xSPDE user's guide: v1.22

Peter D. Drummond, Rodney Polkinghorne, Simon Kiesewetter

June 11, 2016

Centre for Quantum and Optical Science, Swinburne University of Technology, Melbourne, Victoria, Australia.

Contents

1 Introduction

The xSPDE software toolbox is an e**x**tensible **S**tochastic **P**artial **D**ifferential **E**quation solver. It can solve any number of simultaneous equations in any number of space dimensions, with averaging and graphical output. This user's guide can be used as the basis for a 3-4 hour self-taught tutorial in solving practical stochastic problems. It includes exercises for the reader. A full-length manual is available online. Research use of this software is encouraged, and should be referenced to the corresponding paper[1]. See: **www.github.com/peterddrummond/xspde_matlab.**

1.1 Stochastic equations

Stochastic equations are equations with random noise terms [2, 3]. They occur in in many fields of biology, chemistry, engineering, economics, physics, meteorology and other disciplines. An ordinary stochastic equation for a real or complex vector $\mathbf{a}$ is:

$$\frac{\partial \boldsymbol{a}}{\partial t} = \dot{\boldsymbol{a}} = \mathbf{A}\left(\boldsymbol{a}\right) + \underline{\mathbf{B}}\left(\boldsymbol{a}\right)\boldsymbol{w}(t). \tag{1.1}$$

Here, $\mathbf{A}$ is a vector, $\underline{\mathbf{B}}$ a matrix, and $\mathbf{w}$ is a real noise vector, delta-correlated in time:

$$\left\langle w_i\left(t\right) w_j\left(t\right)\right\rangle \quad = \quad \delta\left(t-t'\right)\delta_{ij} \tag{1.2}$$

Stochastic partial differential equations including space dependence are also numerically soluble [4, 5] with xSPDE. These are described in Chapter 4.

1.2 xSPDE

xSPDE is a Matlab based software toolbox. It has functions that can numerically solve both ordinary and partial differential stochastic equations, and graph results of correlations and averages. There are many equations of this type, and the xSPDE toolbox can treat a wide range. The extensible general structure permits drop-in replacements of the functions provided. Different simulations can be carried out sequentially, to simulate the various stages in an experiment or other process. It can even be used with no noise.

The code supports parallelism at both the vector instruction level and at the thread level, using Matlab matrix instructions and the parallel toolbox. It calculates averages of arbitrary functions of any number of complex or real fields, as well as Fourier transforms in time or space. It provides error estimates for both step-size and sampling error.

xSPDE is distributed without any guarantee, under the MIT open-source license. Contributions and bug reports always welcome.

2 xSPDE simulations

The general purpose xSPDE toolbox function co is:

- **xspde(in)**, the combined simulation (xSIM) and graphics (xGRAPH) function.

2.1 Running xSPDE interactively

To simulate a stochastic equation interactively, first make sure Matlab path is pointing to the xSPDE folder. If not, proceed as follows:

- Click on the Matlab HOME tab (top left), then Set Path
- Click on Add with Subfolders
- Find the xSPDE folder in the drop-down menu, and select it
- Click on close to save the path.

Next, type *clear* to clear old data, and enter the xSPDE inputs and functions into the command window. For the simplest of examples, do this by cutting and pasting from an electronic file of this manual. You can also use the Matlab built-in editor.

A typical script is as follows:

$$\begin{aligned} \texttt{in}.[label1] &= [parameter1] \\ \texttt{in}.[label2] &= \ldots \\ \texttt{in.da} &= @(\texttt{a},\texttt{w},\texttt{r})\ [derivative] \\ \texttt{xspde}(\texttt{in}) & \end{aligned}$$

Note the following points to remember:

- The notation $\texttt{in}.[label1] = [parameter1]$ inputs a value to the `in.` data structure
- There are many input possibilities, and they all have default values.
- You don't have to save the data if you just want an immediate plot
- The notation $\texttt{in.da} = @(\texttt{a},\texttt{w},\texttt{r})\ [derivative]$ = defines an inline function, da/dt.
- In xSPDE, `a` is the stochastic variable, `w` the random noise, and `r` the parameters.
- For example, `r.t` is the current time, and `r.x`, `r.y`, `r.z`, the grid of coordinates.

2.2 Inputs

All xSPDE simulations use a structure *in* for input data. All functions use an internally generated structure r, combining *in* with other generated parameters. Any name will do for either structure, as long as you are consistent.

All xSPDE inputs have default values, which can be changed by typing parameters into the *in* structure. If you only need the first value of a vector or array, just input the value that is required.

2.2.1 Common xSPDE inputs

The most common xSPDE parameters are:

Label	Type	Default value	Description
in.fields	integer	1	Stochastic field variables
in.noises	integer	1	Number of noises
in.dimension	integer	1	Space-time dimensions
in.name	string	' '	Simulation name
in.olabels	string	*'a'*	Observable labels
in.da	function	*0*	The stochastic derivative
in.initial	function	0	Function to initialize variables
in.ensembles	integer vector	[1,1,1]	Hierarchy of stochastic trajectories
in.observe	function	*a*	Observable function
in.linear	function	0	Linear interaction picture response
in.ranges	real vector	10	Ranges in time and space
in.steps	integer vector	1	Time-steps per plot point
in.points	integer vector	51	Output lattice points in [t,x,y,z]
in.images	integer cell array	{*0,..*}	Movie images in time per observable
in.transforms	vector cells	{*[0 0 0 0],..*}	Output Fourier transforms in [t,x,y,z]

A complete list of user inputs is given in Chapter (6).

2.2.2 Use the dot

Matlab formulae often require a 'dot' or a 'colon'. To multiply vectors element-wise, like $a_i = b_i c_i$, the notation $a = b. * c$ indicates all elements are multiplied. This is used in xspde to speed up calculations in parallel.

A formula for a stochastic field usually requires you to address the first index - which is the field component - and treat all the other elements in parallel. To do this, use the notation $a(1,:)$.

Whenever a formula combines operations over spatial lattices or ensembles, remember to **USE THE DOT**.

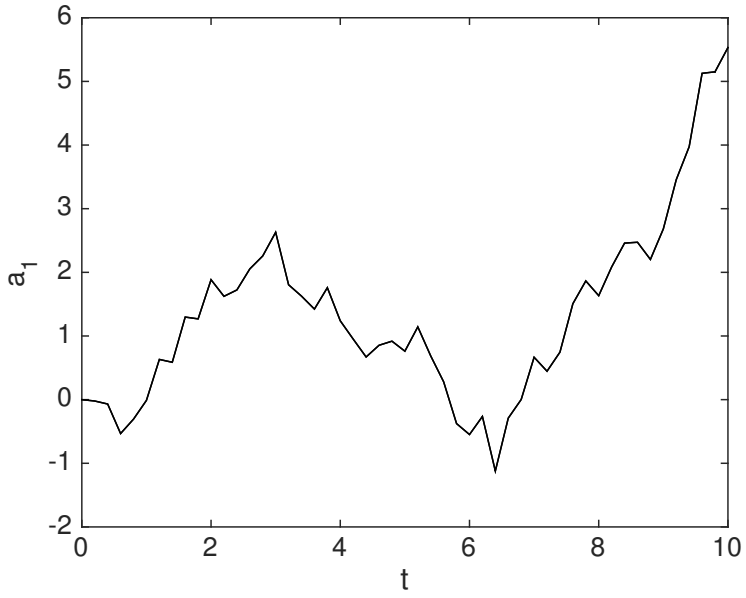

Figure 2.1: The simplest case: a random walk.

2.3 The random walk

The first example is the simplest possible stochastic equation:

$$\dot{a} = w(t)\,, \tag{2.1}$$

Exercises

Remember that, unless you type *clear* first, any changes to the input structure are additive; so you will get the sum of all the previous inputs as well as your new input.

- **Run the complete xSPDE script given below in Matlab.**

It is simplest to cut and paste from an electronic file into the Matlab command window. Be careful; pasting can cause subtle changes that may require correction.

You should get the output in Fig (2.1).

```
in.da=@(a,w,r) w; xspde(in);
```

Here *in.da* defines the derivative function, with z the noise.

