Computer algebra derives invariant manifolds and/or normal forms of general stochastic or non-autonomous multiscale differential equations

A. J. Roberts*

November 19, 2008; modified May 30, 2023

Abstract

Provides a coded procedure to construct invariant manifolds or a normal form of a wide class of systems of non-autonomous or stochastic differential equations (herein abbreviated by S/ODEs). The methodology is based upon earlier research (Cox & Roberts 1991, Chao & Roberts 1996, Roberts 2008). Interpret all S/ODEs in the Stratonovich sense so the analysis applies to deterministic differential equations, both non-autonomous and autonomous. Cater for deterministic autonomous systems by simply omitting the 'noise'. For generality, this coded procedure now caters for unstable modes, and for differential equation systems with a rational right-hand side. This code also underlies an interactive web service (Roberts 2009–2023). Modelling stochastic systems has many important applications. The constructed stochastic coordinate transforms and associated invariant manifolds are a powerful way of disentangling emergent long term dynamics.

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/StoNormForm.zip` Within

the folder `StoNormForm`, start-up *Reduce* and load the procedure by executing the command `in_tex "stoNormForm.tex"$` [1] Test your installation by then executing `examplenormform(csuman);` (see Section 1.1).

Execution Thereafter, construct a specified invariant manifold/normal form of a specific dynamical system by executing the following command with specific values for the input parameters. See `manyExamples.pdf` for many and varied examples.

```
1 stonormalform(dxdt, dydt, dzdt, toosmall, theman);
```

Inputs Write your S/ODE system in terms of slow variables $x_j(t)$, fast stable variables $y_j(t)$ (linearly decaying), and fast unstable variables $z_j(t)$ (linearly growing). For an S/ODE with n_x slow modes, n_y fast stable modes, and/or n_z fast unstable modes, you must denote the slow modes by `x(1)` through to `x(nx)` (or aliases `x1`,...), the stable fast modes by `y(1)` through to `y(ny)` (or aliases `y1`,...), and the unstable fast modes by `z(1)` through to `z(nz)` (or aliases `z1`,...). Each non-autonomous factor must be denoted by `w(.)` where the dot denotes almost any label you care to choose: simple numbers such as `w(1)` and/or `w(2)` are the usual choices (no aliases); but other labels may be used. Often `w(.)` is a Stratonovich white noise, a derivative of a Stratonovich Wiener process. Analyse deterministic, autonomous, systems by omitting any noise term `w()` in the differential equations. The S/ODEs must be linearly diagonalised. [2] Then, as in the example of the next Section 1.1, the input parameters to the procedure are the following:

- `dxdt`, a comma separated list within braces, {...}, of the right-hand sides of the S/ODEs for the slow variables $x_j(t)$ (the list may be empty `{}`);
- `dydt`, a comma separated list within braces, {...}, of the right-hand sides of the S/ODEs for the fast stable variables $y_j(t)$ (the list may be

[1] This script changes many internal settings of *Reduce*, so best to do only when needed.
[2] Although a Jordan form is also acceptable, there are issues in the error control. Also, 'small' off-diagonal terms are allowed.

empty `{}`);

- `dzdt`, a comma separated list within braces, $\{\ldots\}$, of the right-hand sides of the S/ODEs for the fast unstable variables $z_j(t)$ (the list may be empty `{}`);

- `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/normal form, 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 reinterpret the results, and errors, in terms of your systems actual 'small' parameters.

  The code *also* truncates to errors $\mathcal{O}\big(\sigma^3\big)$ where σ characterises the magnitude of the non-autonomous/stochastic effects `w(j)`.

- `theman`, *optional*, *one* of the identifiers in the set {`cman` , `sman` , `uman` , `csman` , `cuman` , `csuman`} specifying which invariant manifold to construct: respectively, centre (slow), stable, unstable, centre-stable, centre-unstable, or a normal form coordinate transform of the entire centre-stable-unstable space. If omitted, then the default is `csuman`.

The above right-hand side expressions for the time-derivatives must be multinomial in variables x_i, y_i, z_i and w_i. To cater for rational function right-hand sides, one also may include some $\dot{x}_i$, $\dot{y}_i$, and $\dot{z}_i$ factors.

Outputs This procedure reports the embedded system it actually analyses, the number of iterations taken, the constructed time dependent coordinate transform (the original variables $(\vec{x},\vec{y},\vec{z})$ as a function of the new variables $(\vec{X},\vec{Y},\vec{Z})$), and the corresponding evolution in the new variables in terms of S/ODEs for $(\vec{X},\vec{Y},\vec{Z})$.

- A plain report to the Terminal window in which *Reduce* is executing.

- A LaTeX source report written to the file `stoNFreport.tex` (with

`stoNFreportHdr.tex` and `stoNFreportSys.tex`). Generate a pdf version by executing `pdflatex stoNFreport`.

- Global arrays such that `x_(i)`, `y_(j)`, and `z_(k)`, respectively, are the normal form coordinate transforms or invariant manifold coordinates $x_i(\vec{X},\vec{Y},\vec{Z},\sigma,\varepsilon)$, $y_j(\vec{X},\vec{Y},\vec{Z},\sigma,\varepsilon)$, and $z_k(\vec{X},\vec{Y},\vec{Z},\sigma,\varepsilon)$. Depending upon the requested invariant manifold, the dependence upon appropriate ones of $\vec{X}$, $\vec{Y}$, or $\vec{Z}$ does not exist.
- Global arrays `ff(i)`, `gg(j)`, and `hh(k)` give the corresponding evolution $\dot{X}_i =$ `ff(i)`, $\dot{Y}_j =$ `gg(j)`, and $\dot{Z}_k =$ `hh(j)` —all as functions of $(\vec{X},\vec{Y},\vec{Z},\sigma,\varepsilon)$, or a subset thereof as appropriate for the specified invariant manifold.

One may change the appearance of the output somewhat. For example, in order to group terms in output expressions, execute a `factor` command before executing the procedure `stonormalform(...)`.

Background The theoretical support for the results of the analysis of this procedure is centre/stable/unstable manifold theory (e.g., Carr 1981, Knobloch & Aulbach 1982, Haragus & Iooss 2011, Roberts 2015), and an embryonic backwards theory (Roberts 2019). This particular procedure is developed from that for human-efficient computer algebra (Roberts 1997), and extended to stochastic/non-autonomous systems (Chao & Roberts 1996, Roberts 2008).

