

# Computer algebra derives normal forms of general stochastic and non-autonomous differential equations

A. J. Roberts\*

November 19, 2008; modified April 20, 2021

## Abstract

Construct stochastic normal form of a wide class of non-autonomous or stochastic differential equations (herein abbreviated by s/ODEs), 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. To construct normal forms of deterministic autonomous differential equations, simply omit specifying any noise. This article documents code designed for an interactive web site (Roberts 2009–2021) that is available to all to use.

Modelling stochastic systems has many important applications. Stochastic coordinate transforms to a normal form is a powerful way of disentangling emergent long term dynamics. Since the analysis involves classic calculus, then the approach also applies to a wide class of non-autonomous dynamical systems. Further, cater for deterministic autonomous systems by simply omitting the time dependence in the system. For generality, this approach now caters for unstable modes, and for differential equation systems with a rational right-hand side.

---

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

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	A simple example: <code>exemplenormform()</code> . . . . .	5
<b>2</b>	<b>Header of the procedure</b>	<b>7</b>
2.1	Preamble to the procedure . . . . .	7
2.2	Extract and scale slow equations . . . . .	8
2.3	Extract and scale stable fast equations . . . . .	9
2.4	Extract and scale unstable fast equations . . . . .	11
<b>3</b>	<b>Setup LaTeX output using <code>rlfi</code></b>	<b>13</b>
<b>4</b>	<b>Delayed write of text info</b>	<b>17</b>
<b>5</b>	<b>Represent the noise</b>	<b>17</b>
<b>6</b>	<b>Operators to solve noisy homological equation</b>	<b>19</b>
<b>7</b>	<b>Initialise approximate transform</b>	<b>21</b>
<b>8</b>	<b>Iterative updates</b>	<b>22</b>
8.1	Fast stable modes . . . . .	23
8.2	Fast unstable modes . . . . .	23
8.3	Slow modes . . . . .	24
<b>9</b>	<b>Output results</b>	<b>25</b>
9.1	Plain text version . . . . .	25
9.2	L <sup>A</sup> T <sub>E</sub> X version . . . . .	26
<b>10</b>	<b>Fin</b>	<b>27</b>
<b>11</b>	<b>Override some <code>rlfi</code> procedures</b>	<b>28</b>

# 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 "invariantManifold.tex"$`<sup>1</sup> Test your installation by then executing `exemplenormform()`; (see [Section 1.1](#)).

**Execution** Thereafter, construct a specified invariant manifold of a specific dynamical system by executing the following command with specific values for the input parameters. See `allExamples.pdf` for many examples.

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

**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)`, the stable fast modes by `y(1)` through to `y(ny)`, and the unstable fast modes by `z(1)` through to `z(nz)`. Each Stratonovich white noise, derivative of a Stratonovich Wiener process, 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; but other labels for the noise may be used. Deterministic, autonomous, normal forms are constructed simply by omitting any noise term `w()` in the differential equations. The S/ODEs must be linearly diagonalised.<sup>2</sup>

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,  $\{\dots\}$ , of the right-hand sides of the S/ODEs for the slow variables  $x_j(t)$ ;

<sup>1</sup>This script changes many internal settings of *Reduce*, so best to do only when needed.

<sup>2</sup>Although a Jordan form is also acceptable, there are issues in the error control.

- `dydt`, a comma separated list within braces,  $\{\dots\}$ , of the right-hand sides of the S/ODEs for the fast stable variables  $y_j(t)$ ;
- `dzdt`, a comma separated list within braces,  $\{\dots\}$ , of the right-hand sides of the S/ODEs for the fast unstable variables  $z_j(t)$ ;
- `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  $\varepsilon$ , and constructs an invariant manifold, and evolution thereon, of the embedding system to the asymptotic error  $\mathcal{O}(\varepsilon^{\text{toosmall}})$  (as  $\varepsilon \rightarrow 0$ ). Often the introduced artificial  $\varepsilon$  has a useful physical meaning, but strictly you should evaluate the output at  $\varepsilon = 1$  to recover results for the specified system, and then interpret the results in terms of actual ‘small’ parameters.

