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

A. J. Roberts*

19 Nov 2008; last modified November 30, 2025

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 2009–2025). 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 (aliases exist for `w0` to `w9`); 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

[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.

sides of the S/ODEs for the fast stable variables $y_j(t)$ (the list may be 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+w0},
6     {},
7     5, 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.

```
 8 load_package rlfi;
 9 operator x;
10 operator y;
11 operator z;
12 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.

```
13 depend x,t;
14 depend y,t;
15 depend z,t;
```

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

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

Let's make the algebra case sensitive: for *psl* we set `raise`; whereas for *csl* we set `lower`. Remember (currently, 2025) `off latex` automatically turns `raise` back on (but not `lower`).

```
17 off raise;  off lower;
```

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

```
18 operator stonormalform;
19 for all dxdt, dydt, dzdt, toosmall let
20     stonormalform(dxdt, dydt, dzdt, toosmall)
21     = stonormalform(dxdt, dydt, dzdt, toosmall, csuman);
22 for all dxdt, dydt, dzdt, toosmall, theman let
23     stonormalform(dxdt, dydt, dzdt, toosmall, theman)
```

```
24      = 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).

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

Determine the text of the task.

```
28 textask:=if theman=cman then "centre manifold" else
29      if theman=sman then "stable manifold" else
30      if theman=uman then "unstable manifold" else
31      if theman=csman then "centre-stable manifold" else
32      if theman=cuman then "centre-unstable manifold" else
33      if theman=csuman then "normal form" else
34      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.

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

Write an intro message.

```
39 write "Construct a stochastic ",textask
40       ," (version 30 Nov 2025)"$
```

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

```
41 maxiter:=32$
```

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

```
42 %factor small,sigma;
```

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

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

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

```
44 trace:=0$
45 %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.

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

2.2 Extract and scale slow equations

Define aliases `wi` to map to `w(i)`.

```
48 for i:=0:9 do if mkid(w,i) neq w(i)
49     then set(mkid(w,i),w(i));
```

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

```
50 xrhs_:=dxdt$
51 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.

```
52 for i:=1:nx do if mkid(x,i) neq x(i)
53     then 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.

```
54 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.

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

Section 4 writes the resulting differential equations for information.

```
57 if trace then for i:=1:nx do
58     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`.

```
59 yrhs_:=dydt$
60 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.

```
61 for i:=1:ny do if mkid(y,i) neq y(i)
62     then 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:

```
63 clear rats_; array rats_(ny);
64 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.

```
65   rats_(i):=coeffn(part(yrhs_,i),y(i),1);
66   rats_(i):=(rats_(i) where
67       {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.

```
68   if not numberp(rats_(i)) then
69   if part(rats_(i),0)=plus then begin
70     rr:=0;
71     for j:=1:arglength(rats_(i)) do
72       if numberp(part(rats_(i),j))
73       then rr:=part(rats_(i),j);
74     rats_(i):=rr;
75   end;
```

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

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

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

```
77   if numberp(rats_(i))and rats_(i)>0 then
78   else begin
79     write "***** Error *****
80     Linear coeffs of y-decay must be negative numbers";
81     return;
82   end;
```

End the loop over all right-hand sides.

```
83 end;
84 if trace then write "End loop over all dydt";
```

Flag later warning if the linear part not diagonal.

```
85 offdiag:=0$
```

```
for i:=1:ny do for j:=1:ny do if i neq j then begin
  jac:=coeffn(part(yrhs_,i),y(j),1);
  if (jac where {x(~k)=>0,y(~k)=>0,z(~k)=>0,w(~k)=>0}) neq 0
  then offdiag:=1$
end;
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 decay terms intact. Users could use `small` in their equations for interesting effects.

```
yrhs_:=for i:=1:ny collect
  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.

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

Section 4 writes the resulting differential equations for information.

```
if trace then for i:=1:ny do
    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`.

```
zrhs_:=dzdt$
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.

```
for i:=1:nz do if mkid(z,i) neq z(i)
```

```
101      then 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:

```
102 clear ratu_; array ratu_(nz);
103 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.

```
104   ratu_(i):=coeffn(part(zrhs_,i),z(i),1);
105   ratu_(i):=(ratu_(i) where
106       {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.