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: `examplenormform()`

Execute this example by invoking the command `examplenormform(csuman);` The classically basic non-trivial system of fast/slow S/ODEs (Roberts 2015, §19.1) is

$$\dot{x} = -xy \quad \text{and} \quad \dot{y} = -y + x^2 - 2y^2 + \sigma w(t),$$

where lowercase $w(t)$, called a *noise* within this document, often denotes the formal derivative dW/dt of a Stratonovich Wiener process $W(t,\omega)$. Alternatively, $w(t)$ represents an arbitrary deterministic time-dependent forcing, or some control, or some 'coloured' random process, or some other extrinsic input to the system. Parameter σ controls the strength of the so-called noise.

Use slow variable `x(1)` to denote $x(t)$, stable variable `y(1)` to denote $y(t)$, there is no unstable variable in this S/ODE, and use `w(1)` to denote the (Stratonovich) noise w. Hence this system is analysed for 'small' (x,y,σ) by executing the following defined procedure `examplenormform()`:

```
2 procedure examplenormform(theman);
3     stonormalform(
4     {-x1*y1},
5     {-y1+x1^2-2*y1^2+w(1)},
6     {},
7     3, theman )$
```

The procedure `stonormalform` automatically multiplies the noise factors by a parameter `sigma` so there is no need include the parameter σ in the specification of the problem—it will be done for you.

Further, the procedure uses the parameter `small`, often denoted by ε, to control truncation in nonlinearity. The fourth parameter in the above specifies to construct the normal form to errors $\mathcal{O}\big(\varepsilon^3\big)$.

Consequently, the procedure embeds the given system as the $\varepsilon=1$ version of the following system that it actually analyses:

$$\dot{x}_1 = -\varepsilon x_1 y_1 \quad \text{and} \quad \dot{y}_1 = \sigma w_1 + \varepsilon\big(x_1^2 - 2y_1^2\big) - y_1$$

using analysis and theory based upon the subspace of equilibria $y_1=\varepsilon=\sigma=0$. The constructed coordinate transform and corresponding S/ODEs are the following.

The stochastic coordinate transform (to one order lower in both ε and σ than actually constructed)

$$x_1 = \sigma\varepsilon \mathrm{e}^{-1t} \star w_1 \, X_1 + \varepsilon X_1 Y_1 + X_1 + O\big(\varepsilon^2, \sigma^2\big)$$
$$y_1 = 4\sigma\varepsilon \mathrm{e}^{-1t} \star w_1 \, Y_1 + \sigma \mathrm{e}^{-1t} \star w_1 \, + \varepsilon\big(X_1^2 + 2Y_1^2\big) + Y_1 + O\big(\varepsilon^2, \sigma^2\big)$$

Result normal form SDEs

$$\dot{X}_1 = 2\sigma^2\varepsilon^2 \mathrm{e}^{-1t} \star w_1 \, w_1 X_1 - \sigma\varepsilon w_1 X_1 - \varepsilon^2 X_1^3 + O\big(\varepsilon^3, \sigma^3\big)$$
$$\dot{Y}_1 = 8\sigma^2\varepsilon^2 \mathrm{e}^{-1t} \star w_1 \, w_1 Y_1 - 4\sigma\varepsilon w_1 Y_1 - 2\varepsilon^2 X_1^2 Y_1 - Y_1 + O\big(\varepsilon^3, \sigma^3\big)$$

- Since this construction is based upon the subspace of equilibria $y_1 = \varepsilon = \sigma = 0$, these constructed expressions are asymptotic as $(Y_1, \varepsilon, \sigma) \to 0$. However, evaluation at $\varepsilon = 1$ typically means that the expressions should be reinterpreted as asymptotic as $(Y_1, X_1, \sigma) \to 0$.

  Alternatively, since ε multiplies only quadratic terms, then one could view it as counting the order of nonlinearity in the new variables (X_1, Y_1). For example, the errors $\mathcal{O}\big(\varepsilon^3\big)$ are equivalent to errors $\mathcal{O}\big(|(X_1, Y_1, \sigma)|^4\big)$. Parameter σ arises in this error because the term σw_1 drives effects of size σ in y_1.

- The $\dot{Y}_1$ S/ODE shows that $Y_1 = 0$ is exactly invariant, and since dominantly is $\dot{Y}_1 \approx -Y_1$, then $Y_1 = 0$ is almost always exponentially quickly attractive (emergent) in some domain about the origin.

- The $\dot{X}_1$ S/ODE is independent of Y_1 and here indicates an algebraic attraction to zero, albeit affected by a multiplicative noise, and moderated by some irreducible noise-noise interactions. The independence of Y_1 implies that the $\dot{X}_1$ S/ODE precisely predicts the system dynamics from its initial value over all time.

- These deductions are transformed into the original xy-space by the constructed time-dependent coordinate transformation.

- Backwards theory (Roberts 2019) would assert that the X_1Y_1-system and the coordinate transform, put together, defines a system in the original xy-space, a system that is close to the original specified S/ODEs.

2 Header of the procedure

Need a couple of things established before defining the procedure: the `rlfi` package for a nicer version of the output via LaTeX; and operator names for the variables of the S/ODEs.

```
load_package rlfi;
operator x;
operator y;
operator z;
operator w;
```

Cater for rational function S/ODEs by allowing time dependence in these variables at specification. For rational function S/ODEs, users must multiply each S/ODE by a common denominator, and put on the right-hand side the nonlinear terms involving the time derivative.

```
depend x,t;
depend y,t;
depend z,t;
```

Clear the identifiers used to specify the invariant manifold/normal form.

```
clear cman,sman,uman,csman,cuman,csuman;
```

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

```
operator stonormalform;
for all dxdt, dydt, dzdt, toosmall let
    stonormalform(dxdt, dydt, dzdt, toosmall)
    = stonormalform(dxdt, dydt, dzdt, toosmall, csuman);
for all dxdt, dydt, dzdt, toosmall, theman let
    stonormalform(dxdt, dydt, dzdt, toosmall, theman)
    = begin
```

