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

A. J. Roberts*

Nov 2013 – April 3, 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

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

1 Introduction

Invoke the analysis to construct an invariant manifold by executing the following command with specific values for the input parameters.

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

Inputs As in the example of the next Section 1.1, 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;

- `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;

- `adjvecs`, a comma separated list of vectors within `mat(...)`, 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.

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.

1.1 A simple example: `exampleslowman()`

Execute this example by invoking the command `exampleslowman();` The example system to analyse is specified to be

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

```
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 $\vec{e}^{\ev} = (1,1)$, and corresponding left-eigenvector is $\vec{z}^{\zv} = \vec{e}^{\ev} = (1,1)$, as specified. The last parameter specifies to construct the slow manifold to errors $\mathcal{O}\big(\varepsilon^5\big)$.

The procedure actually analyses the embedding system

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

So 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 $\vec{u}^{\uv} = (u_1, u_2)$. Hence the specified error $\mathcal{O}\big(\varepsilon^5\big)$ is here the same as error $\mathcal{O}\big(|\vec{u}^{\uv}|^6\big)$.

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

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

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.

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

Now define the procedure as an operator so we can be flexible with its arguments.

```
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 all scalars and most matrices to be declared here.

```
15 scalar ff, freqm, ee, zz, maxiter, ff0, trace, ll, uvec,
16 reslin, ok, rhsjact, jacadj, resd, resde, resz,
17 rhsfn, zs, pp, est, eyem;
```

Transpose the defining matrices so that vectors are columns.

```
18 ff := tp odefns;
19 freqm := -i*evals;
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;
```

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

```
28 operator e_;
```

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

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

```
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 specification of the dynamical system: 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(freqm),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.

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

2 Dissect the linear part

Define complex exponential `cis`$(u) = e^{iu}$. Do not (yet) invoke the simplification of `cis`(0) as I want it to label modes of no oscillation, zero frequency.

```
52 operator cis;
53 let { df(cis(~u),t) => i*df(u,t)*cis(u)
54     , cis(~u)*cis(~v) => cis(u+v)
55     , cis(~u)^~p => cis(p*u)
56     };
```

Need function `conj_` to do parsimonious complex conjugation.

```
57 operator cis__;
58 procedure conj_(a)$ begin scalar aa;
59     aa:=(a where {i=>i__, cis(~b)=>cis__(b) when b neq 0})$
60     return (aa where {i__=>-i,cis__(~b)=>cis(-b)})$
61     end$
```

Make another array of frequencies for simplicity.

```
62 array freq_(m);
63 for j:=1:m do freq_(j):=freqm(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, $e^{i\omega t}$, 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.

Reduce implements `conj` via `repart` and `impart`, so let `repart` do the conjugation of the cis factors.

Note: the 'left eigenvectors' have to be the eigenvectors of the complex conjugate transpose, and for the complex conjugate frequency. 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 un/stable manifolds we have to cope with complex frequencies. Seems to need `zz` to have complex conjugated frequency so store in `ccis_`—which is the same as `dcis_` for real frequencies.

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

2.2 Operator to represent delays

Introduce an operator to represent delay factors more conveniently for analysis.

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

```
82      };
```

Now rewrite the (delay) factors in terms of this operator. Need to say that the symbol `u` depends upon time; later we write things into `u` and this dependence would be forgotten. For the moment limit to a maximum of nine ODEs.

```
83 somerules:={}$
84 depend u1,t; somerules:=(u1(~dt)=d_(u1,t,dt)).somerules$
85 depend u2,t; somerules:=(u2(~dt)=d_(u2,t,dt)).somerules$
86 depend u3,t; somerules:=(u3(~dt)=d_(u3,t,dt)).somerules$
87 depend u4,t; somerules:=(u4(~dt)=d_(u4,t,dt)).somerules$
88 depend u5,t; somerules:=(u5(~dt)=d_(u5,t,dt)).somerules$
89 depend u6,t; somerules:=(u6(~dt)=d_(u6,t,dt)).somerules$
90 depend u7,t; somerules:=(u7(~dt)=d_(u7,t,dt)).somerules$
91 depend u8,t; somerules:=(u8(~dt)=d_(u8,t,dt)).somerules$
92 depend u9,t; somerules:=(u9(~dt)=d_(u9,t,dt)).somerules$
93 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 small any non-zero forcing 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??