- **What do you see if you average over** 10000 **trajectories ?**

```
in.ensembles = 10000;
xspde(in);
```

- **What do you see if you plot the mean square distance? Note that variances should increase linearly with t.**

```
in.observe = @(a,r) a.^2;
in.olabels = '<a^2>';
xspde(in);
```

- **What if you add a force that takes the particle back to the origin?**

$$\dot{a} = -a + w(t)\,, \tag{2.2}$$

```
in.da=@(a,w,r) -a+w
xspde(in);
```

Mathematical exercise:

- If you like mathematics; as an offline exercise, try to solve the equivalent diffusion equation for the probability P to find the variance $\langle a^2 \rangle$! The equation is:

$$\frac{\partial P\left(a\right)}{\partial t} = \left[\frac{\partial}{\partial a} + \frac{1}{2}\frac{\partial^2}{\partial a^2}\right] P\left(a\right)$$

2.4 Laser quantum noise

Next we treat a model for the quantum noise of a single mode laser as it turns on, near threshold:

$$\dot{a} = ga + bw(t) \tag{2.3}$$

where the noise is complex, $w = (w_1 + iw_2)$, so that:

$$\left\langle w(t)w^*(t')\right\rangle = 2\delta\left(t - t'\right)\,. \tag{2.4}$$

Here the coefficient b describes the quantum noise of the laser, and is inversely proportional to the equilibrium photon number.

Exercises

You should type *clear* first when starting new simulations.

- **Solve for the case of** $g = 0.1$**,** $b = 0.01$

Most lasers have more than 100 photons, and hence less noise!

```
clear
in.noises=2;
in.observe = @(a,r) abs(a)^2;
in.olabels = '|a|^2';
in.da=@(a,w,r) a+0.01*(w(1)+i*w(2));
xspde(in);
```

Consider the case where the laser saturates to a steady state:

$$\dot{a} = \left(1 - |a|^2\right) a + bw(t) \tag{2.5}$$

- **Solve for the saturated laser case**

You should get the output graph in Fig (2.2).

```
in.da=@(a,w,r) (1-abs(a)^2)*a+0.01*(w(1)+i*w(2));
xspde(in);
```

2.5 Spectra

Frequency spectra have many uses, especially for understanding the steady-state fluctuations of any physical system. Take the random walk equation in a complex space,

$$\frac{da}{dt} = -a + w(t)$$

where $w(t) = w_1(t) + iw_2(t)$. Use an initial condition of $a = (v_1 + iv_2)/\sqrt{2}$, with $\langle v_i^2 \rangle = 1$, to start near the steady state.

For sufficiently long time-intervals, the solution in frequency space - where $\omega = 2\pi f$ is the angular frequency - is given by:

$$\tilde{a}\left(\omega\right) = \frac{\tilde{\zeta}(\omega)}{1 + i\omega}$$

The expectation value of the noise spectrum, in the large T limit, is therefore:

$$\begin{aligned} \left\langle |\tilde{a}(\omega)|^2 \right\rangle &= \frac{1}{2\pi\left(1+\omega^2\right)} \int\int e^{-i\omega(t-t')} \left\langle \zeta(t)\zeta^*(t') \right\rangle dtdt' . \\ &= \frac{T}{\pi\left(1+\omega^2\right)} \end{aligned}$$

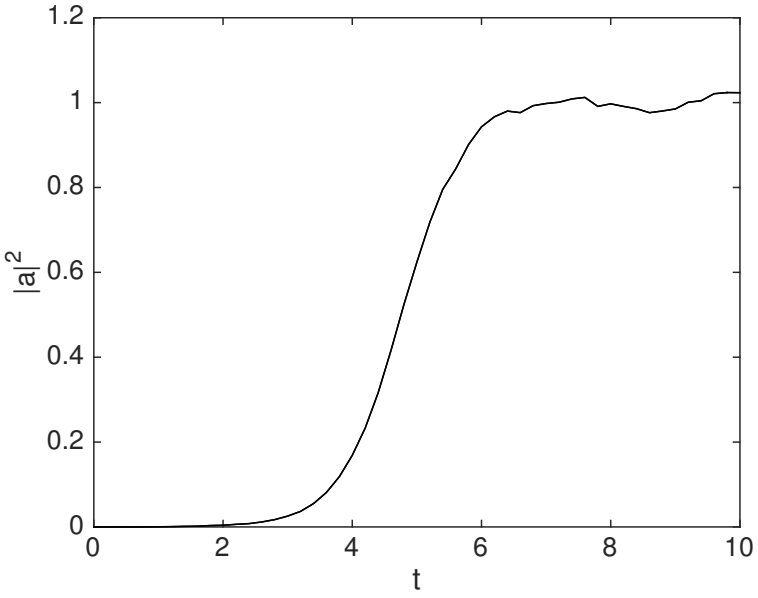

Figure 2.2: Simulation of the stochastic equation describing a laser turning on.

Exercises

- **Plot the spectrum over a range of $t = 100$, with 640 points and a random initial equation near the equilibrium value.**

The input parameters are given below. There are parallel operations here, for ensemble averaging, so **USE THE DOT**.

```
clear
in.points = 640;
in.ranges = 100;
in.noises = 2;
in.ensembles = 10000;
in.initial = @(v,r) (v(1,:)+1i*v(2,:))/sqrt(2);
in.da = @(a,w,r) -a + w(1,:)+1i*w(2,:);
in.observe =@(a,r) a.*conj(a);
in.transforms =1;
in.olabels = '|a(\omega)|^2';
xspde(in);
```

Note that `in.transforms =1` tells xSPDE to Fourier transform the field over the time coordinate before averaging, to give a spectrum. The first argument v of the *initial* function is a random field, used optionally to initialize the stochastic variable. You

can transform any field in any dimension, with any number of resulting averages. See Chapter 6 for details.

To define as many observables as you like, use a Matlab cell array;

```
in.observe{1} = ..;
in.observe{2} = ..;
```

- **Simulate over a range of $t = 200$. What changes do you see? Why?**
- **Change the equation to the laser noise equations. Why is the spectrum much narrower?**

2.6 Ito and Stratonovich

The xSPDE toolbox is mainly designed to treat Stratonovich equations [2], which are the broad-band limit of a finite band-width random noise equation. Another type of of stochastic equation is the Ito form, which is a limit where derivatives are evaluated at the start of each step. If you have an Ito equation, you can either use the Euler integration option, and set $in.step = xEuler$, or else apply the Stratonovich correction to the drift, as outlined below.

To emphasize the difference, an Ito equation is often written as a difference equation:

$$d\mathbf{a} = \mathbf{A}^I[\mathbf{a}] + \underline{\mathbf{B}}[\mathbf{a}] \cdot d\boldsymbol{w}(t). \tag{2.6}$$

Here $\langle dw_i(\boldsymbol{x})\, dw_j(\boldsymbol{x}')\rangle = \delta_{ij} dt$.

When $\mathbf{B}$ is constant, the two methods are identical. If $\mathbf{B}$ is not constant, the Ito drift term $\mathbf{A}^I$ is different to the corresponding Stratonovich one. The relationship is:

$$A_i = A_i^I - \frac{1}{2}\sum_{j,m} \frac{\partial B_{ij}}{\partial a_m} B_{mj}\,. \tag{2.7}$$

2.7 Financial calculus

A well-known Ito stochastic equation is the Black-Scholes equation, used to price financial options. It describes the fluctuations in a stock value:

$$da = \mu a\, dt + a\sigma\, dw, \tag{2.8}$$

where $\langle dw^2\rangle = dt$. Since the noise is multiplicative, the equation is different in Ito and Stratonovich forms of stochastic calculus.