2.1 Preamble to the procedure

Operators and arrays are always global, but we can make variables and matrices local, except for matrices that need to be declared `matrix`. So, move to implement all arrays and operators to have underscores, and almost all scalars and most matrices to be declared local here (for some reason `x/y/zrhs` must be global).

```
scalar maxiter, trace, nx, ny, nz, offdiag, jac, ok, res,
res0, res1, lengthresx, lengthresy, lengthresz, cmanf,
smanf, umanf, textask;
```

Determine the text of the task.

```
textask:=if theman=cman then "centre manifold" else
    if theman=sman then "stable manifold" else
    if theman=uman then "unstable manifold" else
    if theman=csman then "centre-stable manifold" else
    if theman=cuman then "centre-unstable manifold" else
    if theman=csuman then "normal form" else
    rederr "cannot recognise the manifold to construct"$
```

Set corresponding flags to be 1 or 0 depending upon whether that component of the dynamics is to be in the manifold.

```
procedure inlist(a,b)$ if member(a,b) then 1 else 0$
cmanf := inlist(theman,{cman,csman,cuman,csuman})$
smanf := inlist(theman,{sman,csman,csuman})$
umanf := inlist(theman,{uman,cuman,csuman})$
```

Write an intro message.

```
write "Construct a stochastic ",textask
     ," (version 30 May 2023)"$
```

Parameter `maxiter` is the maximum number of allowed iterations; this default may be changed.

```
maxiter:=32$
```

In the printed expressions, by default, factor `small` (ε) and `sigma` (σ).

```
41 %factor small,sigma;
```

The code cannot handle any cubic or higher order in noise amplitude `sigma`.

```
42 let sigma^3=>0;
```

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

```
43 trace:=0$
44 %trace:=1; maxiter:=5;
```

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

```
45 on div; off allfac; on revpri;
46 on rationalize;
```

2.2 Extract and scale slow equations

The number of slow equations is the number of terms in the list in `dxdt`.

```
47 xrhs_:=dxdt$
48 write "no. of slow modes ",nx:=length(xrhs_);
```

Define aliases `xi` map to `x(i)`, except one *must* use the `x(i)`-form *within* any `df(,t)` on the right-hand side.

```
49 for i:=1:nx do set(mkid(x,i),x(i));
```

Multiply all the right-hand sides by `small` so we can control the truncation of the asymptotic construction through discarding high powers of `small`. Users could use `small` in their equations for appropriate effects.

```
50 xrhs_:=for i:=1:nx collect small*part(xrhs_,i)$
```

Adjust the noise terms. Remove the `small` multiplication of noise terms, and instead multiply by `sigma` to empower independent control of the truncation in noise amplitude.

```
xrhs_:=(xrhs_ where w(~j)=>sigma*w(j,1)/small)$
xrhs_:=(xrhs_ where w(~j,1)=>w(j))$
```

Section 4 writes the resulting differential equations for information.

```
if trace then for i:=1:nx do
    write "dx(",i,")/dt = ",1*part(xrhs_,i);
```

2.3 Extract and scale stable fast equations

The number of stable fast equations is the number of terms in the list in `dydt`.

```
yrhs_:=dydt$
write "no. of stable fast modes ",ny:=length(yrhs_);
```

Define aliases `yi` map to `y(i)`, except one *must* use the `y(i)`-form *within* any `df(,t)` on the right-hand side.

```
for i:=1:ny do set(mkid(y,i),y(i));
```

Extract decay rates Extract the linear decay rates of the fast equations into an array. For each expression in the provided set of right-hand sides:

```
clear rats_; array rats_(ny);
for i:=1:ny do begin
```

For the ith right-hand side get the linear dependence upon `y(i)`, then set other dynamic variables to zero to get just the coefficient.

```
  rats_(i):=coeffn(part(yrhs_,i),y(i),1);
  rats_(i):=(rats_(i) where
      {x(~j)=>0,y(~j)=>0,z(~j)=>0,w(~j)=>0});
```

However, the coefficient may depend upon parameters, so if it is not simply a number, but is a sum, then trawl through the sum looking for a simple number to use as the decay rate.

```
  if not numberp(rats_(i)) then
  if part(rats_(i),0)=plus then begin
```

```
65      rr:=0;
66      for j:=1:arglength(rats_(i)) do
67        if numberp(part(rats_(i),j))
68        then rr:=part(rats_(i),j);
69      rats_(i):=rr;
70    end;
```

Change sign to make `rats_` into positive decay rates, rather than negative growth rates.

```
71    rats_(i):=-rats_(i);
```

If all the above has not ended up with a simple number, then exit with an error message.

```
72    if numberp(rats_(i))and rats_(i)>0 then
73    else begin
74      write "***** Error *****
75      Linear coeffs of y-decay must be negative numbers";
76      return;
77    end;
```

End the loop over all right-hand sides.

```
78  end;
79  if trace then write "End loop over all dydt";
```

Flag later warning if the linear part not diagonal.

```
80  offdiag:=0$
81  for i:=1:ny do for j:=1:ny do if i neq j then begin
82    jac:=coeffn(part(yrhs_,i),y(j),1);
83    if (jac where {x(~k)=>0,y(~k)=>0,z(~k)=>0,w(~k)=>0}) neq 0
84    then offdiag:=1$
85  end;
86  if trace then write offdiag:=offdiag;
```

Multiply all the 'nonlinear' terms right-hand sides by `small` so we control the truncation of the asymptotic construction through discarding high pow-

ers of small. Leave the identified linear decay terms intact. Users could use small in their equations for interesting effects.

```
87 yrhs_:=for i:=1:ny collect
88   small*part(yrhs_,i)+(1-small)*(-rats_(i)*y(i))$
```

Remove the small multiplication of noise terms, and instead multiply by sigma to empower independent control of the truncation in noise amplitude.

```
89 yrhs_:=( yrhs_ where w(~j)=>sigma*w(j,1)/small )$
90 yrhs_:=( yrhs_ where w(~j,1)=>w(j) )$
```

Section 4 writes the resulting differential equations for information.

```
91 if trace then for i:=1:ny do
92     write "dy(",i,")/dt = ",1*part(yrhs_,i);
```

