and then put its inverse in place. Subsequently, extract the blocks for the generalised inverses and solvability condition into $\mathbf{l_invs}$ and $\mathbf{g_invs}$.

```
161 matrix llzz_(n+m,n+m);
162 clear l_invs,g_invs,l1_invs,g1_invs,l2_invs,g2_invs;
163 array l_invs(neval), g_invs(neval), l1_invs(neval),
164       g1_invs(neval), l2_invs(neval), g2_invs(neval);
165 clear sp_; operator sp_; linear sp_;
166 for k_:=0:neval do begin
167   if trace then write "ITERATION ",k_;
```

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

```
168   for ii:=1:n do for jj:=1:n do llzz_(ii,jj):=(
169     -sub(small=0,ll(ii,jj)) where d_(1,t,~dt)
170     => 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.

```
171 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.

```
172 for j:=1:length(modes_(k_)) do
173   for ii:=1:n do llzz_(ii,n+j):=ee(ii,part(modes_(k_),j))
174     +(for jj:=1:n sum
175       sp_(ll(ii,jj)*ee(jj,part(modes_(k_),j)),d_)
176       where { sp_(1,d_)=>0
177               , sp_(d_(1,t,~dt),d_)=>dt*(
178                 cos(i*eval_s(k_)*dt)+i*sin(i*eval_s(k_)*dt))
179               });
```

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

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

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

```
183 for i:=1:length(modes_(k_)) do
184   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.

```
185 for i:=length(modes_(k_))+1:m do begin
186   for j:=1:n+i-1 do llzz_(n+i,j):=llzz_(j,n+i):=0;
187   llzz_(n+i,n+i):=1;
188 end;
189 if trace then write "finished Add some trivial rows and columns";
```

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

```
190 if trace then write llzz_:=llzz_;
```



```

191 llzz_:=llzz_^(-1);
192 if trace then write llzz_:=llzz_;
193 l_invs(k_):=for i:=1:n sum for j:=1:n sum e_(i,j)*llzz_(i,j);
194 g_invs(k_):=for i:=1:length(modes_(k_)) sum
195     for j:=1:n sum e_(part(modes_(k_),i),j)*llzz_(i+n,j);
196 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.

```

197 l1_invs(k_) := for ii:=1:n sum for j:=1:n sum
198     e_(ii,j)*conj_(llzz_(j,ii));
199 l2_invs(k_) := for ii:=1:n sum
200     for j:=1:length(modes_(k_)) sum
201         e_(ii,part(modes_(k_),j))*conj_(llzz_(j+n,ii));
202 g1_invs(k_) := for ii:=1:length(modes_(k_)) sum
203     for j:=1:n sum
204         e_(part(modes_(k_),ii),j)*conj_(llzz_(j,ii+n));
205 g2_invs(k_) := for ii:=1:length(modes_(k_)) sum
206     for j:=1:length(modes_(k_)) sum
207         e_(part(modes_(k_),ii),part(modes_(k_),j))
208         *conj_(llzz_(j+n,ii+n));
209 if trace then write "finished Unpack the conjugate transpose";
210 end;

```

2.8 Define operators that invoke these inverses

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

```

211 clear l_inv; operator l_inv; linear l_inv;

```

```

212 let l_inv(e_(~j,~k)*exp(~a),exp)=>l_invproc(a/t)*e_(j,k);
213 procedure l_invproc(a);
214   if a member eval_set
215   then << k_:=0;
216       repeat k_:=k_+1 until a=eval_s(k_);
217       l_invs(k_)*exp(a*t) >>
218   else sub(geneval=a,l_invs(0))*exp(a*t)$

```

Second for the evolution on the invariant manifold.

```

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

```

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

```

227 procedure inv_proc(a,invs);
228   if a member eval_set
229   then << k_:=0;
230       repeat k_:=k_+1 until a=eval_s(k_);
231       invs(k_)*exp(a*t) >>
232   else sub(geneval=a,invs(0))*exp(a*t)$

```

Then define operators that we use to update the projection.

```

233 clear l1_inv; operator l1_inv; linear l1_inv;
234 clear l2_inv; operator l2_inv; linear l2_inv;
235 clear g1_inv; operator g1_inv; linear g1_inv;
236 clear g2_inv; operator g2_inv; linear g2_inv;
237 let { l1_inv(e_(~j,~k)*exp(~a),exp)
238       => inv_proc(a/t,l1_invs)*e_(j,k)
239       , l2_inv(e_(~j,~k)*exp(~a),exp)