The above right-hand side expressions for the time-derivatives must?? be multinomial in variables  $x_i$ ,  $y_i$ ,  $z_i$  and  $w_i$  (and also may include some  $\dot{x}_i$ ,  $\dot{y}_i$ , and  $\dot{z}_i$  effects).

**Outputs** This procedure reports the specified system, the embedded system it actually analyses, the number of iterations taken, the invariant manifold approximation, the evolution on the invariant manifold, and optionally a basis for projecting onto the invariant manifold.

- A plain text report to the Terminal window in which `Reduce` is executing—the invariant manifold is parametrised by variables `s(1)`, `s(2)`,  $\dots$ , and the dynamics by their evolution in time.
- A  $\text{\LaTeX}$  source report written to the file `stoNFreport.tex` (and `stoNFreportSys.tex`)—the invariant manifold is parametrised by variables  $s_1, s_2, \dots$ , and the dynamics by their evolution in time. Generate a pdf version by executing `pdflatex stoNFreport`.

One may change the appearance of the output somewhat. For example, it is often useful to execute `factor s`; before executing `stonormalform(...)` in order to group terms with the same powers of amplitudes/order-parameters/coarse-variables.

**Background** The theoretical support for the results of the analysis of this procedure is centre/stable/unstable manifold theory (e.g., Carr 1981, 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: `exemplenormform()`

Execute this example by invoking the command `exemplenormform()`; 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  $\sigma$  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’  $x, y, \sigma$  by executing the following procedure:

```
2 procedure exemplenormform;
3   stonormalform(
4     {-x(1)*y(1)},
5     {-y(1)+x(1)^2-2*y(1)^2+w(1)},
6     {} ,
7     3 )$
```

The procedure automatically multiplies the noise factors by a parameter **sigma** so there is no need include the parameter  $\sigma$  in the specification of the problem (unless you particularly want to), as it will be done for you.

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

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

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

using analysis and theory based upon the equilibrium at  $\varepsilon = \sigma = 0$ .

**The stochastic coordinate transform** (to one order lower in both  $\varepsilon$  and  $\sigma$  than actually constructed)

$$\begin{aligned} x_1 &= \sigma \varepsilon e^{-1t} \star w_1 X_1 + \varepsilon X_1 Y_1 + X_1 \\ y_1 &= 4\sigma \varepsilon e^{-1t} \star w_1 Y_1 + \sigma e^{-1t} \star w_1 + \varepsilon(X_1^2 + 2Y_1^2) + Y_1 \end{aligned}$$

### Result normal form SDEs

$$\begin{aligned} \dot{X}_1 &= 2\sigma^2 \varepsilon^2 e^{-1t} \star w_1 w_1 X_1 - \sigma \varepsilon w_1 X_1 - \varepsilon^2 X_1^3 \\ \dot{Y}_1 &= 8\sigma^2 \varepsilon^2 e^{-1t} \star w_1 w_1 Y_1 - 4\sigma \varepsilon w_1 Y_1 - 2\varepsilon^2 X_1^2 Y_1 - Y_1 \end{aligned}$$

- The  $\dot{Y}_1$  s/ODE shows that  $Y_1 = 0$  is invariant, and since dominantly is  $\dot{Y}_1 \approx -Y_1$  then it is almost always exponentially quickly attractive in some domain about the origin.
- The  $\dot{X}_1$  s/ODE is independent of  $Y_1$  and indicates an algebraic attraction to zero, albeit affected by a multiplicative noise, and moderated by some irreducible noise-noise interactions.
- These deductions are transformed into the original  $xy$ -space by the constructed time-dependent coordinate transformation.

## 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  $\text{\LaTeX}$ ; 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. Then 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;
```

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

```

16 operator stonormalform;
17 for all dxdt, dydt, dzdt, toosmall let
18     stonormalform(dxdt, dydt, dzdt, toosmall)
19     = 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).

```

20 scalar maxiter, trace, nx, ny, nz, offdiag, jac, ok, res,
21 res0, res1, lengthresx, lengthresy, lengthresz;

```

Write an intro message.

```

22 write "Construct a stochastic normal form (version 20 Apr 2021)"$

```

Define default parameters for the iteration: `maxiter` is the maximum number of allowed iterations; `toosmall` is the order of errors in the analysis in terms of the parameter `small`. Specific problems may override these defaults.

```

23 maxiter:=29$
24 factor small,sigma;

```

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

```

25 let sigma^3=>0;

```

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

```

26 trace:=0$
27 %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.