2.4 Extract and scale unstable fast equations

The number of unstable fast equations is the number of terms in the list in dzdt.

```
93 zrhs_:=dzdt$
94 write "no. of unstable fast modes ",nz:=length(zrhs_);
```

Define aliases zi map to z(i), except one *must* use the z(i)-form *within* any df(,t) on the right-hand side.

```
95 for i:=1:nz do set(mkid(z,i),z(i));
```

Extract growth rates Extract the linear growth rates of the fast equations into an array. For each expression in the provided set of right-hand sides:

```
96 clear ratu_; array ratu_(nz);
97 for i:=1:nz do begin
```

For the ith right-hand side get the linear dependence upon z(i), then set other dynamic variables to zero to get just the coefficient.

```
98    ratu_(i):=coeffn(part(zrhs_,i),z(i),1);
99    ratu_(i):=(ratu_(i) where
100       {x(~j)=>0,y(~j)=>0,z(~j)=>0,w(~j)=>0});
```

However, the coefficient may depend upon parameters, so if it is not simply a number, but is a sum, then trawl through the sum looking for a simple number to use as the growth rate.

```
101   if not numberp(ratu_(i)) then
102   if part(ratu_(i),0)=plus then begin
103     rr:=0;
104     for j:=1:arglength(ratu_(i)) do
105       if numberp(part(ratu_(i),j))
106       then rr:=part(ratu_(i),j);
107     ratu_(i):=rr;
108   end;
```

If all the above has not ended up with a simple number, then exit with an error message.

```
109   if numberp(ratu_(i))and ratu_(i)>0 then
110   else begin
111     write "***** Error *****
112     Linear coeffs of z-growth must be positive numbers";
113     return;
114   end;
```

End the loop over all z-right-hand sides.

```
115 end;
116 if trace then write "End loop over all dzdt";
```

Flag warning if the linear part not diagonal.

```
117 for i:=1:nz do for j:=1:nz do if i neq j then begin
118   jac:=coeffn(part(zrhs_,i),z(j),1);
119   if (jac where {x(~k)=>0,y(~k)=>0,z(~k)=>0,w(~k)=>0}) neq 0
120   then offdiag:=1$
121 end;
```

```
122 if trace then write offdiag:=offdiag;
```

Multiply all the 'nonlinear' terms right-hand sides by `small` so we control the truncation of the asymptotic construction through discarding high powers of `small`. Leave the identified linear growth terms intact. Users could use `small` in their equations for interesting effects.

```
123 zrhs_:=for i:=1:nz collect
124   small*part(zrhs_,i)+(1-small)*(+ratu_(i)*z(i))$
```

Remove the `small` multiplication of noise terms, and instead multiply by `sigma` to empower me to independently control the truncation in noise amplitude.

```
125 zrhs_:=( zrhs_ where w(~j)=>sigma*w(j,1)/small )$
126 zrhs_:=( zrhs_ where w(~j,1)=>w(j) )$
```

Section 4 writes the resulting differential equations for information.

```
127 if trace then for i:=1:nz do
128     write "dz(",i,")/dt = ",1*part(zrhs_,i);
```

3 Setup LaTeX output using rlfi

Use inline math environment so that long lines, the norm, get line breaks. The command `\raggedright` in the LaTeX preamble appears the best option for the line breaking, but `\sloppy` would also work reasonably.

```
129 mathstyle math;
```

Define names for LaTeX formatting Define some names I use, so that rlfi translates them to Greek characters in the LaTeX.

```
130 defid small,name="\eps";%varepsilon;
131 defid alpha,name=alpha;
132 defid beta,name=beta;
133 defid gamma,name=gamma;
134 defid delta,name=delta;
```

```
135 defid epsilon,name=epsilon;
136 defid varepsilon,name=varepsilon;
137 defid zeta,name=zeta;
138 defid eta,name=eta;
139 defid theta,name=theta;
140 defid vartheta,name=vartheta;
141 defid iota,name=iota;
142 defid kappa,name=kappa;
143 defid lambda,name=lambda;
144 defid mu,name=mu;
145 defid nu,name=nu;
146 defid xi,name=xi;
147 defid pi,name=pi;
148 defid varpi,name=varpi;
149 defid rho,name=rho;
150 defid varrho,name=varrho;
151 defid sigma,name=sigma;
152 defid varsigma,name=varsigma;
153 defid tau,name=tau;
154 defid upsilon,name=upsilon;
155 defid phi,name=phi;
156 defid varphi,name=varphi;
157 defid chi,name=chi;
158 defid psi,name=psi;
159 defid omega,name=omega;
160 defid Gamma,name=Gamma;
161 defid Delta,name=Delta;
162 defid Theta,name=Theta;
163 defid Lambda,name=Lambda;
164 defid Xi,name=Xi;
165 defid Pi,name=Pi;
166 defid Sigma,name=Sigma;
167 defid Upsilon,name=Upsilon;
168 defid Phi,name=Phi;
169 defid Psi,name=Psi;
```

```
defid Omega,name=Omega;
```

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

```
defindex w(down);
defindex x(down);
defindex y(down);
defindex z(down);
defid xx,name="X";
defid yy,name="Y";
defid zz,name="Z";
defindex xx(down);
defindex yy(down);
defindex zz(down);
defindex hh(down);
defindex gg(down);
defindex ff(down);
```

First use these for the specified dynamical system, later use them for the normal form equations.

```
defid hh,name="\dot z";
defid gg,name="\dot y";
defid ff,name="\dot x";
```

The 'Ornstein–Uhlenbeck' operator is to translate into a LaTeX command, see the preamble, that typesets the convolution in a reasonable manner. The definition of the LaTeX command is a bit dodgy as convolutions of convolutions are not printed in the correct order; however, convolutions commute so it does not matter.

```
defid ou,name="\ou";
defindex ou(arg,arg,arg);
```

Write the LaTeX dynamical system Because of the way rfli works, to get good quality output to the LaTeX document, I need to write the algebraic

expressions to a file, then read them back in again. While being read back in, I send the output to the LaTeX file. In this convoluted way I avoid extraneous output lines polluting the LaTeX.