```
107   if not numberp(ratu_(i)) then
108   if part(ratu_(i),0)=plus then begin
109     rr:=0;
110     for j:=1:arglength(ratu_(i)) do
111       if numberp(part(ratu_(i),j))
112       then rr:=part(ratu_(i),j);
113     ratu_(i):=rr;
114   end;
```

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

```
115   if numberp(ratu_(i))and ratu_(i)>0 then
116   else begin
117     write "***** Error *****
118     Linear coeffs of z-growth must be positive numbers";
119     return;
120   end;
```

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

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

Flag warning if the linear part not diagonal.

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

```
zrhs_:=for i:=1:nz collect
  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.

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

Section 4 writes the resulting differential equations for information.

```
if trace then for i:=1:nz do
    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.

```
135 mathstyle math;
```

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

```
136 defid small,name="\eps";%varepsilon;
137 defid alpha,name=alpha;
138 defid beta,name=beta;
139 defid gamma,name=gamma;
140 defid delta,name=delta;
141 defid epsilon,name=epsilon;
142 defid varepsilon,name=varepsilon;
143 defid zeta,name=zeta;
144 defid eta,name=eta;
145 defid theta,name=theta;
146 defid vartheta,name=vartheta;
147 defid iota,name=iota;
148 defid kappa,name=kappa;
149 defid lambda,name=lambda;
150 defid mu,name=mu;
151 defid nu,name=nu;
152 defid xi,name=xi;
153 defid pi,name=pi;
154 defid varpi,name=varpi;
155 defid rho,name=rho;
156 defid varrho,name=varrho;
157 defid sigma,name=sigma;
158 defid varsigma,name=varsigma;
159 defid tau,name=tau;
160 defid upsilon,name=upsilon;
161 defid phi,name=phi;
162 defid varphi,name=varphi;
163 defid chi,name=chi;
164 defid psi,name=psi;
165 defid omega,name=omega;
```

```
166 defid Gamma,name=Gamma;
167 defid Delta,name=Delta;
168 defid Theta,name=Theta;
169 defid Lambda,name=Lambda;
170 defid Xi,name=Xi;
171 defid Pi,name=Pi;
172 defid Sigma,name=Sigma;
173 defid Upsilon,name=Upsilon;
174 defid Phi,name=Phi;
175 defid Psi,name=Psi;
176 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.

```
177 defindex w(down);
178 defindex x(down);
179 defindex y(down);
180 defindex z(down);
181 %defid xx,name="X";
182 %defid yy,name="Y";
183 %defid zz,name="Z";
184 defindex X(down);
185 defindex Y(down);
186 defindex Z(down);
187 defindex hh(down);
188 defindex gg(down);
189 defindex ff(down);
```

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

```
190 defid hh,name="\dot z";
191 defid gg,name="\dot y";
192 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.

```
193 defid ou,name="\ou";
194 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.

```
195 clear ff,gg,hh;
196 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.

```
197 out "scratchfile.red"$
198 write "off echo;"$  % do not understand why needed in 2021??
199 write "write ""\)
200 \paragraph{Specified dynamical system}
201 \(""$";
202 for i:=1:nx do write "ff(",i,"):=1*part(xrhs_,",i,");";
203 for i:=1:ny do write "gg(",i,"):=1*part(yrhs_,",i,");";
204 for i:=1:nz do write "hh(",i,"):=1*part(zrhs_,",i,");";
205 write "end;";
206 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. Switch `raise` back off.

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

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.

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

Write the plain text versions of the dynamical system.

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

```
224 depend w,tt;
225 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

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

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

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

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

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

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

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

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

The operator `ou(f,tt,mu)` represents the convolution $\mathrm{e}^{\mu t}\star f$ 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)
    , ou(ou(~r,tt,~nu),tt,~mu) =>
      -sign(mu)*(ou(r,tt,mu)-ou(r,tt,nu))/(mu-nu)
      when (mu*nu>0)and(mu neq nu)
    };
```

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`.

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

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

```
depend yz,ttyz;
clear zres_; operator zres_; linear zres_;
let zres_(~a,ttyz,~r)=>ou(sign(df(a,yz)*yz/a-r)
    *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.

```
243 clear zint_; operator zint_; linear zint_;
244 clear znon_; operator znon_; linear znon_;
```

First, avoid obvious secularity.