```

28 on div; off allfac; on revpri;
29 on rationalize;

```

## 2.2 Extract and scale slow equations

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

```

30 xrhs_:=dxdt$
31 write "no. of slow modes ",nx:=length(xrhs_);

```

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.



```
32 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 me to independently control the truncation in noise amplitude.

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

Section 4 writes the resulting differential equations for information.

```
35 if trace then for i:=1:nx do
36     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`.

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

**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:

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

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

```
41 rats_(i):=coeffn(part(yrhs_,i),y(i),1);
42 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.

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

```

45     rr:=0;
46     for j:=1:arglength(rats_(i)) do
47         if numberp(part(rats_(i),j))
48             then rr:=part(rats_(i),j);
49     rats_(i):=rr;
50 end;

```

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

```

51     rats_(i):=-rats_(i);

```

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

```

52     if numberp(rats_(i))and rats_(i)>0 then
53     else begin
54         write "***** Error *****
55         Linear coeffs of y-decay must be negative numbers";
56         return;
57     end;

```

End the loop over all right-hand sides.

```

58 end;
59 if trace then write "End loop over all dydt";

```

Flag later warning if the linear part not diagonal.

```

60 offdiag:=0$
61 for i:=1:ny do for j:=1:ny do if i neq j then begin
62     jac:=coeffn(part(yrhs_,i),y(j),1);
63     if (jac where {x(~k)=>0,y(~k)=>0,z(~k)=>0,w(~k)=>0}) neq 0
64     then offdiag:=1$
65 end;
66 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.

```
67 yrhs_:=for i:=1:ny collect
68   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.

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

Section 4 writes the resulting differential equations for information.

```
71 if trace then for i:=1:ny do
72   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`.

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

**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:

```
75 clear ratu_; array ratu_(nz);
76 for i:=1:nz do begin
```

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

```
77   ratu_(i):=coeffn(part(zrhs_,i),z(i),1);
78   ratu_(i):=(ratu_(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 growth rate.

```

79  if not numberp(ratu_(i)) then
80  if part(ratu_(i),0)=plus then begin
81    rr:=0;
82    for j:=1:arglength(ratu_(i)) do
83      if numberp(part(ratu_(i),j))
84        then rr:=part(ratu_(i),j);
85    ratu_(i):=rr;
86  end;

```

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

```

87  if numberp(ratu_(i))and ratu_(i)>0 then
88  else begin
89    write "***** Error *****
90    Linear coeffs of z-growth must be positive numbers";
91    return;
92  end;

```

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

```

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

```

Flag warning if the linear part not diagonal.

```

95 for i:=1:nz do for j:=1:nz do if i neq j then begin
96   jac:=coeffn(part(zrhs_,i),z(j),1);
97   if (jac where {x(~k)=>0,y(~k)=>0,z(~k)=>0,w(~k)=>0}) neq 0
98   then offdiag:=1$
99 end;
100 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.

```
101 zrhs_:=for i:=1:nz collect
102   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.

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

Section 4 writes the resulting differential equations for information.

```
105 if trace then for i:=1:nz do
106   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 L<sup>A</sup>T<sub>E</sub>X preamble appears the best option for the line breaking, but `\sloppy` would also work reasonably.

```
107 mathstyle math;
```

**Define names for L<sup>A</sup>T<sub>E</sub>X formatting** Define some names I use, so that `rlfi` translates them to Greek characters in the L<sup>A</sup>T<sub>E</sub>X.

