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

A. J. Roberts*

November 19, 2008; modified April 19, 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

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"$` [1] Test your installation by then executing `examplenormform();` (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.

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

[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.

- `dydt`, a comma separated list within braces, $\{\ldots\}$, of the right-hand sides of the S/ODEs for the fast stable variables $y_j(t)$;
- `dzdt`, a comma separated list within braces, $\{\ldots\}$, of the right-hand sides of the S/ODEs for the fast unstable variables $z_j(t)$;
- `toosmall`, an integer giving the desired order of error in the asymptotic approximation that is constructed. The procedure embeds the specified system in a family of systems parametrised by ε, and constructs an invariant manifold, and evolution thereon, of the embedding system to the asymptotic error $\mathcal{O}\big(\varepsilon^{\texttt{toosmall}}\big)$ (as $\varepsilon \to 0$). Often the introduced artificial ε has a useful physical meaning, but strictly you should evaluate the output at $\varepsilon = 1$ to recover results for the specified system, and then interpret the results in terms of actual 'small' parameters.

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)`, ..., and the dynamics by their evolution in time.
- A LaTeX source report written to the file `stoNFreport.tex` (and `stoNFreportSys.tex`)—the invariant manifold is parametrised by variables s_1, s_2, ..., 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: `examplenormform()`

Execute this example by invoking the command `examplenormform();` 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 procedure:

```
procedure examplenormform;
    stonormalform(
    {-x(1)*y(1)},
    {-y(1)+x(1)^2-2*y(1)^2+w(1)},
    {},
    3 )$
```

The procedure automatically multiplies the noise factors by a parameter `sigma` so there is no need include the parameter σ 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 ε. 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:

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

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

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

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

Result normal form SDEs

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

- 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/ODEis 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 LaTeX; and operator names for the variables of the S/ODEs.

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

Cater for rational function S/ODEs by allowing time dependence in these variables at specification. 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.

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

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

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

2.1 Preamble to the procedure

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

```
scalar maxiter, trace, nx, ny, nz, offdiag, jac, ok, res, res0,
```

Write an intro message.

```
write "Construct a stochastic normal form (version 19 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.

```
maxiter:=29$
factor small,sigma;
```

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

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

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

```
trace:=0$
%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.

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

2.2 Extract and scale slow equations

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

```
xrhs_:=dxdt$
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.

```
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.

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

Section 4 writes the resulting differential equations for information.

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

2.3 Extract and scale stable fast equations

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

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

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:

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

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

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

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

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

```
    for j:=1:arglength(rats_(i)) do
      if numberp(part(rats_(i),j))
      then rr:=part(rats_(i),j);
    rats_(i):=rr;
  end;
```

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

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

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

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

End the loop over all right-hand sides.

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

Flag later warning if the linear part not diagonal.

```
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_);
```

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:

```
array ratu_(nz);
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.

```
  ratu_(i):=coeffn(part(zrhs_,i),z(i),1);
  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.

```
   if not numberp(ratu_(i)) then
   if part(ratu_(i),0)=plus then begin
     rr:=0;
     for j:=1:arglength(ratu_(i)) do
       if numberp(part(ratu_(i),j))
       then rr:=part(ratu_(i),j);
     ratu_(i):=rr;
   end;
```

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

```
   if numberp(ratu_(i))and ratu_(i)>0 then
   else begin
     write "***** Error *****
     Linear coeffs of z-growth must be positive numbers";
     return;
   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.

```
mathstyle math;
```

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

```
defid small,name="\eps";%varepsilon;
defid alpha,name=alpha;
defid beta,name=beta;
defid gamma,name=gamma;
defid delta,name=delta;
defid epsilon,name=epsilon;
defid varepsilon,name=varepsilon;
defid zeta,name=zeta;
```

```
defid eta,name=eta;
defid theta,name=theta;
defid vartheta,name=vartheta;
defid iota,name=iota;
defid kappa,name=kappa;
defid lambda,name=lambda;
defid mu,name=mu;
defid nu,name=nu;
defid xi,name=xi;
defid pi,name=pi;
defid varpi,name=varpi;
defid rho,name=rho;
defid varrho,name=varrho;
defid sigma,name=sigma;
defid varsigma,name=varsigma;
defid tau,name=tau;
defid upsilon,name=upsilon;
defid phi,name=phi;
defid varphi,name=varphi;
defid chi,name=chi;
defid psi,name=psi;
defid omega,name=omega;
defid Gamma,name=Gamma;
defid Delta,name=Delta;
defid Theta,name=Theta;
defid Lambda,name=Lambda;
defid Xi,name=Xi;
defid Pi,name=Pi;
defid Sigma,name=Sigma;
defid Upsilon,name=Upsilon;
defid Phi,name=Phi;
defid Psi,name=Psi;
defid Omega,name=Omega;
```

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

LaTeX, and also define that their arguments appear as subscripts.

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

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

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

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

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

Write the LaTeX dynamical system Because of the way rfli works, to get good quality output to the LaTeX document, I need to write the algebraic expressions to a file, then read them back in again. While being read back in, I send the output to the LaTeX file. In this convoluted way I avoid extraneous output lines polluting the LaTeX.

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

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

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

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

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

```
write "Ignore the following 15 lines of LaTeX"$
on latex$
out "stoNFreportSys.tex"$
in "scratchfile.red"$
shut "stoNFreportSys.tex"$
off latex$
```

4 Delayed write of text info

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

Write the delayed warning message about off-diagonal terms.

```
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 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.

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

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 μ, define the Ornstein–Uhlenbeck convolution

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

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

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

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

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

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

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

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. 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:

```
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 Section 5 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.

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`.

