The Quantum and Stochastic Toolbox: xSPDE4

Peter D. Drummond, Run Yan Teh, Manushan Thenabadu
Channa Hatharasinghe, Chris McGuigan, Alex Dellios
Ned Goodman, Margaret D. Reid

March 8, 2025

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

This is the fourth major release of the xSPDE toolbox, which solves stochastic partial and ordinary differential equations, with applications in biology, chemistry, engineering, medicine, physics and quantum technologies. It computes statistical averages, including time-step and sampling error estimation. xSPDE can provide higher order convergence, Fourier spectra and probability densities. The toolbox has graphical output and χ^2 statistics, as well as weighted, projected, or forward-backward equations. It can generate input-output quantum spectra. The equations can have independent periodic, Dirichlet, and Neumann or Robin boundary conditions in any dimension, for any vector component, and at either end of any interval. xSPDE has functions that can numerically solve both ordinary and partial differential stochastic equations of any type, obtaining correlations, probabilities and averages. The toolbox has a core treating stochastic differential equations, with averages, probability distributions and full error estimates. There are stochastic extensions treating applications to partial differential equations, projected equations, quantum stochastic equations, master equations and quantum phase-space simulations including Gaussian boson sampling experiments.

Current version: 4.21

Contents

I. Introduction 7

II. Stochastic differential equations 11

1. SDE toolbox 12
   - 1.1. Using xSPDE 12
   - 1.2. SDE parameters 14
   - 1.3. Fields and observables 16
   - 1.4. Random fields and noises 19
   - 1.5. Multivariate probabilities 23
   - 1.6. Auxiliary fields and noises 25
   - 1.7. Time-domain spectra 26
   - 1.8. Scanned parameter plots 27
   - 1.9. Stochastic projections 29
   - 1.10. Weighted equations 32
   - 1.11. Forward-backward stochastic equations 33

2. SDE theory 37
   - 2.1. General form 37
   - 2.2. Stochastic calculus 38
   - 2.3. Example: random walk 39
   - 2.4. Interaction picture 40
   - 2.5. Stochastic equations with jumps 41
   - 2.6. Probability distributions 41
   - 2.7. Probability of a Wiener process 42
   - 2.8. Fourier transforms 43
   - 2.9. Forward-backward stochastic equations 44

3. SDE Examples 45
   - 3.1. Complex damped spectrum 45
   - 3.2. The Black-Scholes equation 49
   - 3.3. Kubo oscillator 50
   - 3.4. Loss and gain with noise 53
   - 3.5. Probability of a Wiener process 55
   - 3.6. Projected SDE on a catenoid 58
   - 3.7. Cell array coupled SDE 60

3.8. Additive FBSDE 63

III. Stochastic partial differential equations 69

4. SPDE toolbox 70

4.1. SPDE parameters 70
4.2. Multidimensional Wiener process 71
4.3. Differential operators 73
4.4. Spectral propagators 75
4.5. Transverse lattice 77
4.6. Boundary conditions 79
4.7. Output transforms 81
4.8. Initial random fields 82
4.9. Noise fields 84

5. SPDE theory 85

5.1. SPDE definitions 85
5.2. Boundary conditions 86
5.3. Spatial grid and boundaries 86
5.4. Multidimensional walk 87
5.5. Interaction picture 89
5.6. Fourier transforms 90
5.7. Trigonometric transforms 91
5.8. Transforms and boundaries 93
5.9. Frequency or momentum grid 96
5.10. Derivatives 97

6. SPDE examples 99

6.1. Gaussian diffraction 99
6.2. Stochastic Ginzburg-Landau equation 100
6.3. NLS soliton 102
6.4. Planar noise 103
6.5. Gross-Pitaevskii equation 104
6.6. Characteristic equation 107
6.7. Nonlinear Anderson localization 107
6.8. Nonlinear Schrödinger equation with Neumann boundary conditions . . . 108
6.9. Planar noise growth 111
6.10. Gross-Pitaevskii equation with vortex formation 115
6.11. Heat equation with non-periodic boundaries 118
6.12. Peregrine solitary wave with arbitrary boundary conditions 122

IV. Quantum phase-space **125**

7. Phase-space toolbox **126**
7.1. Quantum phase-space 126
7.2. Laser amplification noise 126
7.3. Input parameters 127
7.4. Gaussian boson sampling outputs 128
7.5. Sampling methods in phase-space 130

8. Quantum phase-space theory **133**
8.1. Phase-space representations 133
8.2. Dynamics in phase-space 137
8.3. Damped harmonic oscillator 139
8.4. Stochastic gauge expansion 140
8.5. Input-output spectra 142
8.6. Linear photonic network theory 143
8.7. Quantum input states 144
8.8. Photon counting 147
8.9. Intensity correlations 151
8.10. Grouped correlations 152

9. Phase-space examples **154**
9.1. Saturated laser noise 154
9.2. Nonlinear quantum simulation 154
9.3. Quantum linear oscillator 157
9.4. Quantum network 160

V. Quantum Monte-Carlo **162**

10. Quantum stochastic toolbox **163**
10.1. Wave-functions and density matrices 163
10.2. Bose-Hubbard Hamiltonian 164
10.3. Sparse matrix methods 164
10.4. Input parameters 167
10.5. Dissipative parameters 168
10.6. Support functions for operators 169
10.7. Sparse operators 170
10.8. Observe, expect, output and compare 170
10.9. SSE derivative 171
10.10Solving with the MCWF method 172

11. Quantum Monte-Carlo theory **173**
11.1. Master equations 173
11.2. Stochastic Schrödinger equation (SSE) 174

11.3. Stratonovich SSE . . . 176
11.4. Monte Carlo wave-function method . . . 179
11.5. Examples . . . 181

12. Quantum examples 184
12.1. Linear decay, complex SSE . . . 184
12.2. Time-dependent decay, real SSE . . . 186
12.3. Nonlinear decay, real SSE . . . 189
12.4. Two-mode Bose-Hubbard model . . . 192

VI. Methods, API and Examples 195

13. Stochastic methods 196
13.1. Introduction to algorithms . . . 196
13.2. General differential form . . . 197
13.3. Standard methods . . . 198

14. Errors 202
14.1. Time-step discretization errors . . . 202
14.2. Iteration errors . . . 204
14.3. Statistical errors . . . 204
14.4. Chi-squared estimates . . . 207
14.5. Error outputs . . . 210

15. Simulation parameters and extensibility 213
15.1. Overview: high-level xSPDE functions . . . 213
15.2. Input, output and logic . . . 215
15.3. SDE and SPDE input parameters . . . 218
15.4. Quantum parameters . . . 235
15.5. User function reference . . . 237
15.6. xSIM support functions . . . 242
15.7. Quantum support functions . . . 243
15.8. Internal xSIM parameters . . . 246
15.9. Examples, testing and structure . . . 250

16. Graphics parameters 252
16.1. xGRAPH overview . . . 252
16.2. Parameter table . . . 253
16.3. xGRAPH Parameter reference . . . 262
16.4. xGRAPH structure . . . 270

Part I.