```
108 defid small,name="\eps";%varepsilon;
109 defid alpha,name=alpha;
110 defid beta,name=beta;
111 defid gamma,name=gamma;
112 defid delta,name=delta;
113 defid epsilon,name=epsilon;
114 defid varepsilon,name=varepsilon;
115 defid zeta,name=zeta;
```

```
116 defid eta,name=eta;
117 defid theta,name=theta;
118 defid vartheta,name=vartheta;
119 defid iota,name=iota;
120 defid kappa,name=kappa;
121 defid lambda,name=lambda;
122 defid mu,name=mu;
123 defid nu,name=nu;
124 defid xi,name=xi;
125 defid pi,name=pi;
126 defid varpi,name=varpi;
127 defid rho,name=rho;
128 defid varrho,name=varrho;
129 defid sigma,name=sigma;
130 defid varsigma,name=varsigma;
131 defid tau,name=tau;
132 defid upsilon,name=upsilon;
133 defid phi,name=phi;
134 defid varphi,name=varphi;
135 defid chi,name=chi;
136 defid psi,name=psi;
137 defid omega,name=omega;
138 defid Gamma,name=Gamma;
139 defid Delta,name=Delta;
140 defid Theta,name=Theta;
141 defid Lambda,name=Lambda;
142 defid Xi,name=Xi;
143 defid Pi,name=Pi;
144 defid Sigma,name=Sigma;
145 defid Upsilon,name=Upsilon;
146 defid Phi,name=Phi;
147 defid Psi,name=Psi;
148 defid Omega,name=Omega;
```

For the variables names I use, as operators, define how they appear in the

$\text{\LaTeX}$ , and also define that their arguments appear as subscripts.

```

149 \defindex w(down);
150 \defindex x(down);
151 \defindex y(down);
152 \defindex z(down);
153 \defid xx,name="X";
154 \defid yy,name="Y";
155 \defid zz,name="Z";
156 \defindex xx(down);
157 \defindex yy(down);
158 \defindex zz(down);
159 \defindex hh(down);
160 \defindex gg(down);
161 \defindex ff(down);

```

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

```

162 \defid hh,name="\dot z";
163 \defid gg,name="\dot y";
164 \defid ff,name="\dot x";

```

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

```

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

```

**Write the  $\text{\LaTeX}$  dynamical system** Because of the way *rlfi* works, to get good quality output to the  $\text{\LaTeX}$  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  $\text{\LaTeX}$  file. In this convoluted way I avoid extraneous output lines polluting the  $\text{\LaTeX}$ .

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

```
167 clear ff,gg,hh;
168 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.

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

Then switch on  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  output before writing to file as this  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  file is to be input from the main  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  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  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ .

```
179 write "Ignore the following 15 lines of LaTeX"$
180 on latex$
181 out "stoNFreportSys.tex"$
182 in "scratchfile.red"$
183 shut "stoNFreportSys.tex"$
184 off latex$
```



## 4 Delayed write of text info

Because it is messy to interleave L<sup>A</sup>T<sub>E</sub>X and plain output, I delay writing anything much in plain text until here.

Write the delayed warning message about off-diagonal terms.

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

Write the plain text versions of the dynamical system.

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

## 5 Represent the noise

The white noises  $w$  depend upon time. But we find it useful to discriminate upon the notionally fast time fluctuations of the noise processes, 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  $z_i$ .

```
196 depend w,tt;
197 depend tt,t,ttzy;
```

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

any non-zero parameter  $\mu$ , define the Ornstein–Uhlenbeck convolution

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

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

$$e^{\mu t} \star 1 = \frac{1}{|\mu|}, \quad (2)$$

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

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

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

$$e^{\mu t} \star e^{\nu t} \star = \begin{cases} \frac{1}{|\mu - \nu|} [e^{\mu t} \star + e^{\nu t} \star], & \mu \nu < 0, \\ \frac{-\operatorname{sgn} \mu}{\mu - \nu} [e^{\mu t} \star - e^{\nu t} \star], & \mu \nu > 0 \text{ \& } \mu \neq \nu. \end{cases} \quad (6)$$

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. 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 [Section 5](#) and its action upon deterministic terms [Section 5](#):

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

Also code the transform [Section 5](#) that successive convolutions at different rates may be transformed into several single convolutions.

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

```

203      , ou(ou(~r,tt,~nu),tt,~mu) =>
204      -sign(mu)*(ou(r,tt,mu)-ou(r,tt,nu))/(mu-nu)
205      when (mu*nu>0)and(mu neq nu)
206  };

```

The above properties are *critical*: they must be correct for the results to be correct.