```
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;
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. 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.

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

First, avoid obvious secularity.

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

Second, by 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.

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

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

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

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

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

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

```
    , zint_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
      ou(ou(r,tt,-nu),tt,-nu)*ou(s,tt,nu)
      +zint_(ou(ou(r,tt,-nu),tt,-nu)*s,tt) when (mu+nu=0)and(nu>0
    , znon_(ou(~r,tt,~mu)*ou(~s,tt,~nu),tt)=>
      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.

```
};
```

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.

```
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.

```
clear x,y,z;
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`.

```
operator xx; operator yy; operator zz;
depend xx,t; depend yy,t; depend zz,t;
let { df(xx(~i),t)=>ff(i)
    , df(yy(~i),t)=>gg(i)
    , 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 `ff(i)`, $\dot Y_i \approx -r_iY_i$, in `gg(i)`, and $\dot Z_i \approx +r_iZ_i$, in `hh(i)`.

```
for i:=1:nx do x(i):=xx(i);
for i:=1:ny do y(i):=yy(i);
```

```
for i:=1:nz do z(i):=zz(i);
for i:=1:nx do ff(i):=0;
for i:=1:ny do gg(i):=-rats_(i)*yy(i);
for i:=1:nz do hh(i):=+ratu_(i)*zz(i);
```

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

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

8 Iterative updates

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

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

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.

```
  lengthresy:={};
  for i:=1:ny do begin
```

```
    res:=-df(y(i),t)+part(yrhs_,i);
    ok:=if res=0 then ok else 0;
    lengthresy:=append(lengthresy,{length(res)});
    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.

```
    res:=sub(y4y,res);
    res0:=reso_(res,yz,+rats_(i));
    res1:=res-res0*yz^rats_(i);
    gg(i):=gg(i)+znon_(res0,tt);
    if trace then write "dY",i,"/dt = ",gg(i);
    y(i):=y(i) +zint_(res0,tt) -zres_(res1,ttyz,rats_(i));
    if trace then write "y",i," = ",y(i);
  end;
  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.

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

```
    res:=sub(y4y,res);
    res0:=reso_(res,yz,-ratu_(i));
    res1:=res-res0/yz^ratu_(i);
    hh(i):=hh(i)+znon_(res0,tt);
    z(i):=z(i) +zint_(res0,tt) -zres_(res1,ttyz,-ratu_(i));
  end;
  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.

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

```
    res:=sub(y4y,res);
    res0:=reso_(res,yz,0);
    res1:=res-res0;
    ff(i):=ff(i)+znon_(res0,tt);
    if trace then write "dX",i,"/dt = ",ff(i);
    x(i):=x(i) +zint_(res0,tt) -zres_(res1,ttyz,0);
    if trace then write "x",i," = ",x(i);
  end;
  if nx>0 then write lengthresx:=lengthresx;
```

Terminate the iteration loop once all residuals are zero.

```
  showtime;
```

```
  if ok then write "Number of iterations ",
    iter:=1000000+iter;
end;
```

9 Output results

Only proceed to print if terminated successfully.

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

9.1 Plain text version

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

```
write "The stochastic/non-autonomous coordinate transform,
(to one order lower in both small and sigma)";
for i:=1:nz do begin z(i):=sigma*small*z(i);
               write z(i):=z(i)/small/sigma; end;
for i:=1:ny do begin y(i):=sigma*small*y(i);
               write y(i):=y(i)/small/sigma; end;
for i:=1:nx  do begin x(i):=sigma*small*x(i);
               write x(i):=x(i)/small/sigma; end;
```

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

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

9.2 LaTeX version

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

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

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

```
write "write ""\end{math}
\paragraph{Time dependent coordinate transform}
\begin{math}"";";
for i:=1:nz do write "z(",i,"):=z(",i,");";
for i:=1:ny do write "y(",i,"):=y(",i,");";
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.

```
write "write ""\end{math}
\paragraph{Result normal form DEs}
\begin{math}"";";
for i:=1:nz do write "hh(",i,"):=hh(",i,");";
for i:=1:ny do write "gg(",i,"):=gg(",i,");";
for i:=1:nx do write "ff(",i,"):=ff(",i,");";
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 hh,name="\dot Z";
defid gg,name="\dot Y";
defid ff,name="\dot X";
```

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

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

10 Fin

That's all folks, so end the procedure.

```
return Finished_constructing_normal_form_of_system$
end$
```

11 Override some rlfi procedures

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

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

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

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

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.

```
symbolic procedure prinlaend;
<<terpri();
  prin2 "\end{";
  prin2 mstyle!*;
  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 >>$
```

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}";
  prin2t "\usepackage[a5paper,margin=13mm]{geometry}";
  prin2t "\usepackage{parskip,time} \raggedright";
  prin2t "\def\ou\big(#1,#2,#3\big){{{\rm e}^{\if#31\else#3\fi t
  prin2t "\def\eps{\varepsilon}";
  prin2t "\title{Normal form of your dynamical system}";
  prin2t "\author{A. J. Roberts, University of Adelaide\\";
  prin2t "\texttt{http://orcid.org/0000-0001-8930-1552}}";
  prin2t "\date{\now, \today}";
  prin2t "\begin{document}";
  prin2t "\maketitle";
  prin2t "Throughout and generally: the lowest order, most";
```

```
prin2t "important, terms are near the end of each expression."
prin2t "\input{stoNFreportSys}";
if !*verbatim then
    <<prin2t "\begin{verbatim}";
      prin2t "REDUCE Input:">>;
put('tex,'rtypefn,'(lambda(x) 'tex)) >>$
```

End the file when input to Reduce

```
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. (2009–2021), 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].