The corresponding Stratonovich equation, as used in xSPDE for the standard default integration routine is:

$$\dot{a} = \left(\mu - \sigma^2/2\right) a + a\sigma w(t). \tag{2.9}$$

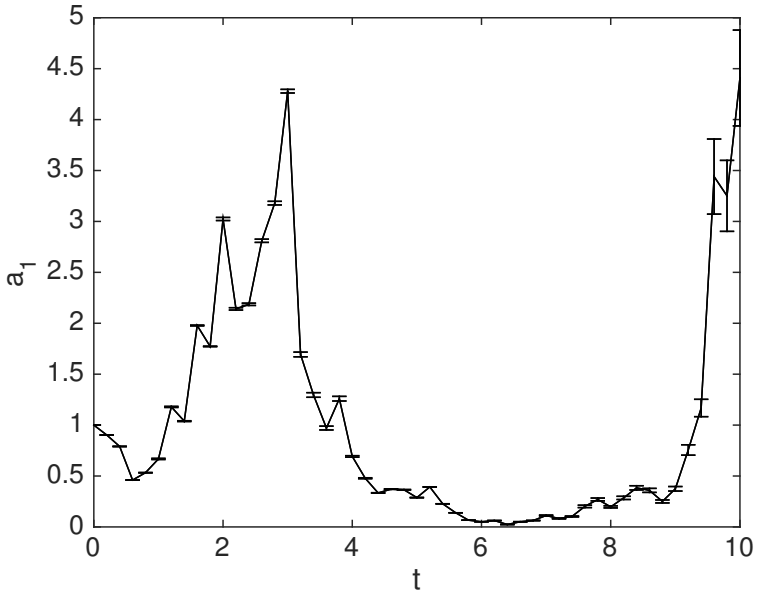

Figure 2.3: Simulation of the Black-Scholes equation describing stock prices.

Exercises

- **Solve for a startup with a volatile stock having $\mu = 0.1$, $\sigma = 1$.**

An interactive xSPDE script in Matlab is given below with an output graph in Fig (2.3), Note the spiky behaviour, typical of multiplicative noise, and also of the risky stocks in the small capitalization portions of the stock market.

```
clear
in.initial=@(v,r) 1
in.da=@(a,w,r) -0.4*a+a*w
xspde(in)
```

Note that *in.initial* describes the initialization function. The first argument of $@(v, r)$ is v, an initial random variable with unit variance. The error-bars are estimates of step-size error. Errors can be reduced by using more time-steps.

- **Solve for a more mature stock having $\mu = 0.1$, $\sigma = 0.1$.**

3 xSPDE Projects

Projects are larger scale investigations, where files are used to record inputs and data. This requires a new input parameter:

Label	Type	Default value	Description
in.file	string	”	Matlab or HDF5 data file

3.1 Ensembles and sampling errors

Averages over stochastic ensembles are the specialty of xSPDE, which requires specification of the ensemble size. A sophisticated hierarchy of ensemble specifications in three levels is possible, to allow maximum resource utilization, so that:

$$in.ensembles = [ensembles(1), ensembles(2), ensembles(3)] .$$

The local ensemble, *ensembles* (1), gives within-thread parallelism, allowing vector instruction use for single-core efficiency. The serial ensemble, *ensembles* (2), gives a number of independent trajectories calculated serially.

The parallel ensemble, *ensembles* (3), gives multi-core parallelism, and requires the Matlab parallel toolbox. This improves speed when there are multiple cores. One should optimally put *ensembles* (3) equal to the available number of CPU cores.

The *total* number of stochastic trajectories or samples is

$$ensembles(1) \times ensembles(2) \times ensembles(3) .$$

Either $ensembles(2)$ or $ensembles(3)$ are required if sampling error-bars are to be calculated, owing to the sub-ensemble averaging method used in xSPDE to calculate sampling errors accurately.

3.2 xSIM and xGRAPH

An XPDE session can either run simulations interactively, described in Chapter 2, or else using a function file called a project file. This allows xSPDE to run in a batch mode, as needed for longer projects which involve large ensembles. In either case, the Matlab path must include the xSPDE folder. For generating graphs automatically, the script input or project function should end with the combined function **xspde(in)**.

With batch jobs it is often useful to divide xSPDE into its simulation function, xSIM, and its graphics function, xGRAPH, to allow graphs to be made at a later time from

the simulation. In this case the function $xsim$ runs the simulation, and $xgraph$ makes the graphs. The two-stage option is better for running batch jobs, which you can graph at a later time.

To create a data file, you must enter the filename when running the simulation, using the $in.file = filename$ input. A typical xSPDE project function of this type is as follows:

$$\begin{aligned} \texttt{function}[e, ec] &= project.m \\ \texttt{in.}[label1] &= [parameter1] \\ \texttt{in.}[label2] &= \ldots \\ \texttt{in.file} &= [my\,file]\texttt{.mat} \\ [\texttt{e}, \texttt{in}] &= \texttt{xsim(in)} \\ \texttt{ec} &= \texttt{xgraph(in.file)} \\ \texttt{end} & \end{aligned}$$

After preparing a project file using the editor, click on the Run arrow above the editor window.

In summary, a simple batch job workflow is as follows:

- Create the metadata in, and include a file name
- Change the Matlab directory path to home using $cd\ \sim$.
- Run the simulation with **[e,in]=xsim(in)**.
- Note that xSIM changes the file-name if it already exists, and returns the new name for plotting.
- Run **xgraph(in.file)**, and the data will be accessed and graphed.
- You can use either Matlab (.mat) or standard HDF5 (.h5) file-types.

3.3 Kubo project

To get started on more complex programs, we next simulate the Kubo oscillator, which is an oscillator with a random frequency:

$$\dot{a} = iaw \tag{3.1}$$

Exercises

- **Simulate the Kubo oscillator using a file, $Kubo.m$, with two ensembles to allow sampling error estimates. The error vector $error$ gives the time-step error plus the sampling error.**

```
function [error] = Kubo()
in.name = 'Kubo oscillator';
in.ensembles = [400,16];
in.initial = @(v,r) 1+0*v;
in.da = @(a,w,r) i*a.*w;
in.olabels = '<a_1>';
in.file = 'kubo.mat';
[error,in]=xsim(in);
xgraph(in.file);
end
```

The initial value of 1 is modified by adding $0 * v$. This is necessary, so that it returns an array whose first dimension equals the ensemble size.

This function is designed to generate a data file, `kubo.mat`. If you run this more than once without deleting the earlier file, you will get a warning and a new file-name, `kubo1.mat`, which is stored in the in structure. Under these circumstance, xgraph will graph the data in the most recent file saved, with the new file-name. This occurs because xSPDE will **not** overwrite old files, to protect your previous data.

You can also include modified graphics parameters as a second input when running `xgraph,` just in case the first graphs you generate need changes.

3.4 Challenge problem #1

This is a much harder example, involving a full nonlinear quantum phase-space simulation. The method can also be used to investigate quantum non-equilibrium phase transitions, tunneling in open systems, quantum entanglement, Einstein-Podolsky-Rosen paradoxes, Bell violations, and many other problems treated in the literature[2, 3].

The following new ideas are introduced:

1. `in.ranges` **is the space-time integration range.**
2. `in.steps` **gives integration steps per plot-point, for accuracy.**

A simple case is the nonlinear driven quantum oscillator - for example, an optomechanical, atomic or nonlinear optical medium in a driven cavity.

The phase-space Fokker-Planck equation is equivalent to an SDE with the following form

$$\begin{aligned} \frac{d\alpha}{dt} &= -\tilde{\gamma}\alpha - i\kappa\alpha^2\beta + \mathcal{E} + i\sqrt{i\kappa}\alpha w_1(t), \\ \frac{d\beta}{dt} &= -\tilde{\gamma}^*\beta + i\kappa\beta^2\alpha + \mathcal{E}^* + \sqrt{i\kappa}\beta w_2(t), \end{aligned} \tag{3.2}$$

where $\xi_i(t)$ are independent Gaussian noise terms with zero means and the following nonzero correlations

$$\langle w_i(t)w_j(t')\rangle = \delta_{ij}\delta(t-t'). \tag{3.3}$$

Note that these equations require that $\kappa \ll \gamma$, which is typical experimentally, otherwise one must use additional stochastic gauge stabilization to improve sampling error.

Defining the dimensionless parameters ε and λ as

$$\varepsilon = \frac{2\mathcal{E}}{\kappa}\,,\ \lambda = \frac{2\tilde{\gamma}}{\kappa}\ . \tag{3.4}$$

Provided $\gamma + \Delta\omega < 0$ and $(\mathrm{Re}\lambda)^2 > 3|\lambda|^2$ there is a bistable region. This means that for $|\varepsilon|^2$ in a certain range, there are two physically possible values of n_0. For $\tilde{\gamma} = 2.5 - 10i$, $\kappa = 1$, the bistable region is around $\mathcal{E} = 8$. The observed quantity is the particle number, $\left\langle \hat{a}^\dagger \hat{a} \right\rangle = \langle \alpha\beta \rangle$, with stable values of $n \approx 1$ and $n \approx 6$.