Introduction

xSPDE is an eXtensible Stochastic Partial Differential Equation solver.

There are many equations of this type [1–5] in physics, chemistry, engineering, biology, medicine, and finance. Typical applications are in physics, quantum technology and biostatistics [6–25], but the code has general applicability. The emphasis in xSPDE is on combining a simple user interface with a wide range of useful functions, including the essential features of averaging and global error estimates. The code enables an efficient use memory and parallelism, which is vital for large stochastic models, and it is able to be further extended if needed.

xSPDE solves both ordinary and partial differential stochastic equations which have partial spatial derivatives, like the Maxwell equation. It also solves forward-backward equations, projected equations, stochastic Schrödinger equations, master equations, and quantum phase-space simulations. The extensible structure of the code-base permits drop-in replacements of the algorithms. Different simulations can be carried out sequentially. This models different stages in an experiment or simulated environment. It can be used with or without noise terms, and can use a range of either built-in or user defined integration algorithms. This user guide describes xSPDE4, which is an improved and extended version of xSPDE3, and earlier toolboxes [26, 27].

xSPDE calculates and plots averages and probabilities of arbitrary functions of any number of complex or real fields, as well as Fourier transforms in time or space with any given dimensionality. Importantly, it gives error estimates for both the discretization and sampling error, but the algorithm, the step-size and the number of samples used is up to the user to control to obtain the required error levels.

Ordinary and stochastic differential equations of many types can be treated numerically [28, 29], including stochastic partial differential equations with space dependence [30]. Comparative χ^2 statistical tests are available. Additional libraries exist for projected, forward-backward, and weighted equations.

The algorithms included are designed to be useful and fast in many practical applications. Higher order convergence is obtained through order extrapolation. This allows higher-order convergence to be realized in a uniform way. More complex higher-order algorithms are known [29, 31], which can be included if preferred, as the code is extensible.

The code can be used interactively or in batch mode. All graphs, data, and input parameters, including default values, can be stored permanently using standard file-types. It has a fully integrated graphics program, xGRAPH, which graphs data of any dimensions, including multiple types of graphical output, error-bars and comparisons.

xSPDE supports parallelism at both vector instruction and multiple core level using array and parallel loop syntax. This version is Octave/Matlab based. Matlab is a commercial product, GNU Octave [32] is free and open-source. They each have excellent user interfaces and reliable implementations. Full parallel operation currently requires the Matlab parallel toolbox.

Part 2 covers SDE theory and numerical solutions. Readers who are simply interested in how to use the code can go directly to Chapter 1, which describes the numerical solution of SDEs with xSPDE. This includes an explanation of the user interface, how to input parameters and equations, how to define the output in terms of functional averages or probabilities, and how to define and access auxiliary fields and noises. This chapter uses the default algorithms, and a more detailed explanation is given in chapter 13. Chapter 2 has definitions and notations for stochastic differential equations (SDEs). This is useful for understanding later chapters. This part includes Ito and Stratonovich calculus, probability distributions and Fokker-Planck equations. It also explains and defines the Fourier input-output spectra used in quantum technology.

Part 3 gives the theory and numerical implementation of stochastic partial differential equations (SPDEs). It includes details of spectral methods and the interaction picture approach. It has an explanation of how Fourier transforms and discrete sine or cosine transforms are implemented. It also explains how boundary conditions can be implemented using finite differences. In Chapter 4, the practical approach to solving stochastic partial differential equations with xSPDE is explained. The techniques used are an extension of the ordinary SDE methods, so a thorough understanding of chapter 1 is strongly recommended.

Part 4 treats phase-space methods, including the Wigner, Q-function, Glauber P-function and positive P-function. These can be treated simply using Part 2 or 3, but the purpose of this tool-box is to provide functions for initializing, propagating and observing quantum systems. In particular it covers quantum networks such as Gaussian boson sampling quantum computers, together with scalable verification and photon-counting algorithms.

Part 5 treats open quantum system theory and their numerical solutions using stochastic methods. It treats master equations, phase-space methods, and stochastic Schrödinger equations. Logic gates are also included here. Although there are more specialized programs that are dedicated for this purpose, it is useful to understand how different types of decoherence can change gate operations. As it has a modular design, it is possible to include systematic or non-Markovian gate errors, as well as Markovian noise.

Part 6 is for reference purposes. Chapter 13 outlines the integration algorithms used in the manual. It includes a number of extended integration libraries, applicable to more specialized problems. This chapter also outlines how integration errors, including time-step and stochastic errors, can be estimated and displayed. Chapter 15 provides a reference for the details of the internals as well as a comprehensive explanation of the input parameters useful in xSPDE simulations. This explains how to create projects with separated computation and graphics, as well as workflow and data storage. It also provides an extensive description of the visualization aspects of xSPDE, using the integrated xGRAPH function, which includes an automatic 'cascade' of graphic output where high dimensional data is reduced to lower dimensional, visualizable data through projections.

Input parameters related to this are described as well. Data can also be graphed externally or stored for later analysis if preferred. Both average and raw trajectory data can be stored. However, the storage of raw data is generally not recommended, due to the large storage requirements. Additional examples in Chapter 15.9 demonstrate how to obtain parametric plots against input parameters. Plots of one component value against another can be graphed. A function that analyses convergence rates is also available.

To run xSPDE, an Octave or Matlab environment is needed. A Julia option will be available in future. The current xSPDE distribution includes the toolbox: *xspde.mltbx*, or a folder: xspde_matlab, which includes the following:

- Simulator folder with the core functions
- Methods folder for the different applications
- Examples folder that can also be used as templates
- Graphics folder for the integrated graphics
- Documentation folder with this user's guide
- License.txt that contains the BSD license

xSPDE can be run interactively as a script, or as a function in batch mode, either at a local workstation or on a remote cluster. Data can be either plotted immediately, or saved then plotted later. To simulate a stochastic equation interactively, first check that the toolbox or folder is installed.