```

```

240      => inv_proc(a/t,l2_invs)*e_(j,k)
241    , g1_inv(e_(~j,~k)*exp(~a),exp)
242      => inv_proc(a/t,g1_invs)*e_(j,k)
243    , g2_inv(e_(~j,~k)*exp(~a),exp)
244      => inv_proc(a/t,g2_invs)*e_(j,k)
245  };

```

3 Initialise LaTeX output

Set the default output to be inline mathematics.

```
246 mathstyle math;
```

Define the Greek alphabet with `small` as well.

```

247 defid small,name="\eps";%varepsilon;
248 defid alpha,name=alpha;
249 defid beta,name=beta;
250 defid gamma,name=gamma;
251 defid delta,name=delta;
252 defid epsilon,name=epsilon;
253 defid varepsilon,name=varepsilon;
254 defid zeta,name=zeta;
255 defid eta,name=eta;
256 defid theta,name=theta;
257 defid vartheta,name=vartheta;
258 defid iota,name=iota;
259 defid kappa,name=kappa;
260 defid lambda,name=lambda;
261 defid mu,name=mu;
262 defid nu,name=nu;
263 defid xi,name=xi;
264 defid pi,name=pi;
265 defid varpi,name=varpi;
266 defid rho,name=rho;

```

```
267 defid varrho,name=varrho;
268 defid sigma,name=sigma;
269 defid varsigma,name=varsigma;
270 defid tau,name=tau;
271 defid upsilon,name=upsilon;
272 defid phi,name=phi;
273 defid varphi,name=varphi;
274 defid chi,name=chi;
275 defid psi,name=psi;
276 defid omega,name=omega;
277 defid Gamma,name=Gamma;
278 defid Delta,name=Delta;
279 defid Theta,name=Theta;
280 defid Lambda,name=Lambda;
281 defid Xi,name=Xi;
282 defid Pi,name=Pi;
283 defid Sigma,name=Sigma;
284 defid Upsilon,name=Upsilon;
285 defid Phi,name=Phi;
286 defid Psi,name=Psi;
287 defid Omega,name=Omega;
```

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

```
288 defindex e_(down,down);
289 defid e_,name="e";
290 defindex d_(arg,down,down);
291 defid d_,name="D";
292 defindex u(down);
293 defid u1,name="u\sb1";
294 defid u2,name="u\sb2";
295 defid u3,name="u\sb3";
296 defid u4,name="u\sb4";
297 defid u5,name="u\sb5";
298 defid u6,name="u\sb6";
```

```

299 defid u7,name="u\sb7";
300 defid u8,name="u\sb8";
301 defid u9,name="u\sb9";
302 defindex s(down);
303 defid exp,name="\exp";
304 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.

```

305 clear tmp_,tmp_s,tmp_z;
306 array tmp_(n),tmp_s(m),tmp_z(m);
307 defindex tmp_(down);
308 defindex tmp_s(down);
309 defindex tmp_z(down);
310 defid tmp_,name="\dot u";
311 defid tmp_s,name="\vec e";
312 defid tmp_z,name="\vec z";
313 rhs_:=rhsfn$
314 for k:=1:m do tmp_s(k):={for j:=1:n collect ee(j,k),exp(eval_(k))};
315 for k:=1:m do tmp_z(k):={for j:=1:n collect zz(j,k),exp(eval_(k))};

```

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.

```

316 out "scratchfile.red";
317 write "off echo;"$ % do not understand why needed in 2021??
318 write "write ""\
319 \paragraph{The specified dynamical system}
320 \("";";
321 for j:=1:n do write "tmp_(",j,"):=coeffn(rhs_,e_("",j,",1),1);";
322 write "write ""\
323 \paragraph{Invariant subspace basis vectors}
324 \("";";
325 for j:=1:m do write "tmp_s(",j,"):=tmp_s(",j,");";
326 for j:=1:m do write "tmp_z(",j,"):=tmp_z(",j,");";
327 write "end;";

```

```
328 shut "scratchfile.red";
```

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

```
329 write "Ignore the following 15 lines of LaTeX"$
330 on latex$
331 out "invarManReportSys.tex"$
332 in "scratchfile.red"$
333 shut "invarManReportSys.tex"$
334 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.

```
335 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.

```
336 clear gg;
337 clear s; operator s; depend s,t;
338 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.

```
339 procedure manifold_(uu,n);
340   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.