```
245 let { zint_(w(~i),tt)=>0, znon_(w(~i),tt)=>w(i)
246     , zint_(1,tt)=>0, znon_(1,tt)=>1
247     , 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.

```
248     , zint_(ou(~r,tt,~mu),tt)
249       =>ou(r,tt,mu)/mu+zint_(r,tt)/abs(mu)
250     , 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.

```
251     , zint_(ou(~r,tt,~mu)^2,tt)=>ou(~r,tt,~mu)^2/(2*mu)
252                                  +zint_(r*ou(r,tt,mu),tt)/abs(mu)
253     , 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.

```
254     , zint_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)
255       =>ou(r,tt,mu)*ou(s,tt,nu)/(mu+nu)
```

```
256       +zint_(sign(mu)*r*ou(s,tt,nu)+sign(nu)*s*ou(r,tt,mu),tt)
257       /(mu+nu) when mu+nu neq 0
258     , znon_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
259       +znon_(sign(mu)*r*ou(s,tt,nu)+sign(nu)*s*ou(r,tt,mu),tt)
260       /(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.

```
261     , zint_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
262       ou(ou(r,tt,-nu),tt,-nu)*ou(s,tt,nu)
263       +zint_(ou(ou(r,tt,-nu),tt,-nu)*s,tt)
264       when (mu+nu=0)and(nu>0)
265     , znon_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
266       znon_(ou(ou(r,tt,-nu),tt,-nu)*s,tt)
267       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.

```
268 };
```

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.

```
269 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.

```
clear x_,y_,z_;
let { x(~j)=>x_(j), y(~j)=>y_(j), z(~j)=>z_(j) };
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 `X(i)`, `Y(i)` and `Z(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`.

```
clear X,Y,Z;
operator X; operator Y; operator Z;
depend X,t; depend Y,t; depend Z,t;
let { df(X(~i),t)=>ff(i)*cmanf
    , df(Y(~i),t)=>gg(i)*smanf
    , df(Z(~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):=X(i)*cmanf;
for i:=1:ny do y_(i):=Y(i)*smanf;
for i:=1:nz do z_(i):=Z(i)*umanf;
for i:=1:nx do ff(i):=0;
for i:=1:ny do gg(i):=-rats_(i)*Y(i)*smanf;
for i:=1:nz do hh(i):=+ratu_(i)*Z(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 $\prod_j Y_j^{q_j} Z_k^{r_k} \in$ 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 Y(i)=Y(i)*yz^rats_(i)$
z4z:=for i:=1:nz collect Z(i)=Z(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.

```
288 for iter:=1:maxiter do begin
289   ok:=1;
290   if trace then write "
291   ITERATION = ",iter,"
292   -------------";
```

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.

```
293   lengthresy:={};
294   for i:=1:ny do begin
295     res:=-df(y(i),t)+part(yrhs_,i);
296     ok:=if res=0 then ok else 0;
297     lengthresy:=append(lengthresy,{length(res)});
298     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.

```
299 %write
300     res:=sub(y4y,res);
301 %write
302     res0:=reso_(res,yz,+rats_(i));
303 %write
304     res1:=res-res0*yz^rats_(i);
305     if smanf then gg(i):=gg(i)+znon_(res0,tt)
306     else if znon_(res0,tt) neq 0
307         then rederr("oops y-res: contact me");
```

```
308     if trace then write "dY",i,"/dt = ",gg(i);
309 %write ratsi:=rats_(i);
310 %write zint0:=zint_(res0,tt);
311 %write zint1:=zres_(res1,ttyz,rats_(i));
312 %write
313     y_(i):=y_(i) +zint_(res0,tt) -zres_(res1,ttyz,rats_(i));
314     if trace then write "y",i," = ",y(i);
315   end;
316   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.

```
317   lengthresz:={};
318   for i:=1:nz do begin
319     res:=-df(z(i),t)+part(zrhs_,i);
320     ok:=if res=0 then ok else 0;
321     lengthresz:=append(lengthresz,{length(res)});
322     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.

```
323     res:=sub(y4y,res);
324     res0:=reso_(res,yz,-ratu_(i));
325     res1:=res-res0/yz^ratu_(i);
326     if umanf then hh(i):=hh(i)+znon_(res0,tt)
327     else if znon_(res0,tt) neq 0
328         then rederr("oops z-res: contact me");
329     z_(i):=z_(i) +zint_(res0,tt) -zres_(res1,ttyz,-ratu_(i));
330   end;
```

```
331   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.