If you have the toolbox file, *xspde.mltbx***, just open it and click on** *install***.** Otherwise the Octave/Matlab path must point to the xSPDE folder and subfolders. If you have the folders, but not the toolbox, proceed as follows:

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

Type *clear* to clear old data, and enter the inputs and functions into the command window interactively. For more advanced cases, it is best to create a function that calls xspde. There are many examples listed in this manual, and there are more in the Examples folder. Any of these can be used as templates for building your own simulation code.

See: **www.github.com/peterddrummond/xspde_matlab.** For those familiar with earlier versions, a list of the main xspde changes since the documentation of the previously published version (v3.44) [26, 27] is as follows:

1. Cell arrays for multiple variables with differing labels and/or spatial grids
2. Error-checking outputs with both maximum and RMS error estimates
3. Quantum stochastic Schrödinger equations
4. Integration of master equations
5. Jump algorithms, in addition to Gaussian noise methods
6. DST and DCT spectral methods for SPDEs with non-zero boundaries

xSPDE is distributed with no guarantee, under an open-source license. Contributions and bug reports are welcome. An alternative approach to SPDEs [33, 34] is available in C++ at http://www.xmds.org/.

Part II.

Stochastic differential equations

1. SDE toolbox

This chapter describes how to use the xSPDE numerical toolbox to solve an SDE to obtain and graph averages, spectra or probability distributions. For theoretical background, see Chapter (2). For detailed examples, see Chapter (3) .

1.1. Using xSPDE

Stochastic equations generally require numerical solutions. To obtain them, xSPDE has a parameter structure, p, that defines both the equations and numerical parameters. The equations are defined as user functions with arguments (fields..,noises.., parameters).

All input parameters are all passed to functions in the structure *p*. Complete details of the xSPDE input parameters are given in 15.8.3. There are default options that allow one to reduce the required parameter inputs and functions to just the important ones. The three most essential user-specified functions are listed below:

Label	Arguments	Purpose
initial	(w, p)	**Function to initialize fields**
deriv	$(a, ..w, ..p)$	**Stochastic derivative**
observe	$(a, .., p)$	**Observable function**

In the table, " . . . " indicate optional arguments used if there are multiple field variables and noise variables. In the simplest case, the default option is one field and one noise. These can also be vectors if required. If this is needed, use the fields and noises input parameters explained below.

xSPDE has two parts, xSIM for the simulations and xGRAPH for automatic graphic generation. They can be used together in xSPDE, or individually as a batch job, so that data can be stored and graphed separately.

1.1.1. Example 1: the Wiener process

To use xSPDE to solve for a single trajectory of a simple SDE:

$$\dot{a} = w(t)\,, \tag{1.1}$$

which is described in more detail in Eq (2.12) , only two lines are needed:

```
p.deriv = @(a,w,p) w;
xspde(p);
```

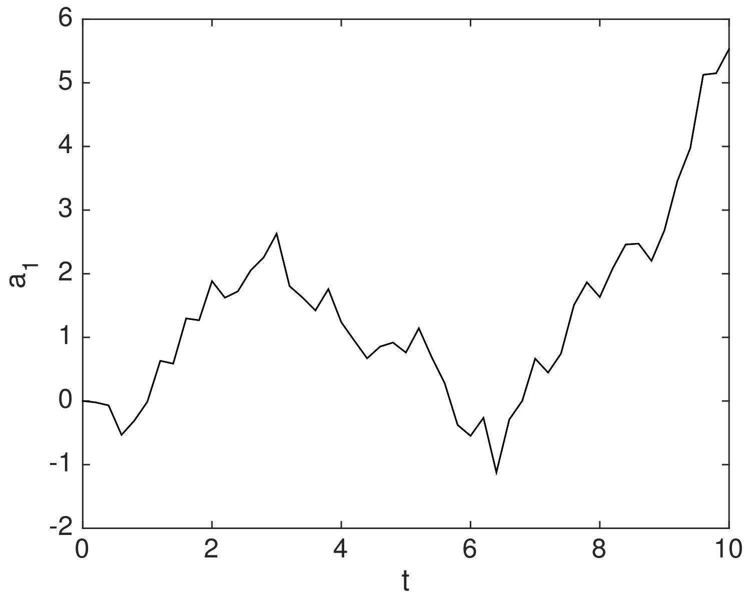


Figure 1.1.: *The simplest example: a random walk.*

Here *p.deriv* defines the time derivative $\dot{a}$ in the input parameter structure p, while w is a delta-correlated Gaussian noise generated internally. There are no other input parameters. The default values are used for *initial*, *observe*, *ensembles*, *points*, *ranges* and *olabels*. This produces the graph shown in Fig (1.1), for a single trajectory.

Results can change with different seeds or random number generators.

At the end of the run, xSPDE reports the RMS errors. There are discretization, sampling and comparison errors, all normalized by the maximum observable value, unless compared to a result of zero. In the present simulation, the discretization or step error is about 10^{-16}, due to round-off. This is a single trajectory, but more can be added using the ensembles input parameter.

1.1.2. General derivatives

All important xSPDE procedures use functions. Functions can be specified inline, which is the simplest, or externally. The last argument of more complex xSPDE functions is the parameter structure. An example already introduced is the derivative function, labeled *p.deriv*.

For example, consider the stochastic differential equation,

$$\frac{da}{dt} = -ga + w. \tag{1.2}$$

The corresponding derivative code definition is:

```
p.deriv = @(a,w,p) - p.G*a + w;
```

This code defines the function handle *p.deriv*, which gives the derivative function, da/dt. In this example, it simply returns the derivative, in terms of the variable a, loss parameter $p.G$, and stochastic noise term w. This user specified inline function is known internally by the function handle *p.deriv*.