```
94 ll:=ee*(tp ee)*0; %zero nxn matrix
95 uzero:=(for k:=1:n collect (mkid(u,k)=0))$
96 equilibrium:=(small=0).uzero$
97 for j:=1:n do for k:=1:n do begin
98   ll(j,k):=df(ff(j,1),mkid(u,k));
99   ll(j,k):=sub(equilibrium,ll(j,k));
100 end;
101 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 frequencies. Check that the frequencies 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):=i*freq_(j);
reslin:=(ll*(ee*dcis_)-(ee*dcis_)*aa_
    where cis(~a)*d_(1,t,~dt)=>sub(t=-dt,cos(a)+i*sin(a))*cis(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 $\mathcal{L}^{\mathrm{L}}_{\mathrm{new}} := \mathcal{L}^{\mathrm{L}} - \mathcal{L}^{\mathrm{L}}_{\mathrm{adj}}$ 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*ccis_);
write
lladj:=(lladj where {cis(0)=>1, cis(~a)=>0 when a neq 0});
write
ll:=ll-lladj;
```

```
write
reslin:=(ll*(ee*dcis_)-(ee*dcis_)*aa_
    where cis(~a)*d_(1,t,~dt)=>sub(t=-dt,cos(a)+i*sin(a))*cis(a)
%for j:=1:n do for k:=1:m do
%    if reslin(j,k) neq 0 then << write
%    "OOPS, INCONSISTENT EIGENVALUES, EIGENVECTORS AND OPERATOR.
%    EMAIL ME; I QUIT"; write reslin:=reslin; rederr "aaaaah";qu
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 ?? but the current if always chooses the first simple alternative.

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

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

```
ff0:=(ff where uzero)$
ff:=ff+(cis(0)-1)*ff0$
```

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)$
```

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

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

2.6 Store invariant manifold frequencies

Extract all the frequencies 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 frequency. Remember the frequency set is not in the 'correct' order. Array `modes_` stores the set of indices of all the modes of a given frequency.

```
array freq_s(m),modes_(m);
nfreq:=0$ freq_set:={}$
for j:=1:m do if not(freq_(j) member freq_set) then begin
  nfreq:=nfreq+1;
  freq_s(nfreq):=freq_(j);
  freq_set:=freq_(j).freq_set;
  modes_(nfreq):=for k:=j:m join
    if freq_(j)=freq_(k) then {k} else {};
end;
```

Set a flag for the case of a slow manifold when all frequencies 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.

```
itisSlowMan_:=if freq_set={0} then 1 else 0$
if trace then write itisSlowMan_:=itisSlowMan_;
```

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

```
freq_s(0):=genfreq$
modes_(0):={}$
```

2.7 Precompute matrices for updates

Precompute matrices to solve for updates for each of the critical frequencies, 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`.

```
matrix llzz_(n+m,n+m);
array l_invs(nfreq),g_invs(nfreq);
array l1_invs(nfreq),g1_invs(nfreq),l2_invs(nfreq),g2_invs(nfreq)
operator sp_; linear sp_;
for k:=0:nfreq do begin
  if trace then write "ITERATION ",k:=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(freq_s(k)*dt)-i*sin(freq_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):=i*freq_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))
```

```
175       +(for jj:=1:n sum
176         sp_(ll(ii,jj)*ee(jj,part(modes_(k),j)),d_)
177         where { sp_(1,d_)=>0
178               , sp_(d_(1,t,~dt),d_)=>dt*(
179                 cos(freq_s(k)*dt)-i*sin(freq_s(k)*dt))
180               });
```

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

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

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

```
184   for i:=1:length(modes_(k)) do
185     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 frequencies.

```
186   for i:=length(modes_(k))+1:m do begin
187     for j:=1:n+i-1 do llzz_(n+i,j):=llzz_(j,n+i):=0;
188     llzz_(n+i,n+i):=1;
189   end;
190   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.