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`.

```

207 clear reso_; operator reso_; linear reso_;
208 let { reso_(~a,yz,~r)=>1 when df(a,yz)*yz=r*a
209      , reso_(~a,yz,~r)=>0 when df(a,yz)*yz neq r*a
210      };

```

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

```

211 depend yz,tt,yz;
212 clear zres_; operator zres_; linear zres_;
213 let zres_(~a,tt,yz,~r)=>ou(sign(df(a,yz)*yz/a-r)
214      *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  $\xi$ , 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. 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. 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.

```
215 clear zint_; operator zint_; linear zint_;
216 clear znon_; operator znon_; linear znon_;
```

First, avoid obvious secularity.

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

Second, by [Section 5](#) a convolution may be split into an integrable part, and a part in its argument which in turn may be integrable or not.

```
220     , zint_(ou(~r,tt,~mu),tt)=>ou(r,tt,mu)/mu+zint_(r,tt)/abs(mu)
221     , 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.

```
222     , zint_(ou(~r,tt,~mu)^2,tt)=>ou(~r,tt,~mu)^2/(2*mu)
223       +zint_(r*ou(r,tt,mu),tt)/abs(mu)
224     , 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.

```
225     , zint_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)
226       =>ou(r,tt,mu)*ou(s,tt,nu)/(mu+nu)
227       +zint_(sign(mu)*r*ou(s,tt,nu)+sign(nu)*s*ou(r,tt,mu),tt)
228       /(mu+nu) when mu+nu neq 0
229     , znon_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
230       +znon_(sign(mu)*r*ou(s,tt,nu)+sign(nu)*s*ou(r,tt,mu),tt)
231       /(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.

```

232     , zint_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
233     ou(ou(r,tt,-nu),tt,-nu)*ou(s,tt,nu)
234     +zint_(ou(ou(r,tt,-nu),tt,-nu)*s,tt) when (mu+nu=0)and(nu>0)
235     , znon_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
236     znon_(ou(ou(r,tt,-nu),tt,-nu)*s,tt) 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.

```

237 };

```

## 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.

```

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

```

Variables `x`, `y` and `z` were operators in the specification of the equations. We now want them to store the approximation to the coordinate transform, so clear and reallocate as storage for the normal form expressions. Need to clear the mapping to the array before exiting.

```

239 clear x_,y_,z_;
240 let { x(~j)=>x_(j), y(~j)=>y_(j), z(~j)=>z_(j) };
241 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`.

```

242 clear xx,yy,zz;
243 operator xx; operator yy; operator zz;

```

```

244 depend xx,t; depend yy,t; depend zz,t;
245 let { df(xx(~i),t)=>ff(i)
246      , df(yy(~i),t)=>gg(i)
247      , df(zz(~i),t)=>hh(i) };

```

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  $\text{ff}(i)$ ,  $\dot{Y}_i \approx -r_i Y_i$ , in  $\text{gg}(i)$ , and  $\dot{Z}_i \approx +r_i Z_i$ , in  $\text{hh}(i)$ .

```

248 for i:=1:nx do x_(i):=xx(i);
249 for i:=1:ny do y_(i):=yy(i);
250 for i:=1:nz do z_(i):=zz(i);
251 for i:=1:nx do ff(i):=0;
252 for i:=1:ny do gg(i):=-rats_(i)*yy(i);
253 for i:=1:nz do hh(i):=+ratu_(i)*zz(i);

```

Update the  $Y_i$  evolution  $\text{gg}(i)$  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  $\text{yz}$  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.

```

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

```

## 8 Iterative updates

We iterate to a solution of the governing S/ODEs to residuals of some order of error. For the moment, iterate for a maximum of nineteen iterations and to the pre-specified errors.

```

257 for iter:=1:maxiter do begin
258   ok:=1;
259   if trace then write "
260     ITERATION = ",iter,"
261     -----";

```

## 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.

```

262  lengthresy:={};
263  for i:=1:ny do begin
264      res:=-df(y(i),t)+part(yrhs_,i);
265      ok:=if res=0 then ok else 0;
266      lengthresy:=append(lengthresy,{length(res)});
267      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.