Exercises

- **Simulate the nonlinear oscillator by creating a file,** *Nonlinear.m*
- **Can you observe quantum tunneling of $n = \alpha\beta$ in the bistable regime?**
- **Do you see transient Schroedinger 'cat states' with a negative n value?**

Since tunneling is random, for real experimental comparisons, one would have to measure correlation functions and spectra. However, a tunneling event in a simulation indicates that it is likely in experiment too. These calculations require long time scales, `in.ranges`, to observe tunneling, and a large number of time steps per plotted point, `in.steps`, to maintain accuracy in the quantum simulations.

4 Stochastic PDEs

A stochastic partial differential equation or $SPDE$ for a complex vector field is defined in both time t and space dimensions$\boldsymbol{x}$. The total dimension d in xSPDE is unlimited, and includes time and space. Large d is memory-intensive and slow! The general equation solved can be written in differential form as

$$\frac{\partial \mathbf{a}}{\partial t} = \mathbf{A}\left[\mathbf{a}\right] + \underline{\mathbf{B}}\left[\mathbf{a}\right] \cdot \boldsymbol{w}(t) + \underline{\mathbf{L}}\left[\boldsymbol{\nabla}\right] \cdot \mathbf{a}\,. \tag{4.1}$$

Here $\mathbf{a}$ is a real or complex vector or vector field. The initial conditions are arbitrary functions. $\mathbf{A}\left[\mathbf{a}\right]$ and $\underline{\mathbf{B}}\left[\mathbf{a}\right]$ are vector and matrix functions of $\mathbf{a}$, $\underline{\mathbf{L}}\left[\boldsymbol{\nabla}\right]$ is a matrix of linear terms and derivatives, diagonal in the vector indices, and $\mathbf{w} = \left[\boldsymbol{w}^x, \boldsymbol{w}^k\right]$ are real delta-correlated noise fields such that:

$$\begin{aligned} \left\langle w_i^x\left(t,\boldsymbol{x}\right) w_j^x\left(t,\boldsymbol{x}'\right)\right\rangle &= \delta\left(\boldsymbol{x}-\boldsymbol{x}'\right)\delta\left(t-t'\right)\delta_{ij} \\ \left\langle w_i^k\left(t,\boldsymbol{k}\right) w_j^k\left(t,\boldsymbol{k}'\right)\right\rangle &= f(\mathbf{k})\delta\left(\boldsymbol{k}-\boldsymbol{k}'\right)\delta\left(t-t'\right)\delta_{ij}. \end{aligned} \tag{4.2}$$

Transverse boundary conditions are assumed periodic. The term $\underline{\mathbf{L}}\left[\boldsymbol{\nabla}\right]$ may be omitted if $d = 1$, as there are no space dimensions. Here $f(\mathbf{k})$ is an arbitrary momentum filter, for correlated noise. The linear function L is input as a separate function in xSPDE, to allow for greatest efficiency and use of an interaction picture. Note that it depends on momentum space coordinates, and this involves Fourier transforms.

4.1 Stochastic Ginzburg-Landau equation

Including space-time dimensions with $d = 3$, an example of a SPDE is the stochastic Ginzburg-Landau equation. This describes symmetry breaking: the system develops a spontaneous phase which can vary spatially as well. The model is widely used to describe lasers, magnetism, superconductivity, superfluidity and even particle physics:

$$\dot{a} = \left(1 - |a|^2\right) a + bw(t) + c\nabla^2 a \tag{4.3}$$

where

$$\left\langle w(x)w^*(x')\right\rangle = 2\delta\left(t-t'\right)\delta\left(x-x'\right). \tag{4.4}$$

The following new ideas are introduced for this problem:

1. `in.dimension` **is the space-time dimension.**
2. **The** 'dot' **notation used for parallel operations over lattices**.

3. `in.linear` **is the linear operator - a laplacian in these cases.**
4. `in.images` **produces movie-style images at discrete time slices.**
5. `r.Dx` **indicates a derivative operation,** $\partial/\partial x$**.**
6. $-5 < x < 5$ **is the default xSPDE coordinate range in space.**

Exercises

- **Solve the stochastic G-L equation for** $b = 0.001$ **and** $c = 0.01i$**.**
- **Change to a real diffusion so that** $c = 0.1$**.**

In the first case, you should get the output graphed in Fig (4.1) .

```
in.noises=2;
in.dimension=3;
in.steps=10;
in.linear = @(r) i*0.01*(r.Dx.^2+r.Dy.^2);
in.observe = @(a,~) abs(a).^2;
in.images =6;
in.olabels = '|a|^2';
in.da=@(a,w,~) (1-abs(a(1,:)).^2).*a+0.001*(w(1,:)+i*w(2,:));
xspde(in)
```

Here the notation $z(1,:)$ is like the dot: the colon means that the operation is repeated over all values of the second index, which is the spatial lattice index.

4.2 Soliton

The famous nonlinear Schrödinger equation (NLSE) is:

$$\frac{da}{dt} = \frac{i}{2}\left[\nabla^2 a - a\right] + ia\left|a\right|^2$$

Together with the initial condition that $a(0,x) = sech(x)$, this has a soliton, an exact solution that doesn't change in time:

$$a(t,x) \quad = \quad sech(x)$$

The Fourier transform at $k = 0$ is simply:

$$\tilde{a}(t,0) \quad = \quad \frac{1}{\sqrt{2\pi}}\int sech(x)dx = \sqrt{\frac{\pi}{2}}$$

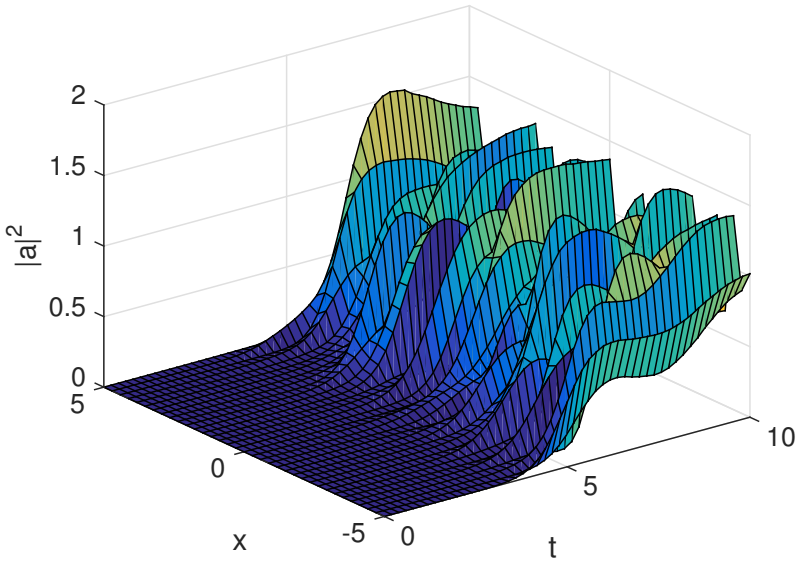

Figure 4.1: Simulation of the stochastic equation describing symmetry breaking in two dimensions. Spatial fluctuations are caused by the different phase-domains that interfere. The graph obtained here is projected onto the $y = 0$ plane.

Exercises

- **Solve the NLSE for a soliton**

```
function [e] = Soliton()
in.name = 'NLS soliton';
in.dimension = 2;
in.initial = @(v,r) sech(r.x);
in.da = @(a,~,r) i*a.*(conj(a).*a);
in.linear = @(r) 0.5*i*(r.Dx.^2-1.0);
in.olabels = {'a_1(x)'};
in.compare{1}= @(t,~) 1;
e = xspde(in);
end
```

- **Add an additive complex noise of $0.01(w_1 + iw_2)$ to the differential equation, then replot with an average over 1000 samples.**

4.3 Gaussian diffraction

Free diffraction of a Gaussian wave-function in three dimensions, is given by

$$\frac{da}{dt} = \frac{i}{2}\nabla^2 a$$

Together with the initial condition that $a(0,x) = exp(-|\mathbf{x}|^2/2)$, this has an exact solution for the diffracted intensity in either ordinary space or momentum space:

$$\begin{aligned} |a(t,\mathbf{x})|^2 &= \frac{1}{(1+t^2)^{3/2}} exp(-|\mathbf{x}|^2 / \left(1+t^2\right)) \\ |\tilde{a}(t,\mathbf{k})|^2 &= exp(-|\mathbf{k}|^2) \end{aligned}$$