Inside a complete xSPDE simulation input with a parameter values, it would look like:

```
p.G = 0.25;
p.deriv = @(a,w,p) - p.G*a + w;
xspde(p);
```

External function handles can also be used. They are useful for complex functions with more internal logic. A typical script first defines parameters and function specifications, in a structure, then runs the simulation code with the parameter structure as an input, as follows:

```
p.[label1] = [parameter1];
...
p.[label2] = [parameter2];
p.deriv = @(a,w,p) [derivative];
xspde(p);
```

Note the following points to remember:

- `p.`[*label*1] = [*parameter*1] defines a parameter in the structure p.
- There are many possible inputs, which all have default values.
- You don't have to save the data if you want an immediate plot.
- The notation `p.deriv` =@(`a`, `w`, `p`) [*derivative*] defines a function, da/dt.
- In this example, `a` is the stochastic variable, `w` the random noise, p a structure.
- Other labels can be used instead of (a,w,p) if preferred.

1.2. SDE parameters

All xSPDE simulations use a structure for input data. Most functions also require a parameter structure, combining the data input with additional internal parameters. Any naming convention will do for either structure, as long as you are consistent.

User-defined parameters can be added freely. To ensure that there is no clash with internal variables, it is best if user defined parameters start with a capital letter.

The xSPDE inputs have default values, which are used if the input values are omitted. If you only need the first element of a vector or array, just input the value required. Parameters can be output with the verbose switch, `p.verbose`. This has four levels of output: -1,0, 1 or 2, with `p.verbose=0` as default, giving final error reports. To get more progress details and individual errors, use `p.verbose=1`. To eliminate almost everything, use `p.verbose=-1`. For maximum information, including all the internal parameter values, use;

```
p.verbose = 2;
```

While this level of detail is not usually needed, it can be useful to print out all the internal parameters and default values to understand how the program operates.

1.2.1. Parameter table

The most common xSPDE simulation parameters used to define the equations and method of solution, together with their default values are:

Label	Type	Default value	Description
fields	vector, array or cell	1	Number of stochastic fields
noises	vector, array or cell	fields	Number of noises
inrandoms	vector, array or cell	noises	Number of initial randoms
name	string	' '	Simulation name
deriv	function	0	The stochastic derivative
initial	function	0	Function to initialize variables
method	function	[see 13]	Integration method
ensembles	integer vector	[1,1,1]	Stochastic ensemble sizes
ranges	real vector	[10]	Time and space ranges
points	integer vector	[51]	Output lattice points in [t,x,y,z,..]
steps	integer	[1]	Intermediate steps per time point
observe{n}	function	a	Observable function for averages
compare{n}	function	0	Comparison function for averages
binranges{n}{m}	vector	[0]	Binning ranges for probabilities

Table 1.1.: Table of most common simulation parameters.

A more detailed explanation of these parameters is found below, and a complete table is given in section 15.3. Fields and noises can be cell arrays if there are multiple scalar or vector fields required.

1.2.2. Graphics parameters

The generated average data can be graphed using any graphics editors, or else using the internal xGRAPH function defined for this purpose. An xSPDE simulation can return

Label	Type	Default value	Description
olabels{n}	string	'a'	Observable label
transverse{n}	integer	0	Transverse slices in time
transforms{n}	vector	0	Set to 1 for Fourier transforms in time
scatters{n}	integer	0	Set to s for s scatter plots in the observable

Table 1.2.: Table of most common graphics parameters

many different averages. These are defined in a cell array with indices in braces. The index is used to address the output data produced.

For each index, one can define parameters that define the quantity stored, together with corresponding graphics outputs. Some commonly used options are:

The full definition of the options is given in the user guide in sections 15.3 and 16.2, although many will be clear from examples.

1.2.3. Hints

- When first using xSPDE, it is a good idea to run the batch test script, Batchtest.
- If the Matlab parallel toolbox is not available, don't use the third ensemble setting.
- To create a project file, it is easiest to start with an existing example function.
- Graphics parameters can be included in either xSPDE or xGRAPH inputs.
- *Compare* allows comparisons with analytic results.
- Chapter 15 lists the input parameters.

1.3. Fields and observables

Stochastic variables in an SDE are fields, stored in a real or complex matrix, $a(f, e)$. Here, f is an internal field index, while e is the ensemble index.

fields gives the range of the first internal index. This has a default value of $fields = 1$. For multiple labeled stochastic fields, fields is a cell array that can specify the dimensions of one or more real or complex arrays, $a(f, e), b(f, e)$...

ensembles allows multiple trajectories to be integrated. This has up to three components. The first component, ensembles(1), gives a vector of local trajectories, so $e = 1, \ldots$ensembles(1). The second ensemble value species ensembles calculated in series, the third specifies ensembles calculated in parallel using multiple cores.

noises are noise dimensions, similar to fields, and used as $w(i,j)$, where the first noise index has *noises* components. The default value is $noises = fields$. Like fields, this can be a cell array of multiple noise dimensions.

In the example above, we could add the fields, dimensions, ensembles and noises:

```
p.fields = 1;
p.dimensions = 1;
p.noises = 1;
p.ensembles = 1;
```

As these are all default values, this is superfluous in a simple case. The full definition of ensembles as a vector is given above. If the third ensemble value is input, it requires the parallel toolbox in Matlab. For an SDE, $p.dimensions = 1$ is the default value, as there is only a time dimension. This input is only needed for stochastic partial differential equations, described in Part (III).

A more complex valid input could be:

```
p.fields = {[2,3],2};
p.noises = 4;
p.ensembles = [10,10];
```

This describes an equation with one 2×3 array, one 2-dimensional vector and one 4-dimensional noise, integrated with 100 trajectories composed of 10 local trajectories repeated 10 times in series.

1.3.1. Initial values, points and ranges

Initial values are required to define any differential equation, and in a numerical calculation one must also have a defined lattice.

initial The initial value is defined by a function *p.initial*. This must return either an initial vector of size fields, or else a random array of size $fields \times ensembles(1)$. The default function simply returns zero. If there is more than one field variable, initial is a cell array, which must be specified.

inrandoms are initial random number dimensions, similar to fields, and used as $v(i,j)$, where the first random dimension has *randoms* components. The default value is $randoms = noises$. Specifies the first argument of the function $p.initial(v,p)$ as a real Gaussian noise vector v with unit variance. The same noise is used when error-checking, so that changes are from the step-size, not from random fluctuations.

points The number of integration points. The default setting is currently 51.

steps The number of integration steps used for each output time-step. The default is 1.

ranges The total integration range in each dimension, the first element being the maximum integration time T. The default setting is currently 10.