```

268      res:=sub(y4y,res);
269      res0:=reso_(res,yz,+rats_(i));
270      res1:=res-res0*yz^rats_(i);
271      gg(i):=gg(i)+znon_(res0,tt);
272      if trace then write "dY",i,"/dt = ",gg(i);
273      y_(i):=y_(i) +zint_(res0,tt) -zres_(res1,ttyz,rats_(i));
274      if trace then write "y",i," = ",y(i);
275  end;
276  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.

```

277  lengthresz:={};
278  for i:=1:nz do begin
279      res:=-df(z(i),t)+part(zrhs_,i);

```

```

280     ok:=if res=0 then ok else 0;
281     lengthresz:=append(lengthresz,{length(res)});
282     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.

```

283     res:=sub(y4y,res);
284     res0:=reso_(res,yz,-ratu_(i));
285     res1:=res-res0/yz^ratu_(i);
286     hh(i):=hh(i)+znnon_(res0,tt);
287     z_(i):=z_(i) +zint_(res0,tt) -zres_(res1,ttyz,-ratu_(i));
288 end;
289 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.

```

290     lengthresx:={};
291     for i:=1:nx do begin
292         res:=-df(x(i),t) +part(xrhs_,i);
293         ok:=if res=0 then ok else 0;
294         lengthresx:=append(lengthresx,{length(res)});
295         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.

```

296     res:=sub(y4y,res);
297     res0:=reso_(res,yz,0);
298     res1:=res-res0;
299     ff(i):=ff(i)+znnon_(res0,tt);

```



```

300     if trace then write "dX",i,"/dt = ",ff(i);
301     x_(i):=x_(i) +zint_(res0,tt) -zres_(res1,tt,yz,0);
302     if trace then write "x",i," = ",x(i);
303 end;
304 if nx>0 then write lengthresx:=lengthresx;

```

Terminate the iteration loop once all residuals are zero.

```

305 showtime;
306 if ok then write "Number of iterations ",
307     iter:=1000000+iter;
308 end;

```

## 9 Output results

Only proceed to print if terminated successfully.

```

309 if ok
310 then write "SUCCESS: converged to an expansion"
311 else <<write "FAILED TO CONVERGE; I EXIT";
312     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.

```

313 write "The stochastic/non-autonomous coordinate transform,
314 (to one order lower in both small and sigma)";
315 for i:=1:nz do begin z_(i):=sigma*small*z_(i);
316     write z_(i):=z_(i)/small/sigma; end;
317 for i:=1:ny do begin y_(i):=sigma*small*y_(i);
318     write y_(i):=y_(i)/small/sigma; end;
319 for i:=1:nx do begin x_(i):=sigma*small*x_(i);
320     write x_(i):=x_(i)/small/sigma; end;

```

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

```
321 write "The normal form S/ODEs";
322 for i:=1:nz do write "dzz(",i,")/dt = ",hh(i);
323 for i:=1:ny do write "dyy(",i,")/dt = ",gg(i);
324 for i:=1:nx do write "dxx(",i,")/dt = ",ff(i);
```

## 9.2 L<sup>A</sup>T<sub>E</sub>X version

As before, we have to write expressions to file for later reading so they get printed without extraneous dross in the L<sup>A</sup>T<sub>E</sub>X source. First open up the temporary file `scratchfile.red` again.

```
325 out "scratchfile.red";
326 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 a spurious fatal blank line in the L<sup>A</sup>T<sub>E</sub>X.

```
327 write "write ""\end{math}
328 \paragraph{Time dependent coordinate transform}
329 \begin{math}"";";
330 for i:=1:nz do write "z_(",i,"):=z_(",i,");";
331 for i:=1:ny do write "y_(",i,"):=y_(",i,");";
332 for i:=1:nx do write "x_(",i,"):=x_(",i,");";
```

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

```
333 write "write ""\end{math}
334 \paragraph{Result normal form DEs}
335 \begin{math}"";";
336 for i:=1:nz do write "hh(",i,"):=hh(",i,");";
337 for i:=1:ny do write "gg(",i,"):=gg(",i,");";
338 for i:=1:nx do write "ff(",i,"):=ff(",i,");";
```

```
339 write "end;";
```

Shut the temporary output file.

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

Get expressions from file and write the main L<sup>A</sup>T<sub>E</sub>X file. But first redefine how these names get printed, namely as the normal form time derivatives.