Exercises

- **Solve the diffraction equation without noise**
- Note that the example below stores data in a standardized HDF5 file.

```
function [e] = Gaussian()
in.dimension = 4;
in.initial = @(v,r) exp(-0.5*(r.x.^2+r.y.^2+r.z.^2));
in.linear = @(r) 1i*0.05*(r.Dx.^2+r.Dy.^2+r.Dz.^2);
in.observe = @(a,r) a.*conj(a);
in.olabels = '|a(x)|^2';
in.file = 'Gaussian.h5';
in.images = 4;
[e,in] = xsim(in);
e = xgraph(in.file);
end
```

- **Add an additive complex noise of $0.01(w_1 + iw_2)$ to the Gaussian differential equation, then replot with an average over 10 samples.**

Note that for this, you'll need to define $in.da = @(a,w,r)0.01 * (w(1,:) + i * w(2,:))$

4.4 Planar noise

The next example is growth of thermal noise of a two-component complex field in a plane, given by the equation

$$\frac{d\boldsymbol{a}}{dt} = \frac{i}{2}\nabla^2\boldsymbol{a} + \boldsymbol{w}(t,x)$$

where $\boldsymbol{\zeta}$ is a delta-correlated complex noise vector field:

$$w_j(t,\mathbf{x}) = \left[w_j^{re}(t,\mathbf{x}) + i\zeta_j^{im}(t,\mathbf{x})\right]/\sqrt{2},$$

with the initial condition that the initial noise is delta-correlated in position space

$$a(0,\mathbf{x}) = \boldsymbol{\zeta}^{(in)}(\boldsymbol{x})$$

where:

$$\boldsymbol{\zeta}^{(in)}(\boldsymbol{x}) = \left[\boldsymbol{\zeta}^{re(in)}(\mathbf{x}) + i\boldsymbol{\zeta}^{im(in)}(\mathbf{x})\right]/\sqrt{2}$$

This has an exact solution for the noise intensity in either ordinary space or momentum space:

$$\begin{aligned}
\left\langle \left|a_j\left(t,\mathbf{x}\right)\right|^2\right\rangle &= (1+t)/\Delta V \\
\left\langle \left|\tilde{a}_j\left(t,\mathbf{k}\right)\right|^2\right\rangle &= (1+t)/\Delta V_k \\
\left\langle \tilde{a}_1\left(t,\mathbf{k}\right)\tilde{a}_2^*\left(t,\mathbf{k}\right)\right\rangle &= 0
\end{aligned}$$

Here, the noise is delta-correlated, and ΔV, ΔV_k are the cartesian space and momentum space lattice cell volumes respectively. Suppose that $N = N_x N_y$ is the total number of spatial points, and $V = R_x R_y$, where there are $N_{x(y)}$ points in the x(y)-direction, with a total range of $R_{x(y)}$. Then, $\Delta x = R_x/N_x$,$\Delta k_x = 2\pi/R_x$, so that:

$$\begin{aligned}
\Delta V &= \Delta x \Delta y = \frac{V}{N} \\
\Delta V_k &= \Delta k_x \Delta k_y = \frac{(2\pi)^2}{V}.
\end{aligned}$$

In the simulations, two planar noise fields are propagated, one using delta-correlated noise, the other with noise transformed to momentum space to allow filtering. This allows use of finite correlation lengths when needed, by including a frequency filter function that is used to multiply the noise in Fourier-space. The Fourier-space noise variance is the square of the filter function.

The first noise index, $in.noises(1)$, indicates how many noise fields are generated, while $in.noises(2)$ indicates how many of these are spatially correlated, via fourier-transform, filter and inverse fourier transform. These appear to the user as additional noises, so the total is $in.noises(1) + in.noises(2)$. The filtered noises have a finite correlation length. They are correlated with the first $in.noises(1)$ x-space noises they are generated from, as this can be useful.

Exercises

- **Solve the planar noise growth equation**

```
function [e] = Planar()
in.name = 'Planar noise growth';
in.dimension = 3;
in.fields = 2;
in.ranges = [1,5,5];
in.steps = 2;
in.noises = [4,2];
in.ensembles = [10,4,4];
in.initial = @Initial;
in.da = @Da;
in.linear = @Linear;
in.observe = @(a,r) a(1,:).*conj(a(1,:));
in.olabels= '<|a_1(x)|^2>';
in.compare = @(t,in) [1+t]/in.dV;
in.images = 4;
e = xspde(in);
end

function a0 = Initial(v,r)
a0(1,:)  = (v(5,:)+1i*v(6,:))/sqrt(2);
a0(2,:)  = (v(3,:)+1i*v(4,:))/sqrt(2);
end

function da = Da(a,z,r)
da(1,:)  = (z(5,:)+1i*z(6,:))/sqrt(2);
da(2,:)  = (z(3,:)+1i*z(4,:))/sqrt(2);
end

function L = Linear(r)
lap = r.Dx.^2+r.Dy.^2;
L(1,:)  = 1i*0.5*lap(:);
L(2,:)  = 1i*0.5*lap(:);
end
```

- **Add a decay rate of $-a$ to the differential equation, then replot**
- **Add growth and nonlinear saturation terms**

4.5 Characteristic equation

The next example is the characteristic equation for a traveling wave at constant velocity. It is included to illustrate what happens at periodic boundaries, when Fourier-transform methods are used for propagation. There are a number of methods known to prevent this effect, including addition of absorbers - often called apodization - at the boundaries. The equation is:

$$\frac{da}{dt} + \frac{da}{dx} = 0$$

Together with the initial condition that $a(0, x) = sech(2x + 5)$, this has an exact solution that propagates at a constant velocity:

$$a(t, x) \quad = \quad sech(2(x - t) + 5)$$

The time evolution at $x = 0$ is simply:

$$a(t, 0) \quad = \quad sech(2(t - 5/2))$$

Exercises

- **Solve the characteristic equation given above, noting the effects of periodic boundaries.**

```
function [e] = Characteristic()
in.name = 'Characteristic'
in.dimension = 2;
in.initial = @(v,r) sech(2.*(r.x+2.5));
in.da = @(a,z,r) 0*a;
in.linear = @(r) -r.Dx;
in.olabels = 'a_1(x)';
in.compare = @(t,in) sech(2.*(t-2.5));
e = xspde(in);
end
```

- **Recalculate with the opposite velocity, and a new exact solution.**

4.6 Challenge problem #2

Counter-intuitively, a random potential prevents normal wave-packet spreading in quantum-mechanics. This is Anderson localization: a famous property of quantum mechanics in a random potential. A typical experimental method is to confine an ultra-cold Bose-Einstein condensate (BEC) in a trap, then release the BEC in a random external potential produced by a laser. The expansion rate of the BEC is reduced by the Anderson

localization due to the random potential. Physically, the observable quantity is the particle density $n = |\psi|^2$.

This can be treated either using a Schroedinger equation with a random potential, at low density, or using the Gross-Pitaevskii (GP) equation to include atom-atom interactions at the mean field level. In this example of a problem where strong localization occurs, the general equations are:

$$\frac{\partial \psi}{\partial t} = \frac{1}{i\hbar}\left[-\frac{\hbar^2}{2m}\nabla^2 + V\left(\boldsymbol{r}\right) + g\left|\psi\right|^2\right]\psi$$

In calculations, it is best to use a dimensionless form by rescaling coordinates and fields. A simple way to simulate this with xSPDE is to treat ψ as a scaled field $a(1)$, and to assume the random potential field $V\left(\boldsymbol{r}\right)$ as caused by interactions with second random field $|a(2)|^2$. This has the advantage that it is similar to the actual experiment, and allows one to treat time-dependent potentials as well, if desired.

With the rescaling, this simplifies to:

$$\frac{\partial a_1}{\partial \tau} = i\left[\frac{\partial}{\partial \zeta^2}^2 - |a_2|^2 - |a_1|^2\right]a_1$$

A convenient initial condition is to use:

$$\begin{aligned} a_1 &= a_0 \exp(-\zeta^2) \\ \left\langle a_2(\zeta) a_2(\zeta')\right\rangle &= v\delta\left(\zeta - \zeta'\right) \end{aligned}$$

Exercise

- **Solve Schroedinger's equation without a random potential, to observe expansion.**
- **Include a random potential v, to observe localization.**
- **Experiment with nonlinear terms and higher dimensions.**

Note that the GP equation is a mean field approximation; this is still not a full solution of the many-body problem! Also, the experiments are somewhat more complicated than this, and actually observe the momentum distribution.