```
332   lengthresx:={};
333   for i:=1:nx do begin
334     res:=-df(x(i),t) +part(xrhs_,i);
335     ok:=if res=0 then ok else 0;
336     lengthresx:=append(lengthresx,{length(res)});
337     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.

```
338     res:=sub(y4y,res);
339     res0:=reso_(res,yz,0);
340     res1:=res-res0;
341     if cmanf then ff(i):=ff(i)+znon_(res0,tt)
342     else if znon_(res0,tt) neq 0 then begin
343         write "**** Requested manifold may not exist due to";
344         write "slow x-residual component: res0=",res0;
345         write "**** So halving the max number of iterations";
346         maxiter:=maxiter/2;
347     end;
348     if trace then write "dX",i,"/dt = ",ff(i);
349     x_(i):=x_(i) +zint_(res0,tt) -zres_(res1,ttyz,0);
350     if trace then write "x",i," = ",x(i);
351   end;
352   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.

```
353   showtime;
354   if ok then write "Number of iterations ",
355     iter:=1000000+iter;
356 end;
```

9 Output results

Only proceed to print if terminated successfully.

```
357 if ok
358   then write "SUCCESS: converged to an expansion"
359   else <<write "FAILED TO CONVERGE; I EXIT";
360     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.

```
361 write "Stochastic/non-autonomous ",textask," coordinates"$
362 write "(to one order lower in both small and sigma)"$
363 for i:=1:nz do if trace then write z_(i):=z_(i)
364     else begin z_(i):=sigma*small*z_(i);
365          write z_(i):=z_(i)/small/sigma; end;
366 for i:=1:ny do if trace then write y_(i):=y_(i)
367     else begin y_(i):=sigma*small*y_(i);
368          write y_(i):=y_(i)/small/sigma; end;
369 for i:=1:nx do if trace then write x_(i):=x_(i)
370     else begin x_(i):=sigma*small*x_(i);
371          write x_(i):=x_(i)/small/sigma; end;
```

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

```
372 write "The ",textask," S/ODEs";
373 if umanf then for i:=1:nz do write "dZ(",i,")/dt = ",hh(i);
374 if smanf then for i:=1:ny do write "dY(",i,")/dt = ",gg(i);
```

```
375 if cmanf then for i:=1:nx do write "dX(",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.

```
376 clear order_;  operator order_;
377 defid order_,name="O";
378 defindex order_(arg,arg);
379 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.

```
380 out "scratchfile.red"$
381 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.

```
382 write "write ""\)
383 \paragraph{Time dependent ",textask," coordinates}
384 \("";";
385 for i:=1:nz do write "z_(",i,"):=z_(",i,
386     ") +order_(varepsilon^",toosmall-1,",sigma_^2);";
387 for i:=1:ny do write "y_(",i,"):=y_(",i,
388     ") +order_(varepsilon^",toosmall-1,",sigma_^2);";
389 for i:=1:nx do write "x_(",i,"):=x_(",i,
390     ") +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,
```

```
418 terms are near the end of each expression.
419 \input{stoNFreportSys}
420 "$
421 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. Switch `raise` back off.

```
422 out "stoNFreport.tex"$
423 on latex$
424 in "scratchfile.red"$
425 off latex$ off raise$
426 shut "stoNFreport.tex"$
```

10 Fin

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

```
427 clear x(~j),y(~j),z(~j);
428 if theman=csuman
429 then return Finished_constructing_normal_form_of_system
430 else return Finished_invariant_manifold_of_system$
431 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.

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

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

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

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

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

```
symbolic procedure latexon;
<<!*!*a2sfn:='texaeval;
  !*raise:=nil;
  prin2t "\documentclass[11pt,a5paper]{article}";
```

```
457   prin2t "\usepackage[a5paper,margin=13mm]{geometry}";
458   prin2t "\usepackage{parskip,time} \raggedright";
459   prin2t "\begin{document}\input{stoNFreportHdr}";
460   if !*verbatim then
461       <<prin2t "\begin{verbatim}";
462         prin2t "REDUCE Input:">>;
463   put('tex,'rtypefn,'(lambda(x) 'tex)) >>$
```

End the file when read by Reduce

```
464 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. (2009–2025), 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].