```
341 defid x_,name="x"; defindex x_(down);
```

```
342 defid y_,name="y"; defindex y_(down);
```

```
343 defid z_,name="z"; defindex z_(down);
```

```
344 defid hh,name="\dot Z";
```

```
345 defid gg,name="\dot Y";
```

```
346 defid ff,name="\dot X";
```

Finally write to the main L<sup>A</sup>T<sub>E</sub>X file so switch on latex after starting to write to the file. Then write expressions in `scratchfile.red` to `stoNFreport.tex` as nice L<sup>A</sup>T<sub>E</sub>X. Switch off latex, to get the end of the document, and finish writing.

```
347 out "stoNFreport.tex"$
```

```
348 on latex$
```

```
349 in "scratchfile.red"$
```

```
350 off latex$
```

```
351 shut "stoNFreport.tex"$
```

## 10 *Fin*

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

```
352 clear x(~j),y(~j),z(~j);
```

```
353 return Finished_constructing_normal_form_of_system$
```

```
354 end$
```

## 11 Override some *rlfi* procedures

Now setup the *rlfi* package to write a  $\text{\LaTeX}$  version of the output. It is all a bit tricky and underhand. We override some stuff from *rlfi.red*.<sup>3</sup>

First, change **name** to get Big delimiters, not left-right delimiters, so  $\text{\LaTeX}$  can break lines.

```
355 deflist('(( ( !\b!i!g!() (!) !\b!i!g!)) (!P!I !\p!i! )
356           (!p!i !\p!i! ) (!E !e) (!I !i) (e !e) (i !i)), 'name)$
```

Override the procedure that prints annoying messages about multicharacter symbols. It ends the output of one expression. This is just a copy from *rlfi.red* with the appropriate if-statement deleted.

```
357 symbolic procedure prinlaend;
358 <<terpri();
359   prin2 "\end{";
360   prin2 mstyle!*;
361   prin2t "}\par";
362   if !*verbatim then
363     <<prin2t "\begin{verbatim}";
364     prin2t "REDUCE Input:">>;
365   ncharspr!*:=0;
366   if ofl!* then linelength(car linel!*)
367     else laline!*:=cdr linel!*;
368   nochar!*:=append(nochar!*,nochar1!*);
369   nochar1!*:=nil >>$
```

Override the procedure that outputs the  $\text{\LaTeX}$  preamble upon the command `on latex`.

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

---

<sup>3</sup>Find it in [reduce-algebra/trunk/packages/misc/rlfi.red](https://github.com/roberts/rlfi/blob/master/reduce-algebra/trunk/packages/misc/rlfi.red)

```

374 prin2t "\usepackage[a5paper,margin=13mm]{geometry}";
375 prin2t "\usepackage{parskip,time} \raggedright";
376 prin2t "\def\ou\big(#1,#2,#3\big){\{\rm e\}^{\if#3\else#3\fi t}}";
377 prin2t "\def\eps{\varepsilon}";
378 prin2t "\title{Normal form of your dynamical system}";
379 prin2t "\author{A. J. Roberts, University of Adelaide\\}";
380 prin2t "\texttt{http://orcid.org/0000-0001-8930-1552}}";
381 prin2t "\date{\now, \today}";
382 prin2t "\begin{document}";
383 prin2t "\maketitle";
384 prin2t "Throughout and generally: the lowest order, most";
385 prin2t "important, terms are near the end of each expression.";
386 prin2t "\input{stoNFreportSys}";
387 if !*verbatim then
388     <<prin2t "\begin{verbatim}";
389     prin2t "REDUCE Input:">>;
390 put('tex,'rtypefn,'(lambda(x) 'tex)) >>$

```

End the file when input to Reduce

```
391 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](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. (2009–2021), 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>].