5 Logic and data

The simulation program logic is straightforward. It is a very compact function called **xspde**. This calls **xsim**, for the simulation, then **xgraph** for the graphics. Most of the work is done by other specialized functions. Input parameters come from an **input** array, output is saved either in a **data** array, or else in a specified file. During the simulation, global averages and error-bars are calculated for both time-step and sampling errors. When completed, timing and error results are printed.

5.1 Simulation function, xsim

5.1.1 Input and data structures

To explain xSPDE in full detail,

- Simulation inputs are stored in the **input** cell array.
- This describes a *sequence* of simulations, so that **input={in1,in2,...}**.
- Each structure **in** describes a simulation, whose output is the input of the next.
- The main simulation function is called using **xsim**(**input**).
- Averages are recorded sequentially in the **data** cell array.
- Raw trajectory data is stored in the **raw** cell array if required.

The sequence **input** has a number of individual simulation objects **in**. Each includes parameters that specify the simulation, with functions that give the equations and observables. If there is only one simulation, just one individual specification **in** is needed. In addition, xSPDE generates graphs with its own graphics program.

5.1.2 User definable simulation functions

The most commonly used functions are indicated with an asterisk. Others have automatic defaults that are not usually changed, except in special cases.

initial** is used to initialize each integration in time. This is a user-defined function, which can involve random numbers if there is an initial probability distribution. This creates a stochastic field on the lattice, called ***a. The default is **xinitialise**, which sets fields to zero.

***observe** is the initial observation function whose output is averaged over the ensembles, called from **xpath**. The default, **xobserve**, returns the amplitudes.

***function** is an optional function used to take arbitrary transforms or time integrals of the initial observe outputs, after averaging over the first ensemble. The default returns the observe outputs unchanged.

***linear** is the linear response, including transverse derivatives in space. The default, **xlinear**, sets this to zero.

***da** is called by **step** to calculate derivatives at every step in the process, including the stochastic terms. The default, **xstep**, sets this to zero

transfer is a function used to initialize sequential simulations, where previous field values are included as input to the function. The default, **xtransfer**, replicates the output of the previous simulation.

step is the algorithm that computes each step in time. This also generates the random numbers at each time-step. Options include **xEuler**, **xRK2**, **xRK4,** for Euler and Runge-Kutta integration methods. This can be user-modified by initializing the handle in.step to add a new integration function. The default is **xRK4**, which is widely applicable. Another good option is**xMP**, which uses a midpoint or central-difference technique in the interaction picture.

rfilter returns a momentum filter for random initial fields generated in momentum space. The default, **xrfilter**, sets this to unity.

nfilter returns a momentum filter for random noises generated in momentum space. The default, **xnfilter**, sets this to unity.

grid calculates the spatial integration grid. The default, **xgrid**, returns a rectangular grid in ordinary and momentum space.

prop is the spatial propagation function. The default, **xprop**, takes a Fourier transform of $\boldsymbol{a}$, multiplies by propfactor to propagate in time, then takes an inverse Fourier transform.

randomgen generates the initial random fields. The default, **xrandomgen**, returns Gaussian real fields that are delta-correlated in space or momentum space.

noisegen generates the initial random fields. The default, **xnoisegen**, returns Gaussian real noises that are delta-correlated in time and also in space or momentum space.

propfactor returns the interaction picture propagation factors. The default, **xpropfactor**, uses data from the **linear** function to calculate this.

5.2 Data arrays and errors

5.2.1 Data arrays

All calculated data, including fields, observables and graphics results, is stored in arrays of implicit or explicit rank $2 + dimension$. The first index is a field index (i), the second a statistics/errors index (j), while the remaining indices $k \equiv k_1, \ldots k_D \equiv k_1, \boldsymbol{k}$ are for time and space. The space-time dimension D is unlimited. When the fields, noises or coordinates are integrated by the xSPDE integration functions, they are flattened to a matrix. The first index is the field index, and the combined second index covers all the rest.

There are several different types of arrays used. These are as follows:

- Field arrays $a(i, j, 1, \boldsymbol{k})$ - these have a statistics index of up to $j = ensembles(1)$, but just a single present time-point for efficiency.
- Random and noise arrays $w(i, j, 1, \boldsymbol{k})$ - these are like field arrays, except that they contain random numbers for the stochastic equations.
- Coordinate arrays $x\{l\}(1, j, 1, \boldsymbol{k})$- these store the values of coordinates at grid-points, depending on the axis label $l = 2, \ldots D$.
- Raw arrays $r\{e, s\}(i, j, k)$ - like fields, but with all points stored. Use with care, as they take up large amounts of memory. Note that when output or saved, these have additional cell indices $e = 1, 2$ for the error-checking time-step and $s = 1, \ldots S$ for the sequence number.
- Data arrays $d\{n\}(i, j, k)$ - these store the averages, or arbitrary functions of them, with a statistics index $j = 1, 2, 3$, to store error data at all time points.
- Graphics arrays $g\{n\}(i, j, k)$ - these store the data that is plotted, and can include further functional transformations if required.

The field index i in a graphics or data array is intended to describe different lines on a graph. There can be quite different first dimensions between fields, noises and output data, as they are specified using different input parameters. If only a single output graph is wanted, the *observe* cell index is not needed.

Outputs have an extra high-level cell index $\{n\}$ called the graph or function index. This corresponds to the index $\{n\}$ of the observe function used to generate averages. One can have several data arrays in a larger cell arrays to make a number of distinct output graphs labelled n, each with multiple averages.

5.2.2 Step-size errors

Errors caused by the finite time-domain step-size, as well as those caused by the finite ensemble, are checked automatically. Those caused by the spatial lattice are not checked in the xSIM code. They must be checked by manually, by comparing results with different transverse lattice ranges and step-size.

Errors due to a finite step-size are estimated by comparing two simulations with different step-sizes and the same random sequence, to make sure the final results are accurate. The final 2D output graphs will have error-bars if $in.errorchecks = 2$ was specified, which is also the default parameter setting. Error-bars below a minimum relative size compared to the vertical range of the plot, specified by the graphics variable *minbar*, are not plotted.

There is a clear strategy if the errors are too large. Either increase the *points*, which gives more plotted points and lower errors, or increase the *steps,* which reduces the step size without changing the graphical resolution. The default algorithm and extrapolation order can also be changed, please read the full xSPDE manual when doing this. Error bars on the graphs can be removed by setting $errorchecks = 1$ or increasing *minbar* in final graphs.

5.2.3 Sampling errors

Sampling error estimation in xSIM uses sub-ensemble averaging. This generally leads to more reliable sampling error estimates, and makes efficient use of the vector instruction sets that are used by Matlab. Ensembles are specified in three levels. The first, *ensemble(1)*, is called the number of samples for brevity. All computed quantities returned by the **observe** functions are first averaged over the samples, which are calculated efficiently using a parallel vector of trajectories. By the central limit theorem, these low-level sample averages are distributed as a normal distribution at large sample number.

Next, the sample averages are averaged **again** over the two higher level ensembles, if specified. This time, the variance is accumulated. The variance of these distributions is used to estimate a standard deviation in the mean, since each computed quantity is now a normally distributed result. This method is applied to all the *graphs* observables. The two lines generated represent $\bar{o} \pm \sigma$, where o is the observe function output, and σ is the standard deviation in the mean.

The highest level ensemble, *ensemble(3),* is used for parallel simulations. This requires the Matlab parallel toolbox. Either type of high-level ensemble, or both together, can be used to calculate sampling errors.

Note that one standard deviation is not a strong bound; errors are expected to exceed this value in 32% of observed measurements. Another point to remember is that stochastic errors are often correlated, so that a group of points may all have similar errors due to statistical sampling.

If $ensembles(2) > 1$ or $ensembles(3) > 1$, which allows xSPDE to calculate sampling errors, it will plot upper and lower limits of one standard deviation. If the sampling errors are too large, try increasing $ensembles(1)$, which increases the trajectories in a single thread. An alternative is to increase $ensembles(2)$. This is slower, but is only limited by the compute time, or else to increase $ensembles(3)$, which gives higher level parallelization. Each is limited in different ways; the first by memory, and the second by time, the third by the number of available cores. Sampling error control helps ensures accuracy.