```

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

```

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

```

345 procedure matify_(a,m,n)$
346   begin matrix z(m,n);
347     for i:=1:m do for j:=1:n do z(i,j):=coeffn(a,e_(i,j),1);
348     return (z where {exp(0)=>1,small=>s});
349   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.

```

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

```

5 Iteratively construct the invariant manifold

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

```

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

```

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

```

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

```

Iteratively construct the invariant manifold.

```

356 write "Start iterative construction of invariant manifold";
357 for iter:=1:maxiter do begin

```

```

358 if trace then write "
359 ITERATION = ",iter,"
360 -----";

```

Compute residual vector (matrix) of the dynamical system [Roberts \(1997\)](#).

```

361 resde:=-df(uu,t)+sub(manifold_(uu,n),rhsfn);
362 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.

```

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

```

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

```

366 if itisSlowMan_ then begin
367   jacadj:=conj_(sub(manifold_(uu,n),rhsjact));
368   if trace then write "jacadj=",matify_(jacadj,n,n);
369   resd:=df(zs,t)+jacadj*zs+zs*pp;
370   if trace then write "resd=",matify_(resd,n,m);

```

Compute residual of the normalisation of the projection.

```

371   resz:=est*zs-eyem*exp(0);
372   if trace then write "resz=",matify_(resz,m,m);
373 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).

```

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

```

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

```

375 uu:=uu+l_inv(resde,exp);
376 gg:=gg+g_inv(resde,exp);
377 if trace then write "gg=",matify_(gg,m,1);

```



```
378 if trace then write "uu=",matify_(uu,n,1);
```

Now update the isochron projection, with normalisation.

```
379 if itisSlowMan_ then begin
380   zs:=zs+l1_inv(resd,exp)-l2_inv(resz,exp);
381   pp:=pp-g1_inv(resd,exp)+youshouldnotseethis*g2_inv(resz,exp);
382   if trace then write "zs=",matify_(zs,n,m);
383   if trace then write "pp=",matify_(pp,m,m);
384 end;
```

Terminate the iteration loop once residuals are zero.

```
385 showtime;
386 if {resde,resd,resz}={0,0,0} then write iter:=iter+10000;
387 end;
```

Only proceed to print if terminated successfully.

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

6 Output text version of results

Once construction is finished, simplify $\exp(0)$.

```
392 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.

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

```

397 for j:=1:m do write "ds(",j,")/dt = ",
398   coeffn(gg,e_(j,1),1);

```

Optionally write the projection vectors.

```

399 if itisSlowMan_ then begin write "
400 The normals to the isochrons at the slow manifold.
401 Use these vectors: to project initial conditions
402 onto the slow manifold; to project non-autonomous
403 forcing onto the slow evolution; to predict the
404 consequences of modifying the original system; in
405 uncertainty quantification to quantify effects on
406 the model of uncertainties in the original system.";
407   for j:=1:m do write "z",j," = ",
408     for i:=1:n collect coeffn(zs,e_(i,j),1);
409 end;

```

Write text results numerically evaluated when expressions are long.

```

410 if length(gg)>30 then begin
411 on rounded; print_precision 4$
412 write "Numerically, the invariant manifold is (to one order lower
413 for j:=1:n do write "u",j," = ",
414   coeffn(small*uu,e_(j,1),1)/small;
415 write "Numerically, the evolution of the real/complex amplitudes
416 for j:=1:m do write "ds(",j,")/dt = ",
417   coeffn(gg,e_(j,1),1);
418 if itisSlowMan_ then begin
419   write "Numerically, normals to isochrons at slow manifold.";
420   for j:=1:m do write "z",j," = ",
421     for i:=1:n collect coeffn(zs,e_(i,j),1);
422 end;
423 off rounded;
424 end;

```

7 Output LaTeX version of results

Change the printing of temporary arrays.

```
425 clear tmp_zz; array tmp_zz(m,n);
426 defid tmp_,name="u";
427 defid tmp_s,name="\dot s";
428 defid tmp_z,name="\vec z";
429 defid tmp_zz,name="z";
430 defindex tmp_zz(down,down);
```

Gather complicated result

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

Write to a file the commands needed to write the LaTeX expressions. Write the invariant manifold to one order lower than computed.