```
191   if trace then write llzz_:=llzz_;
192   llzz_:=llzz_^(-1);
193   if trace then write llzz_:=llzz_;
194   l_invs(k):=for i:=1:n sum for j:=1:n sum e_(i,j)*llzz_(i,j);
195   g_invs(k):=for i:=1:length(modes_(k)) sum
196     for j:=1:n sum e_(part(modes_(k),i),j)*llzz_(i+n,j);
197 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. Will it need to be more subtle for the un/stable manifolds case??

```
198 %  realgenfreq:={repart(genfreq)=>genfreq, impart(genfreq)=>0}$
199   l1_invs(k):=for ii:=1:n sum for j:=1:n sum
200       e_(ii,j)*conj_(llzz_(j,ii));
201   l2_invs(k):=for ii:=1:n sum for j:=1:length(modes_(k)) sum
202       e_(ii,part(modes_(k),j))*conj_(llzz_(j+n,ii));
203   g1_invs(k):=for ii:=1:length(modes_(k)) sum for j:=1:n sum
204       e_(part(modes_(k),ii),j)*conj_(llzz_(j,ii+n));
205   g2_invs(k):=
206     for ii:=1:length(modes_(k)) sum for j:=1:length(modes_(k)) su
207       e_(part(modes_(k),ii),part(modes_(k),j))*conj_(llzz_(j+n,ii
208   if trace then write "finished Unpack the conjugate transpose";
209 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.

```
210 operator l_inv; linear l_inv;
211 let l_inv(e_(~j,~k)*cis(~a),cis)=>l_invproc(a/t)*e_(j,k);
212 procedure l_invproc(a);
213   if a member freq_set
214   then << k:=0;
215     repeat k:=k+1 until a=freq_s(k);
216     l_invs(k)*cis(a*t) >>
217   else sub(genfreq=a,l_invs(0))*cis(a*t)$
```

Second for the evolution on the invariant manifold.

```
operator g_inv; linear g_inv;
let g_inv(e_(~j,~k)*cis(~a),cis)=>ginv_proc(a/t)*e_(j,k);
procedure ginv_proc(a);
  if a member freq_set
  then << k:=0;
    repeat k:=k+1 until a=freq_s(k);
    g_invs(k) >>
  else sub(genfreq=a,g_invs(0))$
```

Copy and adjust the above for the projection. But first define the generic procedure. Perhaps use conjugate/negative of the frequency when applying to the general case of oscillations—but it might already have been accounted for??

```
procedure inv_proc(a,invs);
  if a member freq_set
  then << k:=0;
    repeat k:=k+1 until a=freq_s(k);
    invs(k)*cis(a*t) >>
  else sub(genfreq=a,invs(0))*cis(a*t)$
```

Then define operators that we use to update the projection.

```
operator l1_inv; linear l1_inv;
operator l2_inv; linear l2_inv;
operator g1_inv; linear g1_inv;
operator g2_inv; linear g2_inv;
let { l1_inv(e_(~j,~k)*cis(~a),cis)=>inv_proc(a/t,l1_invs)*e_(j,k
    , l2_inv(e_(~j,~k)*cis(~a),cis)=>inv_proc(a/t,l2_invs)*e_(j,k
    , g1_inv(e_(~j,~k)*cis(~a),cis)=>inv_proc(a/t,g1_invs)*e_(j,k
    , g2_inv(e_(~j,~k)*cis(~a),cis)=>inv_proc(a/t,g2_invs)*e_(j,k
    };
```

3 Initialise LaTeX output

Set the default output to be inline mathematics.

```
241 mathstyle math;
```

Define the Greek alphabet with `small` as well.

```
242 defid small,name="\eps";%varepsilon;
243 %defid small,name=varepsilon;
244 defid alpha,name=alpha;
245 defid beta,name=beta;
246 defid gamma,name=gamma;
247 defid delta,name=delta;
248 defid epsilon,name=epsilon;
249 defid varepsilon,name=varepsilon;
250 defid zeta,name=zeta;
251 defid eta,name=eta;
252 defid theta,name=theta;
253 defid vartheta,name=vartheta;
254 defid iota,name=iota;
255 defid kappa,name=kappa;
256 defid lambda,name=lambda;
257 defid mu,name=mu;
258 defid nu,name=nu;
259 defid xi,name=xi;
260 defid pi,name=pi;
261 defid varpi,name=varpi;
262 defid rho,name=rho;
263 defid varrho,name=varrho;
264 defid sigma,name=sigma;
265 defid varsigma,name=varsigma;
266 defid tau,name=tau;
267 defid upsilon,name=upsilon;
268 defid phi,name=phi;
269 defid varphi,name=varphi;
270 defid chi,name=chi;
271 defid psi,name=psi;
272 defid omega,name=omega;
273 defid Gamma,name=Gamma;
```