Temporarily use these arrays for the right-hand sides of the dynamical system.

```
clear ff,gg,hh;
array ff(nx),gg(ny),hh(nz);
```

Write expressions to the file `scratchfile.red` for later reading. Prepend the expressions with an instruction to write a heading, and surround the heading with anti-math mode to cancel the math environment that rlfi puts in.

```
out "scratchfile.red"$
write "off echo;"$  % do not understand why needed in 2021??
write "write ""\)
\paragraph{Specified dynamical system}
\(""$";
for i:=1:nx do write "ff(",i,"):=1*part(xrhs_,",i,");";
for i:=1:ny do write "gg(",i,"):=1*part(yrhs_,",i,");";
for i:=1:nz do write "hh(",i,"):=1*part(zrhs_,",i,");";
write "end;";
shut "scratchfile.red";
```

Then switch on LaTeX output before writing to file as this LaTeX file is to be input from the main LaTeX file and hence does not need a header. The header here gets sent to the 'terminal' instead. Then write to `stoNFreportSys.tex` the expressions we stored in `scratchfile.red` as nice LaTeX.

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

4 Delayed write of text info

Because it is messy to interleave LaTeX and plain output, I delay writing anything much in plain text until here.

Write the delayed warning message about off-diagonal terms.

```
207 if offdiag then write "
208 ***** Warning ****
209 Off diagonal linear terms in y- or z- equations
210 assumed small.  Answers are rubbish if not
211 asymptotically appropriate. "$
```

Write the plain text versions of the dynamical system.

```
212 write "no. of slow modes ",nx:=length(xrhs_);
213 for i:=1:nx do write "dx(",i,")/dt = ",1*part(xrhs_,i);
214 write "no. of stable fast modes ",ny:=length(yrhs_);
215 for i:=1:ny do write "dy(",i,")/dt = ",1*part(yrhs_,i);
216 write "no. of unstable fast modes ",nz:=length(zrhs_);
217 for i:=1:nz do write "dz(",i,")/dt = ",1*part(zrhs_,i);
```

5 Represent the noise

The 'noises' `w` depend upon time. But we find it useful to discriminate upon the notionally fast time fluctuations of a noise process, and the notionally ordinary time variations of the dynamic variables x_i, y_i, and z_i. Thus introduce a notionally fast time variable `tt`, which depends upon the ordinary time `t`. Equivalently, view `tt`, a sort of 'partial t', as representing variations in time independent of those in the variables x_i, y_i, and z_i.

```
218 depend w,tt;
219 depend tt,t,ttyz;
```

In the construction, convolutions of the noise arise, both backwards over history and also forwards in time to anticipate the noise (Roberts 2008, 2019).

For any non-zero parameter μ, define the 'Ornstein–Uhlenbeck' convolution

$$e^{\mu t}\star\phi = \begin{cases} \int_{-\infty}^{t} \exp[\mu(t-\tau)]\phi(\tau)\,d\tau\,, & \mu<0\,,\\ \int_{t}^{+\infty} \exp[\mu(t-\tau)]\phi(\tau)\,d\tau\,, & \mu>0\,, \end{cases} \tag{1}$$

so that the convolution is always with a bounded exponential. Five useful properties of this convolution are

$$e^{\mu t}\star 1 = \frac{1}{|\mu|}\,, \tag{2}$$

$$\frac{d}{dt}e^{\mu t}\star\phi = -\operatorname{sgn}\mu\,\phi + \mu e^{\mu t}\star\phi\,, \tag{3}$$

$$E[e^{\mu t}\star\phi] = e^{\mu t}\star E[\phi]\,, \tag{4}$$

$$E[(e^{\mu t}\star\phi)^2] = \frac{1}{2|\mu|}\,, \tag{5}$$

$$e^{\mu t}\star e^{\nu t}\star = \begin{cases} \frac{1}{|\mu-\nu|}\left[e^{\mu t}\star + e^{\nu t}\star\right]\,, & \mu\nu<0\,,\\ \frac{-\operatorname{sgn}\mu}{\mu-\nu}\left[e^{\mu t}\star - e^{\nu t}\star\right]\,, & \mu\nu>0 \;\&\; \mu\neq\nu\,. \end{cases} \tag{6}$$

Also remember that although with $\mu<0$ the convolution $e^{\mu t}\star$ integrates over the past, with $\mu>0$ the convolution $e^{\mu t}\star$ integrates into the future—both over a time scale of order $1/|\mu|$.

The operator `ou(f,tt,mu)` represents the convolution $e^{\mu t}\star f$ as defined by (1): called `ou` because it is an Ornstein–Uhlenbeck process when f is a stochastic white noise. The operator `ou` is 'linear' over fast time `tt` as the convolution only arises from solving PDEs in the operator $\partial_t - \mu$. Code its derivative (3) and its action upon autonomous terms (2):

```
clear ou; operator ou; linear ou;
let { df(ou(~f,tt,~mu),t)=>-sign(mu)*f+mu*ou(f,tt,mu)
    , ou(1,tt,~mu)=>1/abs(mu)
```

Also code the transform (6) that successive convolutions at different rates may be transformed into several single convolutions.

```
    , ou(ou(~r,tt,~nu),tt,~mu) =>
      (ou(r,tt,mu)+ou(r,tt,nu))/abs(mu-nu) when (mu*nu<0)
```

```
225     , ou(ou(~r,tt,~nu),tt,~mu) =>
226       -sign(mu)*(ou(r,tt,mu)-ou(r,tt,nu))/(mu-nu)
227       when (mu*nu>0)and(mu neq nu)
228     };
```

The above properties are *critical*: they must be correct for the results to be correct. Currently, they are only coded for *real* rates μ, ν.