1.3.2. Observables

observe is a cell array of functions of stochastic fields, each defining an average. xSPDE expects a (named or anonymous) function that takes two parameters, namely the field matrix a and the input structure p. The function must return a real or complex array, where the first index is used for a vector observable. xSPDE then averages over the last index, to calculate the observable. The default returns all elements of the first cell, as lines.
To plot the variance, for example:

```
p.observe{1} = @(a,p) (a(1,:)-mean(a,2)).^2;
```

rawdata By setting p.rawdata=1 (see section 15.3), one can also store every trajectory including both fine and coarse time-step values, but this is very memory-intensive for large simulations.

olabels is cell array of the output labels associated with each average, although one can also define additional functional transformation of the averages to be graphed and label them.

Observables are computed as a two-dimensional packed array, then unpacked for storage, giving an array of dimension $(d1, dspacetime, ensembles(1))$. Here $d1$ is the local observable dimension, so $d1 = 1$ for a scalar observable. The space-time dimension is $dspacetime = 1$ for an SDE, otherwise a vector for a SPDE, and $ensembles(1)$ is the size of the ensemble of trajectories computed in each processor. Once data is averaged internally over $ensembles(1)$, further transforms of the averages are available.

1.3.3. Using the dot

All equations entered in xSPDE utilize the Matlab syntax. This is designed to handle scientific or mathematical matrix and array-based formulae. It has features to simplify matrix or array equations which often require a 'dot' or a 'colon'.

- Stochastic variables in xSPDE are matrices or arrays, where the last index is used to treat parallel stochastic trajectories, for greater efficiency. This requires use of the 'dot' notation to perform multiplication inside equations.
- To multiply vectors, matrices or arrays element-wise, like $a_{ij} = b_{ij}c_{ij}$, the notation $a = b. * c$ indicates that all the elements are multiplied. This is used to speed up calculations in parallel.
- An equation in xSPDE can apply to many stochastic trajectories in parallel. Using the dot shortens the equation, and it also means that a fast parallel arithmetic will be used. The same principle holds for larger arrays with spatial lattices, treated in in section 4.
- Broadcasting occurs if one or more dimensions has a unit size. For example, arrays of size (1,100) and (6,1) can be added or multiplied to give a (6,100) matrix.

- A formula may require addressing the first index - which is the field component - and treating all the other elements in parallel. To do this in a compact way, one may use the notation $a(n,:)$, which indicates that all the subsequent index elements are being addressed as well.
- This will "flatten" a spatial array into a matrix, in which case it is better to include space indices.

In summary, whenever a formula combines multiplication operations over spatial lattices or ensembles, **USE THE DOT**.

1.4. Random fields and noises

1.4.1. General structure

Noises and initial random variables are cell arrays of scalar, vector or array variables, like the field cells. If there is only a single cell, then just the dimensions are input. The dimensions should be such that the corresponding arrays can be added.

1.4.2. Initial randoms

When *inrandoms* $\sim= 0$, initial Gaussian random numbers $\mathbf{v}$ are generated, with unit variance. When *urandoms* $\sim= 0$, an initial field of uniform random numbers is generated for jump processes. These can all be specified as cells of multiple random fields, with arbitrary scalar, vector or array dimensions.

They are passed to the initial function in the order of [randoms,urandoms].

1.4.3. Noise fields

During propagation in time, noises are Gaussian noise fields that are delta-correlated in space-time. They are calculated in an analogous way, except with an additional factor of $1/\sqrt{\Delta t}$ because they are delta correlated in time. They have a variance of $\sigma^2 = 1/(\Delta t)$.

During error-checking, which is the default option, the time-step is halved. The same noise trajectory is used, except that for coarse steps the two fine step noises are summed, which has the effect of doubling the variance, as required.

When *unoises* $\sim= 0$, an initial field of uniform random numbers is generated for jump processes, on the interval $[0,..1/\Delta t]$. During error-checking the same noise trajectory is used, except that for coarse steps the minimum of the two fine step noises is used, so that if there is a jump in at least one of the fine steps, it will occur in the coarse step.

All noises can all be specified as cells of multiple noise fields, with arbitrary scalar, vector or array dimensions. They are passed to the deriv function in the order of [noises,unoises].

The maximum number of noise or unoise cells equals the number of field cells. If more are required, extra field cells should be specified with zero dimensions.

1.4.4. Example 2: matrix SDEs

As an example, consider a matrix SDE where one Gaussian noise is specified to act on all rows, the other acts on all columns, and initial uniform randoms act independently on all elements. In this case the random variable is a matrix, and the noises can be most readily specified as a cell array of two vectors, one a row vector and one a column vector.

```
p.urandoms = [2,2];
p.fields = [2,2];
p.noises = {[2,1],[1,2]};
p.deriv = @(a,u,v,p) -a+u+v;
p.initial= @(w,p) w;
e = xspde(p);
```

Since there is no specification for the observables, xSPDE4 plots all four random matrix variables.

1.4.5. More advanced random walks

We now return to the random walk, but with the more advanced feature of ensemble averaging:

$$\dot{a} = w(t)\,, \tag{1.3}$$

This is integrated numerically and graphed with $N = points(1)$ points. The first point stored is the initial value, so there are $N-1$ integration steps, of length $dt = ranges(1)/(N-1)$. Numerical graphs have discrete steps, and more detail is obtained if more time steps are used. The default value is $N = 51$, which is predefined in the $xpreferences$ file. This is adjustable by the user. It can also be changed for a simulation, by inputting a new value of *points*.

Unless you type clear first, any changes to the input structure are additive; so in the exercises you should get the combination of all the previous structure inputs as well as your new input.

- **Run the xSPDE script of Example 1 in Matlab.**

It is simple to cut and paste from an electronic file to the command window. Be careful; pasting can cause subtle changes that may require correction. Some generated characters may be invalid input characters, and these will need retyping if this occurs.

You should get the output in Fig (1.1).

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

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

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

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

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

$$\dot{a} = -a + w(t)\,, \tag{1.4}$$

```
p.deriv = @(a,w,p) -a+w;
xspde(p);
```

The corresponding Fokker-Planck equation from Eq (2.27) is:

$$\frac{\partial P\left(a\right)}{\partial t} = \left[\frac{\partial}{\partial a} + \frac{1}{2}\frac{\partial^2}{\partial a^2}\right] P\left(a\right). \tag{1.5}$$

It is easy to verify that inserting this dynamical equation into Eq (2.29) gives the result:

$$\frac{\partial}{\partial t}\left\langle a^2\right\rangle = 1 - 2\left\langle a^2\right\rangle \tag{1.6}$$

- Solve for $\left\langle a^2\left(t\right)\right\rangle$ and use xSPDE to compare the numerical and analytic solutions. The current time is accessible as the parameter *p.t*. Can you explain the graph differences?