```
defid Delta,name=Delta;
defid Theta,name=Theta;
defid Lambda,name=Lambda;
defid Xi,name=Xi;
defid Pi,name=Pi;
defid Sigma,name=Sigma;
defid Upsilon,name=Upsilon;
defid Phi,name=Phi;
defid Psi,name=Psi;
defid Omega,name=Omega;

defindex e_(down,down);
defid e_,name="e";
defindex d_(arg,down,down);
defid d_,name="D";
defindex u(down);
defid u1,name="u\sb1";
defid u2,name="u\sb2";
defid u3,name="u\sb3";
defid u4,name="u\sb4";
defid u5,name="u\sb5";
defid u6,name="u\sb6";
defid u7,name="u\sb7";
defid u8,name="u\sb8";
defid u9,name="u\sb9";
defindex s(down);
defid cis,name="\cis";
defindex cis(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.

```
array tmp_(n),tmp_s(m),tmp_z(m);
defindex tmp_(down);
defindex tmp_s(down);
defindex tmp_z(down);
```

```
defid tmp_,name="\dot u";
defid tmp_s,name="\vec e";
defid tmp_z,name="\vec z";
rhs_:=rhsfn$
for k:=1:m do tmp_s(k):={for j:=1:n collect ee(j,k),cis(freq_(k)*
for k:=1:m do tmp_z(k):={for j:=1:n collect zz(j,k),cis(freq_(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.

```
out "scratchfile.red";
write "off echo;"$  % do not understand why needed in 2021??
write "write ""\)
\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 15 lines of LaTeX"$
on latex$
out "invarManReportSys.tex"$
in "scratchfile.red"$
shut "invarManReportSys.tex"$
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.

```
330 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.

```
331 operator s; depend s,t;
332 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.

```
333 procedure manifold_(uu);
334     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 frequencies 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.

```
335 uu_:=for j:=1:m sum s(j)*cis(freq_(j)*t)
336   *(for k:=1:n sum e_(k,1)*ee(k,j))$
337 gg_:=0$
338 if trace then write uu_:=uu_;
```

For some temporary trace printing??

```
339 procedure matify(a,m,n)$
340   begin matrix z(m,n);
341     for i:=1:m do for j:=1:n do z(i,j):=coeffn(a,e_(i,j),1);
342     return (z where {cis(0)=>1,small=>s});
343     end$
```

For the isochron may need to do something different with frequencies, 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 cis(freq_(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).

```
%write rhsfn:=rhsfn;
%write manifolduu:=manifold_(uu_);
%write duudt:=df(uu_,t);
resde:=-df(uu_,t)+sub(manifold_(uu_),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);
```

```
362 if trace then write "est=",matify(est,m,n);
```

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

```
363 if itisSlowMan_ then begin
364     jacadj:=conj_(sub(manifold_(uu_),rhsjact));
365     if trace then write "jacadj=",matify(jacadj,n,n);
366     resd:=df(zs,t)+jacadj*zs+zs*pp;
367     if trace then write "resd=",matify(resd,n,m);
```

Compute residual of the normalisation of the projection.

```
368     resz:=est*zs-eyem*cis(0);
369     if trace then write "resz=",matify(resz,m,m);
370 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).

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

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

```
372 uu_:=uu_+l_inv(resde,cis);
373 gg_:=gg_+g_inv(resde,cis);
374 if trace then write "gg=",matify(gg_,m,1);
375 if trace then write "uu=",matify(uu_,n,1);
```

Now update the isochron projection, with normalisation.

```
376 if itisSlowMan_ then begin
377 zs:=zs+l1_inv(resd,cis)-l2_inv(resz,cis);
378 pp:=pp-g1_inv(resd,cis)+youshouldnotseethis*g2_inv(resz,cis);
379 if trace then write "zs=",matify(zs,n,m);
380 if trace then write "pp=",matify(pp,m,m);
381 end;
```

Terminate the loop once residuals are zero.

```
382 showtime;
```

```
383 if {resde,resd,resz}={0,0,0} then write iter:=iter+10000;
384 end;
```

Only proceed to print if terminated successfully.

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

6 Output text version of results

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

```
389 let cis(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.

```
390 write "The invariant manifold is (to one order lower)";
391 for j:=1:n do write "u",j," = ",
392   coeffn(small*uu_,e_(j,1),1)/small;
393 write "The evolution of the real/complex amplitudes";
394 for j:=1:m do write "ds(",j,")/dt = ",
395   coeffn(gg_,e_(j,1),1);
```

Optionally write the projection vectors.