5.3 Graphics function, xgraph

The main functions involved for graphics are:

xgraph is called by xSPDE when the ensemble loops finished. The results are graphed and output if required.

5.3.1 User definable graphics functions

It is possible to simply run **xgraph** as is, without much intervention. However, there are many customization options, including two user defined functions which have useful applications. These are as follows:

***function(o,r)** is is a cell array of functions whose output **xgraph** is designed to plot. There is one function for each multidimensional dataset that is plotted. The default value is just the average stored in the data array generated by the simulation observe function, provided these are scalar observables.

An arbitrary number of functions of these observables can also be plotted, including vector observables. The input to functions is the observed data averages, stored in a cell array of size $o\{ngraphs\}(points(1), nspace, ofields)$. Here $ngraphs$ is the number of observable averaging functions, $points(1)$ is the number of time points,$nspace$ the number of space points, $ofields$ the vector dimension of the averaging or observable function. This can change depending on the observable.

***compare(r)** is a cell array of functions used to obtain comparison results. These are calculated from the user-specified **in.compare** handle, giving a dashed line on the generated time-domain graphs, and an error summary which is printed.

The results from the two graphics functions plotted using the **xgraph** routines.

5.3.2 xGRAPH outputs

This routine is prewritten to cover a range of useful graphs, but can be modified to suit the user.

The code is intended to cascade down from higher to lower dimension, generating different types of user-defined graphs. Each type of graph is generated once for each specified graphics function. The graphics axes that are used for plotting, and the points plotted, are defined using the optional ***axes*** input parameters, where $axes\{n\}$ indicates the n-th specified graphics function or set of generated graphical data.

If there are no ***axes*** inputs, or the inputs are zero - for example, $axes\{1\} = \{0, 0, 0\}$ - only the lowest dimensions are plotted, up to 3. If the ***axes*** inputs project out a single point in a given dimension, - for example, $axes\{1\} = \{0, 31, -1, 0\}$, these axes are suppressed in the plots. This reduces the effective dimension of the data - in this case to two dimensions.

Examples:

- $axes\{1\} = \{0\}$ - For function 1, plot all the time points; higher dimensions get defaults.
- $axes\{2\} = \{-1, 0\}$ - For function 2, plot the maximum time (the default), and all x-points.
- $axes\{3\} = \{1 : 4 : 51, 32, 64\}$ - For function 3, plot every 4-th time point at x point 32, y point 64
- $axes\{4\} = \{0, 1 : 4 : 51, 0\}$ - For function 4, plot every time point, every 4-th x point, and al the y-points.

Note that -1 indicates a default 'typical' point, which is the last point on the time axis, and the midpoint on the other axes.

The *pdimension* input can also be used to reduce dimensionality, as this sets the maximum effective plotted dimension. For example, $pdimension\{1\} = 1$ means that only plots vs time are output for the first function plotted. Note that in the following, t, x, y, z may be replaced by corresponding higher dimensions if there are axes that are suppressed. Slices can be taken at any desired point, not just the midpoint. Using the standard notation of, for example, $axes\{1\} = \{6 : 3 : 81\}$, can be used to modify the starting, interval, and finishing points for complete control on the plot points.

The graphics results depend on the resulting **effective** dimension. Since the plot has to include a data axis, the plot will be typically one dimension higher than the underlying space-time lattice:

dimension=4.. For higher lattice dimensions, only a slice through the chosen point, or the default midpoint, is available, reducing the lattice dimension to 3.

dimension=3 For three lattice dimensions, if $images > 1$, a movie of distinct 3D graphic images of observable *vs x,y* are plotted as *images* slices - usually in time. Otherwise, only a slice through the chosen point, or the default midpoint, is available for the highest dimension, reducing the lattice dimension to 2.

dimension=2 For two lattice dimensions, one 3D image of observable *vs x,t* is plotted. A movie of distinct 2D graphic plots is also possible. Otherwise, only a slice through $x = 0$ is available, reducing the lattice dimension to 1.

dimension=1 For one lattice dimension, a 2D plot of observable *vs* t is plotted, with data at each lattice point in time. Exact results, error bars and sampling error bounds are included if available.

In addition to time-dependent graphs, the **xgraph** function can generate *images* (3D) and *transverse* (2D) plots at specified points in time, up to a maximum given by the number of time points specified. The number of these can be individually specified for each graphics output. The images available are specified in *imagetype*: 3D perspective plots, grey-scale colour plots and contour plots.

Note that error bars and sampling errors are only graphed for 2D plots, typically of results vs time. The error-bars are not plotted when they are below a user-specified size, to improve graphics quality. Higher dimensional graphs do not include this, for visibility reasons, but they are still recorded in the data files.

5.4 Object architecture

5.4.1 xSPDE data objects

The xSPDE data objects include parameters and methods, but with a very open type of object-oriented architecture to allow extensibility. All xSPDE functions are modular and easily replaceable. In many cases this is as easy as just defining a new function handle to replace the default value. To define your own integration function, include in the xSPDE input the line:

```
in.step=@Mystep;
```

Next, include anywhere on your Matlab path the function definition, for example:

```
function a = Mystep(a,w,r)
% a = Mystep(a,w,r) propagates a step my way.
..
a = ...;
end
```

5.4.2 xSPDE hints

- When using xSPDE, it is a good idea to run the batch test script, Batchtest.m.
- Batchtest.m tests your parallel toolbox installation. If you have no license for this, omit the third ensemble setting.
- To create a project file, it is often easiest to start with an existing project function with a similar equation: see Examples folder.
- Graphics parameters can also be included in the structure **in**, to modify graphs.
- Comparison functions can be included if you want to compare with analytic results.
- A full list of functionality is listed next.

6 xSPDE metadata

The input data, which we label *input*, is a sequence of data structures that we label *in*. All of the input data is later passed to the *xgraph* function. The input data can be numbers, vectors, strings, functions and cell arrays, which just lists of data enclosed in curly brackets, {}. All xSPDE metadata has preferred values, so only changes from the preferences need to be input. The resulting combined input including preferred values, is stored internally as a sequence of structures in a cell array to describe the simulation sequence.

Simulation metadata, including all preferred default values that were used in a particular simulation, is also stored in the xSPDE output files. This is done in both the Matlab *.mat* and the HDF5 *.h*5 output files, so the entire simulation can be easily reconstructed or changed. Note that in the current version of xSPDE, one must use a Matlab formatted *.mat* file to store raw data which includes all stochastic trajectories. Either the Matlab or the HDF5 files can be used to store both input and averaged output data.

Some conventions are used to simplify inputs, as follows:

- **All input data has default values**
- **Vector inputs of numbers are enclosed in square brackets, [...].**
- **Where multiple inputs of strings, functions or vectors are needed they should be enclosed in curly brackets, {...}, to create a cell array.**
- **Vector or cell array inputs with only one member don't require brackets.**
- **Incomplete or partial vector or cell array inputs are filled in with the last applicable default value.**
- **New function definitions can be handles pointing elsewhere.**

6.1 Simulation data

The simulation data of the *in* structure is passed to the *xsim* program, to define the simulation details. User application constants can be included, but must not be reserved names. However, names starting with '*in.c*' or any capital letter like '*in.*A...' are always available to users of xSPDE. Note that use of globals is **not** recommended, as it is incompatible with the Matlab parallel toolbox.