1.4.6. Discrete Fourier transforms

While exact in this analytic case, the definition above is impractical for numerical calculations. In taking measurements and doing simulations, one has a discrete set of data-points. Assuming the samples are at fixed intervals, the best one can do in practical cases is a discrete Fourier transform, with samples $\bar{a}(\bar{t}_j)$ that are defined as integrals over each small interval dt:

Let $\bar{a}(\bar{t}_j)$ be the average over a small time interval:

$$\bar{a}(\bar{t}_j) = \int_{t_j}^{t_j+dt} a(t)dt\,, \tag{1.7}$$

then to a good approximation as $dt \to 0$, provided ω_n is not too large,

$$\begin{aligned}\tilde{a}(\omega_n) &= \frac{\Delta t}{\sqrt{2\pi}}\sum_{j=1}^{N} e^{i\omega_n\bar{t}_j}\bar{a}(\bar{t}_j)\\ \bar{a}(\bar{t}_j) &= \frac{\Delta\omega}{\sqrt{2\pi}}\sum_{n=1}^{N} e^{-i\omega_n\bar{t}_j}\tilde{a}(\omega_n)\,.\end{aligned} \tag{1.8}$$

These also form an invertible pair provided that $\Delta t \Delta \omega = 2\pi/N$. As well as being more practical, this is very efficient due to the fast Cooley-Tukey (FFT) algorithm [35], allowing computation on time-scales of $O\left(N \ln N\right)$ rather than $O\left(N^2\right)$ as one might expect.

When taking Fourier transforms in the time-domain, xSPDE does a time-averaging of all fields over the current time-step, using the available coarse and fine time-samples. This is done by averaging the field before and after the stochastic time-step. The methods used for this are described in greater detail in Section (1.7).

1.5. Multivariate probabilities

One can utilize xSPDE to graph probability densities of real observables instead of averages, if *p.ensembles* is large. This is achieved by inputting the observable number and binning range:

$$p.binranges\{n\} = \{oa : ostep : ob\}; \tag{1.9}$$

If present, this returns probability density of the n-th observable $o\{n\}$, through binning into ranges of width *ostep* around the centers of each bin, starting at *oa*, and ending at *ob*. The simulation returns a result of $1/ostep$ in the $j-th$ bin if the trajectory is inside the bin, so that $o(j) - ostep/2 < o < o(j) + ostep/2$, and zero otherwise. This gives a probability density on output, plotted against time. Note that on graphing, an extra dimension is added for the variable o. The probability density at *ntimes* equally spaced simulation times can be plotted with *p.transverse*$\{n\}$=*ntimes*.

The probability can be plotted for any observe function of the stochastic variable. For these plots, the ordering of axes is: [time, space, observable], where the appropriate axis label can be added to the graph using the glabels{n}{k} graphics input, where k identifies the axis that is being labeled.

The probability density is multivariate for vector observables. This is possible because the binning ranges are stored in a cell array, which may contain several bin vectors. If the observable $o\{n\}$ is two-dimensional, then one can input:

$$p.binranges\{n\} = \{oa(1) : ostep(1) : ob(1), oa(2) : ostep(2) : ob(2)\}; \tag{1.10}$$

On graphing, two extra axes are added for the variable o in this case. The graphics program xGRAPH will attempt to graph them, but it is limited by graphical visualization constraints. In general, an arbitrary observable dimension is possible, but this is also limited by the sampling and memory, since the number of samples per bin will decrease rapidly with dimensionality.

The graphics program extracts slices and windows of probabilities if required. To plot the probabilities of two observables in different graphs, one for a range of $-5:5$ and the other for 0:25, add the following inputs before the xsim or xspde command:

```
p.binranges{1} = {-5:0.25:5};
p.binranges{2} = {0:0.5:25};
```

In the case of a two-dimensional probability density, plotted against time, there are a total of four graphics dimensions. That is, one for the probability, one for time, and two for the independent variables at each time. One can also plot how the probability density changes in space for the case of a stochastic partial differential equation, as described in section 4.

1.5.1. Marginals and labels

We now treat special cases that often arise.

- Suppose one has computed two outputs, but only the second one requires binning? Putting this another way, if the stochastic variable is $[x, y]$, you can plot $P(x, y)$ easily enough, but what if you want a marginal, $P(x)$ or $P(y)$? One way is to change the observe function to only return x or y.
- Sometimes the observe function is written, and one doesn't want to change it. Then for a marginal, all one has to do is to change the binning statement, and replace the unwanted binning range by an empty vector, []. This variable will be omitted, and again there will be a marginal probability.
- Probability axis labels can be added using the graphics input "glabels", if you would like axis labels. These correspond to the plotted axes, so that the label that is integrated over is omitted. In other words, just enter the labels of variables that are plotted, not those that are ignored.
- xSPDE can also integrate matrix or tensor stochastic differential equations. How does one compute their probabilities? The answer to this is simple. The observe function can only have a vector output, so the tensor or matrix has to be expanded into a vector by the observe function.
- If there are more binranges than variables, the last ranges that are input to binranges are ignored.
- If there are more variables than binranges data, the last variables calculated will be ignored.

1.5.2. Probability summary

In summary, if p.binranges{n} is specified, it takes multidimensional arrays generated by observe{n}, and outputs a probability distribution instead of the n-th average. The first data index or "line" dimension gives the independent variables. The last dimension is the sample index, which is averaged over.

If there are m line index values, and an m-dimensional set of bins, an m-dimensional joint probability is computed by adding up the samples in each bin. Since there are usually more than two indices in total, xSPDE generates independent probabilities for each extra index value.

All results really depend on observe{n}. If it generates data with a singleton first index, you get a one-dimensional probability. When there are more than two indices in the observe data, you get a distinct probability for each value of the other indices. This gives multiple probabilities versus the extra indices.

In xSPDE, without binning, and three index values, you get a 2D plot, not a 3D plot, since the first index indexes the lines, the second the time axis, and the third is averaged. With binning, binranges will not ignore the time dimension. It simply generates a new probability plot for each time.