```
433 out "scratchfile.red";
434 write "off echo;"$ % do not understand why needed in 2021??
435 write "write ""\
436 \paragraph{The invariant manifold}
437 These give the location of the invariant manifold in
438 terms of parameters~\!(s\sb j\!).
439 \("";";
440 for j:=1:n do write "tmp_(",j,
441    "):=coeffn(small*uu,e_(",j,",1),1)/small;";
442 if length(gg)>30 then begin
443 write "on rounded; print_precision 4$"$
444 for j:=1:n do write "tmp_(",j,
445    "):=coeffn(small*uu,e_(",j,",1),1)/small;";
446 write "off rounded;"$
447 end;
```

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

```
448 write "write ""\)
```

```

449 \paragraph{Invariant manifold ODEs}
450 The system evolves on the invariant manifold such
451 that the parameters evolve according to these ODEs.
452 \("");";
453 for j:=1:m do write "tmp_s(",j,"):=1*coeffn(gg,e_(",j,",1),1);";
454 if length(gg)>30 then begin
455 write "on rounded; print_precision 4$"$
456 for j:=1:m do write "tmp_s(",j,"):=1*coeffn(gg,e_(",j,",1),1);";
457 write "off rounded;"$
458 end;

```

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

```

459 if itisSlowMan_ then begin
460   write "write ""\)"
461 \paragraph{Normals to isochrons at the slow manifold}
462 Use these vectors: to project initial conditions
463 onto the slow manifold; to project non-autonomous
464 forcing onto the slow evolution; to predict the
465 consequences of modifying the original system; in
466 uncertainty quantification to quantify effects on
467 the model of uncertainties in the original system.
468 The normal vector \(\vec z\sb{j:=(z\sb{j1},\ldots,z\sb{jn})}\)
469 \("");";
470   for i:=1:m do for j:=1:n do
471     write "tmp_zz(",i,",",j,"):=tmp_zz(",i,",",j,")";
472 end;

```

Finish the scratchfile.

```

473 write "end;";
474 shut "scratchfile.red";

```

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

```

475 out "invarManReport.tex"$

```

```

476 on latex$
477 in "scratchfile.red"$
478 off latex$
479 shut "invarManReport.tex"$

```

8 Fin

That's all folks, so end the procedure.

```

480 return Finished_constructing_invariant_manifold_of_system$
481 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.

```

482 %load_package rlfi; %must be loaded early
483 deflist('(( ( !\b!i!g!() (!) !\b!i!g!)) (!P!I !\p!i! )
484          (!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.

```

485 symbolic procedure prinlaend;
486 <<terpri();
487   prin2t "\\par";
488   if !*verbatim then
489     <<prin2t "\\begin{verbatim}";
490     prin2t "REDUCE Input:">>;
491   ncharspr!*:=0;
492   if ofl!* then linelength(car linel!*)

```

```

493     else laline!*: =cdr line1!*;
494     nochar!*: =append(nochar!*,nochar1!*);
495     nochar1!*: =nil >>$
496     %

```

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

```

497 symbolic procedure prinlabegin;
498 <<if !*verbatim then
499     <<terpri();
500     prin2t "\end{verbatim}">>;
501     line1!*: =linelength nil . laline!*;
502     if ofl!* then linelength(laline!* + 2)
503     else laline!*: =car line1!* - 2;
504     prin2t "\" >>$

```

Override the procedure that outputs the L^AT_EX 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.

```

505 symbolic procedure latexon;
506 <<!!*a2sfn:= 'texaeval;
507     !*raise:=nil;
508     prin2t "\documentclass[11pt,a5paper]{article}";
509     prin2t "\usepackage[a5paper,margin=13mm]{geometry}";
510     prin2t "\usepackage{parskip,time} \raggedright";
511     prin2t "\def\exp\big(#1\big){\, ,{\rm e}^{\#1}}";
512     prin2t "\def\eps{\varepsilon}";
513     prin2t "\title{Invariant manifold of your dynamical system}";
514     prin2t "\author{A. J. Roberts, University of Adelaide\\}";
515     prin2t "\texttt{http://orcid.org/0000-0001-8930-1552}}";
516     prin2t "\date{\now, \today}";
517     prin2t "\begin{document}";
518     prin2t "\maketitle";
519     prin2t "Throughout and generally: the lowest order, most";
520     prin2t "important, terms are near the end of each expression."

```

```

521 prin2t "\input{invarManReportSys}";
522 if !*verbatim then
523     <<prin2t "\begin{verbatim}";
524     prin2t "REDUCE Input:">>;
525 put('tex','rtypefn','(lambda(x) 'tex)) >>$

```

End the file when read by Reduce

```

526 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’, *Phys. Lett. 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. (1989a), ‘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. (1989b), ‘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. (2019), Backwards theory supports modelling via invariant manifolds for non-autonomous dynamical systems, Technical report, [<http://arxiv.org/abs/1804.06998>].