6.1.1 Simulation parameters

All simulation data has default values as distributed, which are also user-modifiable through the *xpreferences* function. Any defaults can be checked by typing $in.print = 2$, then running *xsim(in)* or *xspde(in).*

Label	Default value	Description
version	*'xSIM1.22'*	Current version number
name	*"*	Simulation name
dimension	1	Space-time dimension
fields	1	Number of stochastic fields
ranges	10	Range of coordinates in [t,x,y,z,..]
origin	0	Origin of coordinates in [t,x,y,z,..]
points	51	Output lattice points in [t,x,y,z,..]
noises	*[1 0]*	Number of noise fields in [x,k]
randoms	*[1 0]*	Initial random fields in [x,k]
ensembles	*[1 1 1]*	Ensembles used for averaging
steps	*1*	Integration steps per output point
iterations	*4*	Maximum iterations
order	*1*	Extrapolation order
errorchecks	*2*	Number of error-checking cycles
seed	*0*	Seed for random number generator
file	*"*	File-name: *'f.mat'* = Matlab, *'f.h5'* = HDF5
ipsteps	*2*	IP transforms per time-step
graphs	*1*	Number of observables
print	1	Print: 1 for medium, 2 for high output
c	-	User specified static parameters
olabels	*{'a_1',..}*	Observable labels
averages	1	Maximum number of observables
transforms	*{[0 0 0 0],..}*	Fourier transforms in [t,x,y,z,..]
raw	0	Raw data switch: 1 for raw output
numberaxis	0	Numbered axis switch: 1 for numbered
octave	*0*	Force octave syntax: 1 for octave

6.1.2 Common functions

The most common functions, output field dimensions and calling arguments, with user-definable functions marked using an asterisk, are:

Label	Arguments	Purpose
**da*	*(a,z,r)*	Stochastic derivative
**initial*	*(v,r)*	Function to initialize fields
**linear*	(r)	Linear derivative function
**rfilter*	(r)	Random input filter in k-space
**nfilter*	(r)	Noise filter function in k-space
**transfer*	*(v,r,a0,r0)*	Initialization in a sequence
**step*	(a,z,dt,r)	Calculation of a time-step
**grid*	(r)	Grid calculator for lattice
**prop*	(a,r)	Interaction picture propagator
**propfactor*	(nc,r)	Propagator array calculation
**observe*	*(a,r)*	Observable function cell array
**function*	{@(av,~)av{1}(:,:,:).^2,..}	Functions stored of averages
**compare*	(t,in)	Comparison function cell array
randomgen	(r)	Initial random generator
noisegen	(r)	Noise generator
xint	(a,dx,r)	Integrates over a spatial lattice
xave	(a,av,r)	Averages over a spatial lattice
xd	(a,D,r)	Takes a spatial derivative

6.2 Internal data

Internally, xSPDE data is stored in a cell array, *latt*, of the structures called r passed to functions. This includes the data given above from the *in* structures. In addition, it includes the computed parameters given below, which includes various internal array dimensions. The stochastic variable arrays like a, w, v are internally flattened to become two-dimensional matrices where possible, to reduce dimensionality and simplify parallel operations.

For $d > 4$ total dimensions, the spatial grid, momentum grid and derivative grid notation of $t, x, y, z, \omega, kx, ky, kz$ and $r.Dx, r.Dy, r.Dz$ is changed to use numerical labels that correspond to the dimension numbers, i.e., $x\{1\}, \ldots x\{d\}$, $k\{1\}, \ldots k\{d\}$, $D\{2\}, \ldots D\{d\}$. These are simpler to handle with large numbers of space dimensions. This numeric labeling preference can be used even if $d < 5$, by setting the input switch $numberaxis = 1$. The graph labels will use this notation also, unless specified otherwise.

Label	Type	Typical value	Description
t	real	5	Current value of time, t
w	real	0.6	Frequency passed to in.observe : ω
x, y, z	array	-	Grid of x, y, z
kx, ky, kz	array	-	Grid of k_x, k_y, k_z
Dx, Dy, Dz	array	-	Grid of D_x, D_y, D_z
$x\{1\}, \ldots x\{d\}$	array	-	Grid of$x_1, \ldots x_d$
$k\{1\}, \ldots k\{d\}$	array	-	Grid of $k_1, \ldots k_d$
$D\{2\}, \ldots D\{d\}$	array	-	Grid of $D_2, \ldots D_d$
dx	vector	*0.2*	Steps in $[t, x, y, z]$
dk	vector	*0.6160*	Steps in $[\omega, k_x, k_y, k_z]$
dt	double	*0.2000*	Output time-step
dtr	double	*0.1000*	Reduced, internal time-step
V	real	*1*	Spatial lattice volume
K	real	*1*	Momentum lattice volume
dV	real	*1*	Spatial cell volume
dK	real	*1*	Momentum cell volume
xc	vector cells	*{[0,... 10]}*	Space-time coordinate axes in $[t, x, y, z]$
kc	vector cells	*{[0,..15,-15...]}*	Computational axes in$[\omega, k_x, k_y, k_z]$
s.dx	double	1	Initial stochastic normalization
s.dxt	double	*3.1623*	Propagating stochastic normalization
s.dk	double	1	Initial k stochastic normalization
s.dkt	double	*3.1623*	Propagating k stochastic normalization
nspace	integer	35	Number of spatial lattice points
nlattice	integer	3500	Total lattice: ensembles(1) x n.space
ncopies	integer	20	ensembles(2) x ensembles(3)
randoms	integer	2	Number of initial random fields
noises	integer	2	Number of noise fields
d.int	vector	*[1 1]*	Dimensions for lattice integration
d.a	vector	*[1 1]*	Dimensions for a field (flattened)
d.r	vector	*[1 1]*	Dimensions for coordinates (flattened)
d.ft	vector	*[1 1 1]*	Dimensions for field transforms
d.k	vector	*[0 1 1]*	Dimensions for noise transforms
d.obs	vector	*[1 1 1 1]*	Dimensions for observations
d.data	vector	*[1 3 51 1]*	Dimensions for average data (flattened)
d.raw	vector	*[1 1 51 1]*	Dimensions for raw data (flattened)

6.3 Graphics data

The simulation data of the *in* structure is also passed to the *xgraph* program, along with optional graphics parameters for each observable, to create an extended graphics data structure.

These are all cell arrays when there is more than one type of graph or observable, so that the graphics parameters can be individually set for each output that is plotted, using the cell index $\{n\}$ in a curly bracket. The axis labels, axis points and axis transformations of each graph are also cell arrays, as there can be any number of axes: so these are all nested cell arrays. If only a single input is specified, the cell index is automatically supplied. Graphics defaults are also user-modifiable by editing the *xgpreferences* function.

The plotted result can be an arbitrary function of the generated average data, by using the optional input *gfunctions.* If this is omitted, the generated average data from *xsim* is plotted.

Label	Default value	Description
gversion	*'xGRAPH1.22'*	Current version number
xlabels	{*'t' 'x' 'y' 'z'...*}	Generic axis labels
klabels	{*'\omega' 'k_x' 'k_y' 'k_z'...*}	Transformed axis labels
minbar{*n*}	*0.01*	Minimum relative error-bar
**gfunction*{*n*}	@(d,~) d{n}	Functions plotted of averages
font{*n*}	18	Font size for graph labels
headers{*n*}	”	Graph headers
images{*n*}	0	Number of movie images
imagetype{*n*}	0	Type of movie images
transverse{*n*}	0	Number of transverse plots
pdimension{*n*}	3	Maximum plot dimensions
**compare*{*n*}	-	Comparison functions
**xfunctions*{*n*}	-	Axis transformations
axes{*n*}	{0,..}	Points plotted for each axis
glabels{*n*}	{*'t' 'x' 'y' 'z'*}	Graph-specific axis labels

Bibliography

[1] S. Kiesewetter, R. Polkinghorne, B. Opanchuk, and P. D. Drummond, *xSPDE: extensible software for stochastic equations*, SoftwareX, March (2016).

[2] C. W. Gardiner, *Handbook of Stochastic Methods: for Physics, Chemistry and the Natural Sciences* (Springer 2004).

[3] P. D. Drummond and M. Hillery, *The Quantum Theory of Nonlinear Optics* (Cambridge University Press, Cambridge, 2014).

[4] P. D. Drummond and I. K. Mortimer, *Computer simulations of multiplicative stochastic differential equations.* J. Comp. Phys. **93**, 144-170 (1991).

[5] P.E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations* (Springer, 1995).

[6] M. J. Werner and P. D. Drummond, *Robust algorithms for solving stochastic partial differential equations.* J. Comp. Phys. **132**, 312-326 (1997).

[7] B. M. Caradoc-Davies, *Vortex dynamics in Bose-Einstein condensates* (Doctoral dissertation, PhD thesis, University of Otago (NZ), 2000).

[8] G. R. Collecutt, P. D. Drummond, *Xmds: eXtensible multi-dimensional simulator.* Comput. Phys. Commun. **142**, 219-223 (2001).

[9] Graham R. Dennis, Joseph J. Hope, Mattias T. Johnsson, *XMDS2: Fast, scalable simulation of coupled stochastic partial differential equations,* Computer Physics Communications **184**, 201–208 (2013).

[10] http://sourceforge.net/projects/xmds/