When there are extra space dimensions as well, there are probabilities at every space-time point. These are not joint probabilities, since the curse of dimensionality would make them too high-dimensional and sparse. One can also reduce the amount of data generated with “axes” (see later).

1.6. Auxiliary fields and noises

In some cases it is useful to access the noise terms, or functions of the noises and their correlations with the fields at the same time. This is handled in xSPDE with auxiliary fields or auxfields. These are fields that are functions of noise terms and the integrated fields. The number of these is defined in the input structures using the parameter p.auxfields, which is arbitrary.

Auxiliary fields are calculated using a function p.define, which is similar to p.deriv, except that it specifies the current value of the auxiliary field, not the derivative. These fields are defined as the average over the previous step in time of the auxiliary function, including the noise term. This is essential in calculating spectra, in order to eliminate systematic errors in Fourier transforms.

More details on this are given in Section (1.7). To access the auxiliary fields, one can compute any observable average using a p.observe function as usual, or else store the raw trajectories including auxiliary fields by setting p.rawdata=1. In either case, the auxiliary fields are appended to the integrated fields by adding extra cells.

1.6.1. Outputting the noise

As a simple example, suppose one wishes to calculate the noise terms and compare them with the field trajectories in a simple Wiener process. Since there is now an extra cell for the auxiliary field in the define function, it is passed as an additional field argument to the observe function. The following code can be used:

```
clear
p.auxfields = 1;
p.deriv = @(a,w,p) w;
p.define = @(a,w,p) w;
p.observe = @(a,x,p) [a;x];
p.olabels = {'a, w'};
xspde(p);
```

The observe function calculates both rows of the output array, including the auxiliary field which is defined as the noise term and plotted as a dashed line. There is no ensemble averaging, and hence no ensemble error-bars in the example. This is because because no ensembles were specified in the input parameters. Similarly, there are no time-step error-bars for this observable, because the fine and coarse noises are equal to each other after time averaging.

The result that is plotted is therefore the coarse noise, whose correlation time equals the time step. This is plotted below in Fig (1.2), which plots the same Wiener process as before, except adding the driving noise term as well. The standard deviation of the

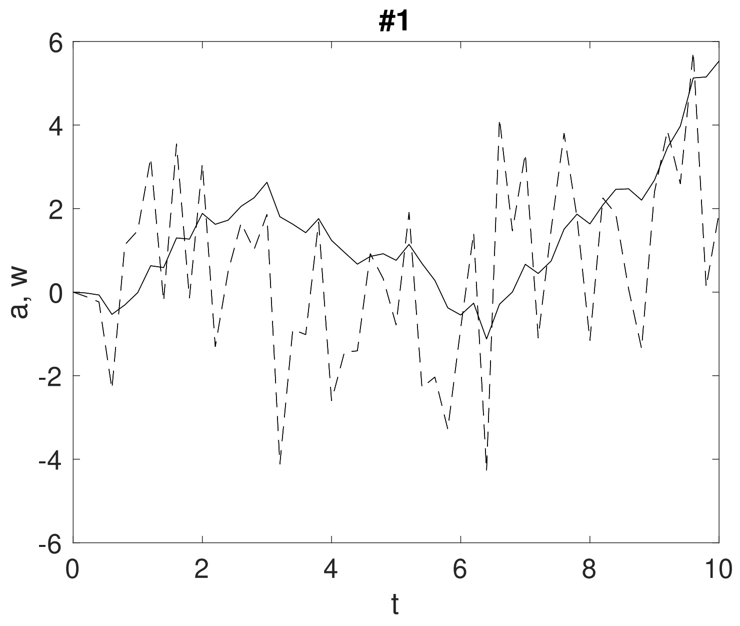


Figure 1.2.: *A single trajectory of a random walk, with the noise terms w graphed using dashed lines, and the integrated variable a plotted as the solid line.*

noise in a single step here is $\sqrt{1/dt}$, where $1/dt = 50/10 = 5$ for the default range of 10 and default time points of 51. Note that noise terms do not converge at small time-steps for delta-correlated noise, even when the integrated stochastic process does converge. This is why it is necessary to choose to plot one or the other, or else to time-average to obtain a converged result.

If multiple steps are used, only the noise during the last step prior to the time-point is plotted.

1.7. Time-domain spectra

To get an output from a temporally Fourier transformed field, set $transforms\{n\} = 1$ for the observable (n) you need to calculate in transform space. This parameter is a cell array. It can have a different value for every observable and for every dimension in space-time, if you have space dimensions as well.

To obtain spectra from Eq (1.7) with greater accuracy, all fields are must be averaged internally. The code will use trapezoidal integration in time over the integration interval, to give the average midpoint value. This employs the same interval for fine and coarse integration, to allow comparisons for error-checking. After this, the resulting step-averaged fields are then Fourier transformed.

In the simplest case of just one internal step, with no error-checking, this means that the field used to calculate a spectrum is:

$$\bar{a}_j = (a_j + a_{j+1})/2, \tag{1.11}$$

which corresponds to the time in the spectral Fourier transform of:

$$\bar{t}_j = (t_j + t_{j+1})/2. \tag{1.12}$$

Note that if any temporal Fourier transform is specified, all the field variables are time-averaged over a step. This is not strictly necessary, but it means that there is a reduced code complexity for cases where there is a Fourier transform for some but not all variables. As described above, the auxiliary variables are always time-averaged to allow error-checking, so there is no change for these.

1.7.1. Error-checking

For an error-checking calculation with two internal steps, there are three successive valuations: a_j, $a_{j+1/2}$, a_{j+1}. In this case, for spectral calculations one averages according to:

$$\bar{a}_j = \left(a_j + 2a_{j+1/2} + a_{j+1}\right)/4. \tag{1.13}$$

In addition, one must define the noise terms, both for error-checking and for output, since spectral calculations in quantum input-output theory include noise terms as well as fields. The noise term used to calculate a spectrum involving $\bar{a}_j$ is w_j. A coarse noise term is set equal to the average of two successive fine noise terms:

$$\bar{w}_1 = \frac{1}{2}\left(w_1 + w_{1/2}\right). \tag{1.14}$$

The time integral is carried out numerically as a sum which has $N = points(1)$ time points of interval dt. In xSPDE, $dt = T/(N-1)$, where $T = ranges(1)$. The effective integration time for the Fourier transform time integrals is

$$T_{eff} = Ndt = 2\pi/d\omega \tag{1.15}$$

When there are larger numbers of steps from using the internal steps parameter, there are more points to Fourier transform. These additional frequencies are computed while carrying out the Fourier transform, but only N low frequency points are saved. The unused high frequency results are not stored or plotted, to conserve memory.