Second, identify the resonant parts, some of which must go into the evolution `gg(i)`, and some into the transform. It depends upon the exponent of `yz` compared to the decay rate of this mode, here `r`.

```
229 clear reso_; operator reso_; linear reso_;
230 let { reso_(~a,yz,~r)=>1 when df(a,yz)*yz=r*a
231     , reso_(~a,yz,~r)=>0 when df(a,yz)*yz neq r*a
232     };
```

Lastly, the remaining terms get convolved at the appropriate rate to solve their respective homological equation by the operator `zres_`.

```
233 depend yz,ttyz;
234 clear zres_; operator zres_; linear zres_;
235 let zres_(~a,ttyz,~r)=>ou(sign(df(a,yz)*yz/a-r)
236     *sub(yz=1,a),tt,df(a,yz)*yz/a-r);
```

6 Operators to solve noisy homological equation

When solving homological equations of the form $F + \xi_t = \text{Res}$ (the resonant case $\mu = 0$), we separate the terms in the right-hand side Res into those that are integrable in fast time, and hence modify the coordinate transform by changing ξ, and those that are not, and hence must remain in the evolution by changing F. the operator `zint_` extracts those parts of a term that we know are integrable; the operator `znon_` extracts those parts which are not knowably bounded integrable. With more research, more types of terms may be found to be integrable; hence what is extracted by `zint_` and what is left by `zint_` may change with more research, or in different scenarios. These transforms are not critical: changing the transforms may change intermediate

computations, but as long as the iteration converges, the computer algebra results will be algebraically correct.

```
clear zint_; operator zint_; linear zint_;
clear znon_; operator znon_; linear znon_;
```

First, avoid obvious secularity.

```
let { zint_(w(~i),tt)=>0, znon_(w(~i),tt)=>w(i)
    , zint_(1,tt)=>0, znon_(1,tt)=>1
    , zint_(w(~i)*~r,tt)=>0, znon_(w(~i)*~r,tt)=>w(i)*r
```

Second, by (3) a convolution may be split into an integrable part, and a part in its argument which in turn may be integrable or not.

```
    , zint_(ou(~r,tt,~mu),tt)
      =>ou(r,tt,mu)/mu+zint_(r,tt)/abs(mu)
    , znon_(ou(~r,tt,~mu),tt)=>znon_(r,tt)/abs(mu)
```

Third, squares of convolutions may be integrated by parts to an integrable term and a part that may have integrable or non-integrable parts.

```
    , zint_(ou(~r,tt,~mu)^2,tt)=>ou(~r,tt,~mu)^2/(2*mu)
                                  +zint_(r*ou(r,tt,mu),tt)/abs(mu)
    , znon_(ou(~r,tt,~mu)^2,tt)=>znon_(r*ou(r,tt,mu),tt)/abs(mu)
```

Fourth, different products of convolutions may be similarly separated using integration by parts.

```
    , zint_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)
      =>ou(r,tt,mu)*ou(s,tt,nu)/(mu+nu)
      +zint_(sign(mu)*r*ou(s,tt,nu)+sign(nu)*s*ou(r,tt,mu),tt)
      /(mu+nu) when mu+nu neq 0
    , znon_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
      +znon_(sign(mu)*r*ou(s,tt,nu)+sign(nu)*s*ou(r,tt,mu),tt)
      /(mu+nu) when mu+nu neq 0
```

However, a zero divisor arises when $\mu + \nu = 0$ in the above. Here code rules to cater for such terms by increasing the depth of convolutions over past history.

```
255     , zint_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
256       ou(ou(r,tt,-nu),tt,-nu)*ou(s,tt,nu)
257       +zint_(ou(ou(r,tt,-nu),tt,-nu)*s,tt)
258       when (mu+nu=0)and(nu>0)
259     , znon_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
260       znon_(ou(ou(r,tt,-nu),tt,-nu)*s,tt)
261       when (mu+nu=0)and(nu>0)
```

The above handles quadratic products of convolutions. Presumably, if we seek cubic noise effects then we may need cubic products of convolutions. However, I do not proceed so far and hence terminate the separation rules.

```
262 };
```

7 Initialise approximate transform

Truncate asymptotic approximation of the coordinate transform depending upon the parameter `toosmall`. Use the 'instant evaluation' property of a loop index to define the truncation so that Reduce omits small terms on the fly.

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

Variables `x`, `y` and `z` are operators in the specification of the equations. We now want them to map to the approximation to the coordinate transform, so point them to arrays storing the normal form expressions. Need to clear the mapping to the array before exiting.

```
264 clear x_,y_,z_;
265 let { x(~j)=>x_(j), y(~j)=>y_(j), z(~j)=>z_(j) };
266 array x_(nx),y_(ny),z_(nz);
```

Express the normal form in terms of new evolving variables X_i, Y_i, and Z_i, denoted by operators `xx(i)`, `yy(i)` and `zz(i)`, which are nonlinear modifications to x_i, y_i and z_i. The expressions for the normal form S/ODEs are stored in `ff`, `gg` and `hh`.

```
267 clear xx,yy,zz;
```

```
operator xx; operator yy; operator zz;
depend xx,t; depend yy,t; depend zz,t;
let { df(xx(~i),t)=>ff(i)*cmanf
    , df(yy(~i),t)=>gg(i)*smanf
    , df(zz(~i),t)=>hh(i)*umanf };
```

The first linear approximation is then $x_i \approx X_i$, $y_i \approx Y_i$ and $z_i = Z_i$, such that $\dot{X}_i \approx 0$, in `ff(i)`, $\dot{Y}_i \approx -r_i Y_i$, in `gg(i)`, and $\dot{Z}_i \approx +r_i Z_i$, in `hh(i)`. Depending upon the specified manifold, these are multiplied by the appropriate 0-1 flags.

```
for i:=1:nx do x_(i):=xx(i)*cmanf;
for i:=1:ny do y_(i):=yy(i)*smanf;
for i:=1:nz do z_(i):=zz(i)*umanf;
for i:=1:nx do ff(i):=0;
for i:=1:ny do gg(i):=-rats_(i)*yy(i)*smanf;
for i:=1:nz do hh(i):=+ratu_(i)*zz(i)*umanf;
```

Consider updating the Y_i evolution `gg(i)` and the y_i transform. The residual is of the form of a sum of terms $\prod_j Y_j^{q_j} Z_k^{r_k} \in \text{Res}$. So updates involve dividing by, or convolving with, $\beta_i - \sum_j \beta_j q_j + \sum_k \gamma_k r_k$. First, form the substitutions needed to introduce `yz` to count the number of variables Y_i and Z_i in any given term, weighted according to their rate coefficient in the homological equation.

```
y4y:=for i:=1:ny collect yy(i)=yy(i)*yz^rats_(i)$
z4z:=for i:=1:nz collect zz(i)=zz(i)/yz^ratu_(i)$
y4y:=append(y4y,z4z)$
```