```
396 if itisSlowMan_ then begin
397   write "The normals to the isochrons at the slow manifold.
398 Use these vectors: to project initial conditions
399 onto the slow manifold; to project non-autonomous
400 forcing onto the slow evolution; to predict the
401 consequences of modifying the original system; in
402 uncertainty quantification to quantify effects on
403 the model of uncertainties in the original system.";
```

```
404   for j:=1:m do write "z",j," = ",
405     for i:=1:n collect coeffn(zs,e_(i,j),1);
406 end;
```

Write text results numerically evaluated when expressions are long.

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

There is an as yet unresolved problem in the typesetting when the argument of `cis` (frequency) is a rational number instead of integer: the numerator has an extra pair of parentheses which then makes the typesetting wrong; maybe we need a pre-LaTeX filter??

7 Output LaTeX version of results

Change the printing of temporary arrays.

```
422 array tmp_zz(m,n);
423 defid tmp_,name="u";
424 defid tmp_s,name="\dot s";
425 defid tmp_z,name="\vec z";
426 defid tmp_zz,name="z";
```

```
427 defindex tmp_zz(down,down);
```

Gather complicated result

```
428 %for k:=1:m do tmp_z(k):=for j:=1:n collect (1*coeffn(zs,e_(j,k),
429 for k:=1:m do for j:=1:n do 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.

```
430 out "scratchfile.red";
431 write "off echo;"$ % do not understand why needed in 2021??
432 write "write ""\)
433 \paragraph{The invariant manifold}
434 These give the location of the invariant manifold in
435 terms of parameters~\(s\sb j\).
436 \("";";
437 for j:=1:n do write "tmp_(",j,
438   "):=coeffn(small*uu_,e_(",j,",1),1)/small;";
```

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

```
439 write "write ""\)
440 \paragraph{Invariant manifold ODEs}
441 The system evolves on the invariant manifold such
442 that the parameters evolve according to these ODEs.
443 \("";";
444 for j:=1:m do write "tmp_s(",j,"):=1*coeffn(gg_,e_(",j,",1),1);"
```

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

```
445 if itisSlowMan_ then begin
446   write "write ""\)
447 \paragraph{Normals to isochrons at the slow manifold}
448 Use these vectors: to project initial conditions
449 onto the slow manifold; to project non-autonomous
450 forcing onto the slow evolution; to predict the
451 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 "tmp_zz(",i,",",j,"):=tmp_zz(",i,",",j,");";
end;
```

Finish the scratchfile.

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

Execute the file 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.

```
return Finished_constructing_invariant_manifold_of_system$
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.

```
%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.

```
%write "Ignore immediately following messages";
symbolic procedure prinlaend;
<<terpri();
  prin2t "\)\par";
  if !*verbatim then
      <<prin2t "\begin{verbatim}";
        prin2t "REDUCE Input:">>;
  ncharspr!*:=0;
  if ofl!* then linelength(car linel!*)
    else laline!*:=cdr linel!*;
  nochar!*:=append(nochar!*,nochar1!*);
  nochar1!*:=nil >>$
  %
```

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

```
symbolic procedure prinlabegin;
% Initializes the output
<<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.

```
493 symbolic procedure latexon;
494 <<!*!*a2sfn:='texaeval;
495   !*raise:=nil;
496   prin2t "\documentclass[11pt,a5paper]{article}";
497   prin2t "\usepackage[a5paper,margin=13mm]{geometry}";
498   prin2t "\usepackage{parskip,time} \raggedright";
499   prin2t "\def\cis\big(#1\big){\,e^{#1i}}";
500   prin2t "\def\eps{\varepsilon}";
501   prin2t "\title{Invariant manifold of your dynamical system}";
502   prin2t "\author{A. J. Roberts, University of Adelaide\\";
503   prin2t "\texttt{http://orcid.org/0000-0001-8930-1552}}";
504   prin2t "\date{\now, \today}";
505   prin2t "\begin{document}";
506   prin2t "\maketitle";
507   prin2t "Throughout and generally: the lowest order, most";
508   prin2t "important, terms are near the end of each expression.";
509   prin2t "\input{invarManReportSys}";
510   if !*verbatim then
511       <<prin2t "\begin{verbatim}";
512         prin2t "REDUCE Input:">>;
513   put('tex,'rtypefn,'(lambda(x) 'tex)) >>$
```

References

Roberts, A. J. (1989), '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. (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.

End the file when read by Reduce

```
514 end;
```