8 Iterative updates

We iterate to a solution of the governing S/ODEs to residuals of some order of error. The number of iterations are limited by a maximum.

```
for iter:=1:maxiter do begin
  ok:=1;
  if trace then write "
  ITERATION = ",iter,"
```

```
286   -------------";
```

8.1 Fast stable modes

Compute the residual of each of the y_i S/ODEs, updating `ok` to track whether all S/ODEs are satisfied. Keep track of the lengths of the residuals to indicate progress in the iteration.

```
287   lengthresy:={};
288   for i:=1:ny do begin
289     res:=-df(y(i),t)+part(yrhs_,i);
290     ok:=if res=0 then ok else 0;
291     lengthresy:=append(lengthresy,{length(res)});
292     if trace then write "resy",i," = ",res;
```

Within the loop: first insert the weighted count of Y and Z variables; then split the residual into two parts of possibly resonant, `res0` and the rest, `res1`; then allocate to the evolution or the transform.

```
293     res:=sub(y4y,res);
294     res0:=reso_(res,yz,+rats_(i));
295     res1:=res-res0*yz^rats_(i);
296     if smanf then gg(i):=gg(i)+znon_(res0,tt)
297     else if znon_(res0,tt) neq 0
298         then rederr("oops y-res: contact me");
299     if trace then write "dY",i,"/dt = ",gg(i);
300     y_(i):=y_(i) +zint_(res0,tt) -zres_(res1,ttyz,rats_(i));
301     if trace then write "y",i," = ",y(i);
302   end;
303   if ny>0 then write lengthresy:=lengthresy;
```

8.2 Fast unstable modes

Compute the residual of each of the z_i S/ODEs, updating `ok` to track whether all S/ODEs are satisfied. Keep track of the lengths of the residuals to indicate progress in the iteration.

```
304   lengthresz:={};
305   for i:=1:nz do begin
306     res:=-df(z(i),t)+part(zrhs_,i);
307     ok:=if res=0 then ok else 0;
308     lengthresz:=append(lengthresz,{length(res)});
309     if trace then write "resz",i," = ",res;
```

Update the Z_i evolution `hh(i)` and the z_i transform. Within the loop: first insert the weighted count of Y and Z variables; then split the residual into two parts of possibly resonant, `res0`, and the rest, `res1`; then allocate to the evolution or the transform.

```
310     res:=sub(y4y,res);
311     res0:=reso_(res,yz,-ratu_(i));
312     res1:=res-res0/yz^ratu_(i);
313     if umanf then hh(i):=hh(i)+znon_(res0,tt)
314     else if znon_(res0,tt) neq 0
315         then rederr("oops z-res: contact me");
316     z_(i):=z_(i) +zint_(res0,tt) -zres_(res1,ttyz,-ratu_(i));
317   end;
318   if nz>0 then write lengthresz:=lengthresz;
```

8.3 Slow modes

Compute the residual of each of the x S/ODEs, updating `ok` to track whether all S/ODEs are satisfied. Keep track of the lengths of the residuals to indicate progress in the iteration.

```
319   lengthresx:={};
320   for i:=1:nx do begin
321     res:=-df(x(i),t) +part(xrhs_,i);
322     ok:=if res=0 then ok else 0;
323     lengthresx:=append(lengthresx,{length(res)});
324     if trace then write "resx",i," = ",res;
```

Update the X_i evolution `ff(i)` and the x_i transform. Use the same process as for the fast variables; the difference is that here the mode rate is zero.

```
325     res:=sub(y4y,res);
326     res0:=reso_(res,yz,0);
327     res1:=res-res0;
328     if cmanf then ff(i):=ff(i)+znon_(res0,tt)
329     else if znon_(res0,tt) neq 0 then begin
330         write "**** Requested manifold may not exist due to";
331         write "slow x-residual component: res0=",res0;
332         write "**** So halving the max number of iterations";
333         maxiter:=maxiter/2;
334     end;
335     if trace then write "dX",i,"/dt = ",ff(i);
336     x_(i):=x_(i) +zint_(res0,tt) -zres_(res1,ttyz,0);
337     if trace then write "x",i," = ",x(i);
338   end;
339   if nx>0 then write lengthresx:=lengthresx;
```

Terminate the iteration loop once all residuals are zero, or the maximum number of iterations has been done.

```
340   showtime;
341   if ok then write "Number of iterations ",
342     iter:=1000000+iter;
343 end;
```

9 Output results

Only proceed to print if terminated successfully.

```
344 if ok
345   then write "SUCCESS: converged to an expansion"
346   else <<write "FAILED TO CONVERGE; I EXIT";
347     return; >>;
```

9.1 Plain text version

Print the resultant coordinate transform: but only print to one lower power in `small` and `sigma` in order to keep output relatively small.

```
write "Stochastic/non-autonomous ",textask," coordinates"$
write "(to one order lower in both small and sigma)"$
for i:=1:nz do if trace then write z_(i):=z_(i)
    else begin z_(i):=sigma*small*z_(i);
         write z_(i):=z_(i)/small/sigma; end;
for i:=1:ny do if trace then write y_(i):=y_(i)
    else begin y_(i):=sigma*small*y_(i);
         write y_(i):=y_(i)/small/sigma; end;
for i:=1:nx do if trace then write x_(i):=x_(i)
    else begin x_(i):=sigma*small*x_(i);
         write x_(i):=x_(i)/small/sigma; end;
```

Lastly print the normal form S/ODEs: first the fast, second the slow.

```
write "The ",textask," S/ODEs";
if umanf then for i:=1:nz do write "dzz(",i,")/dt = ",hh(i);
if smanf then for i:=1:ny do write "dyy(",i,")/dt = ",gg(i);
if cmanf then for i:=1:nx do write "dxx(",i,")/dt = ",ff(i);
```

9.2 LaTeX version

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. Further, to avoid `sigma^3` being replaced by zero, introduce `sigma_` that maps to σ.

```
clear order_;  operator order_;
defid order_,name="O";
defindex order_(arg,arg);
defid sigma_,name="\sigma";
```

As before, we have to write expressions to file for later reading so they get printed without extraneous dross in the LaTeX source. First open up the

temporary file `scratchfile.red` again.

```
out "scratchfile.red"$
write "off echo;"$ % do not understand why needed in 2021??
```

Write the stochastic coordinate transform to file, with a heading, and with an anti-math environment to cancel the auto-math of rlfi. For some reason we have to keep these writes short as otherwise it generates spurious fatal blank lines in the LaTeX.

```
write "write ""\)
\paragraph{Time dependent ",textask," coordinates}
\("";";
for i:=1:nz do write "z_(",i,"):=z_(",i,
    ") +order_(varepsilon^",toosmall-1,",sigma_^2);";
for i:=1:ny do write "y_(",i,"):=y_(",i,
    ") +order_(varepsilon^",toosmall-1,",sigma_^2);";
for i:=1:nx do write "x_(",i,"):=x_(",i,
    ") +order_(varepsilon^",toosmall-1,",sigma_^2);";
```

Write the resultant stochastic normal form to file, with a heading, and with an anti-math environment to cancel the auto-math of rlfi.

```
write "write ""\)
\paragraph{Result ",textask," DEs}
\("";";
if umanf then for i:=1:nz do write "hh(",i,"):=hh(",i,
    ") +order_(varepsilon^",toosmall,",sigma_^3);";
if smanf then for i:=1:ny do write "gg(",i,"):=gg(",i,
    ") +order_(varepsilon^",toosmall,",sigma_^3);";
if cmanf then for i:=1:nx do write "ff(",i,"):=ff(",i,
    ") +order_(varepsilon^",toosmall,",sigma_^3);";
write "end;";
```

Shut the temporary output file.

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

Get expressions from file and write the main LaTeX file. But first redefine

how these names get printed, namely as the normal form time derivatives.

```
defid x_,name="x"; defindex x_(down);
defid y_,name="y"; defindex y_(down);
defid z_,name="z"; defindex z_(down);
defid hh,name="\dot Z";
defid gg,name="\dot Y";
defid ff,name="\dot X";
```

Penultimately, write the header information file that is to be included in the report via an `\input` in the modified `on latex`.

```
out "stoNFreportHdr.tex"$
write "\title{A ",textask," of your dynamical system}"$
write "\author{A. J. Roberts, University of Adelaide\\
\texttt{http://orcid.org/0000-0001-8930-1552}}
\date{\now, \today}
\def\ou\big(#1,#2,#3\big)%
    {{{\rm e}^{\if#31\else#3\fi t}\star}#1\,}
\def\eps{\varepsilon}
\maketitle
Generally, the lowest order, most important,
terms are near the end of each expression.
\input{stoNFreportSys}
"$
shut "stoNFreportHdr.tex"$
```

Finally write to the main LaTeX file so switch on latex after starting to write to the file. Then write expressions in `scratchfile.red` to `stoNFreport.tex` as nice LaTeX. Switch off latex, to get the end of the document, and finish writing.

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

10 Fin

That's all folks, so end the procedure, after clearing the mapping from operators to the stored expressions.

```
clear x(~j),y(~j),z(~j);
if theman=csuman
then return Finished_constructing_normal_form_of_system
else return Finished_invariant_manifold_of_system$
end$
```

11 Override some rlfi procedures

Now setup the rlfi package to write a LaTeX version of the output. It is all a bit tricky and underhand. We override some stuff from `rlfi.red`.[3]

First, change `name` to get Big delimiters, not left-right delimiters, so LaTeX can break lines.

```
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 mainly a copy from `rlfi.red` with the appropriate if-statement deleted.

```
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!*;
```

[3] Find it in reduce-algebra/trunk/packages/misc/rlfi.red

```
430   nochar!*:=append(nochar!*,nochar1!*);
431   nochar1!*:=nil >>$
```

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

```
432 symbolic procedure prinlabegin;
433 <<if !*verbatim then
434       <<terpri();
435         prin2t "\end{verbatim}">>;
436   linel!*:=linelength nil . laline!*;
437   if ofl!* then linelength(laline!* + 2)
438     else laline!*:=car linel!* - 2;
439   prin2 "\(" >>$
```

Override the procedure that outputs the LaTeX preamble upon the command `on latex`.

```
440 symbolic procedure latexon;
441 <<!*!*a2sfn:='texaeval;
442   !*raise:=nil;
443   prin2t "\documentclass[11pt,a5paper]{article}";
444   prin2t "\usepackage[a5paper,margin=13mm]{geometry}";
445   prin2t "\usepackage{parskip,time} \raggedright";
446   prin2t "\begin{document}\input{stoNFreportHdr}";
447   if !*verbatim then
448       <<prin2t "\begin{verbatim}";
449         prin2t "REDUCE Input:">>;
450   put('tex,'rtypefn,'(lambda(x) 'tex)) >>$
```

End the file when input to Reduce

```
451 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

Chao, X. & Roberts, A. J. (1996), 'On the low-dimensional modelling of Stratonovich stochastic differential equations', *Physica A* **225**, 62–80.

Cox, S. M. & Roberts, A. J. (1991), 'Centre manifolds of forced dynamical systems', *J. Austral. Math. Soc. B* **32**, 401–436.

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

Haragus, M. & Iooss, G. (2011), *Local Bifurcations, Center Manifolds, and Normal Forms in Infinite-Dimensional Dynamical Systems*, Springer.

Knobloch, H.-W. & Aulbach, B. (1982), The role of center manifolds in ordinary differential equations, *in* M. Gregus, ed., 'Proceedings of the fifth Czechoslovak conference on differential equations and their applications', BSB B.G. Teubner Verlagsgesellschaft, Leipzig, pp. 179–189.
http://dml.cz/bitstream/handle/10338.dmlcz/702285/Equadiff_05-1982-1_40.pdf

Roberts, A. J. (1997), 'Low-dimensional modelling of dynamics via computer algebra', *Computer Phys. Comm.* **100**, 215–230.

Roberts, A. J. (2008), 'Normal form transforms separate slow and fast modes in stochastic dynamical systems', *Physica A* **387**, 12–38.

Roberts, A. J. (2009–2023), Normal form of stochastic or deterministic multiscale differential equations, Technical report, http://www.maths.adelaide.edu.au/anthony.roberts/sdenf.php.

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].