1.8. Scanned parameter plots

Since xSIM is a function that can be called, plots of results against simulation parameters are possible. This requires repeated calls to xSIM with different parameter values,

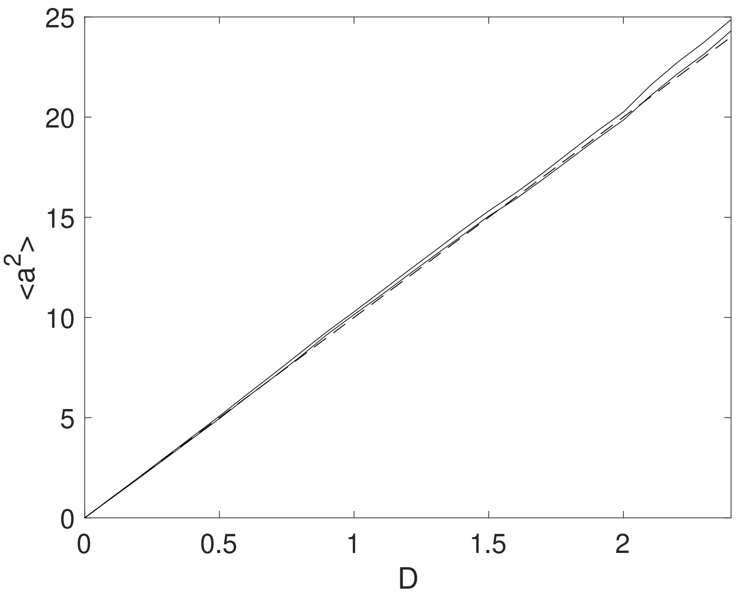


Figure 1.3.: *Scanned parameter output with a variable diffusion, for the case of a pure Wiener process, $\dot{a} = Bw(t)$. Exact value is the dashed line.*

together with data storage in an xGRAPH compatible form, and a call to xGRAPH. If different random seeds are required, the seed needs to be reset in each call. The relevant axes points plotted, labels and the values of scanned parameters also need to be input.

The simulation function xSIM uses the last data array index, c, to store the data values and up to two corresponding errors. This takes up three index values. A value of $c = 4$ is used to store comparison data, and its errors if there are any in $c = 5, 6$. This can be used for exact results, approximations, or experimental data.

1.8.1. Example: Scanned diffusion

As an example, consider the simplest possible stochastic equation, with a scanned diffusion:

$$\dot{a} = Bw(t) \,. \tag{1.16}$$

The equation is integrated over the interval $t = 0 : 10$, with $a = 0$ initially, using 10^4 trajectories to give an expected error of around $\pm 1\%$. The variance of a at $t = 10$ is plotted as a function of $D = B^2$, then compared to an exact value. The result is in Fig (1.3). The corresponding code is given as well.

```
function e = WienerScan()
p.name = 'Wiener process';
p.ensembles = [1000,10];
p.points = 12;
p.deriv = @(a,z,p) z*p.B;
p.observe = @(a,p) a.^2;
p.olabels = {'<a^2>'};
p.glabels{1} = {'D'};
scanpoints = 25;
data{1}{1} = zeros(1,scanpoints,4);
for j = 1:scanpoints
  p.seed = j;
  p.B = sqrt((j-1)*0.1);
  [e,data1,input,~] = xsim(p);
  data{1}{1}(1,j,1:3) = data1{1}{1}(1,p.points,:);
  xk{1}{1}(j) = p.B^2;
  D(j) = p.B^2;
end
data{1}{1}(1,:,4) = input.ranges(1)*D(:);
input.xk = xk;
input.axes{1}{1} = 1:scanpoints;
xgraph(data,input);
end
```

Here *p.deriv* defines the time derivative function $\dot{a}$, with w being the delta-correlated Gaussian noise that is generated internally.

1.9. Stochastic projections

It is sometimes necessary to constrain an equation to a sub-manifold [36], with an equation of form:

$$\mathbf{f}\left(\mathbf{a}\right) = 0, \tag{1.17}$$

where $\mathbf{f}\left(\mathbf{a}\right)$ is a scalar or vector function that defines the relevant manifold in Euclidean space. The projected SDE then has the form of a Stratonovich SDE, where:

$$\frac{\partial \mathbf{a}}{\partial t} = \mathcal{P}^{\parallel}_{\mathbf{a}} \left[\mathbf{A}\left[\mathbf{a}\right] + \underline{\mathbf{B}}\left[\mathbf{a}\right] \cdot \mathbf{w}(t)\right] , \tag{1.18}$$

where $\mathcal{P}^{\parallel}_{\mathbf{a}}$ is a tangential projection operator at location $\mathbf{a}$ on the sub-manifold, and as usual, $\mathbf{A}$ is a vector, $\underline{\mathbf{B}}$ a matrix and $\mathbf{w}$ is a real Gaussian noise vector, delta-correlated in time.

Similarly, the general stochastic partial differential equation can be written in projected form as

$$\frac{\partial \mathbf{a}}{\partial t} = \mathcal{P}_{\mathbf{a}}^{\parallel} \left[\mathbf{A}\left[\mathbf{a}\right] + \underline{\mathbf{B}}\left[\mathbf{a}\right] \cdot \mathbf{w}(t,\mathbf{x}) + \underline{\mathbf{L}}\left[\nabla, \mathbf{a}\right] \right]. \tag{1.19}$$

The projection library has three predefined algorithms,

- ***Enproj***,
- ***MPproj***,
- ***MPnproj***.

Here the capital E stands for Euler, MP for midpoint. All use tangential projection. The letter *n=normal* indicates if an additional normal projection is used. In all cases, if it is present, a normal projection is used last. The recommended type is ***MPnproj***, due to its much lower errors.

Tangential and normal projections are needed to define the geometry of any sub-manifold. These are input by setting the variable *project* equal to a function handle that defines the projection. These can be user provided if required. There are three different predefined manifold geometry types, which need different inputs, given below.

Currently, these specialized functions only apply to vector stochastic variables without a cell index.

1.9.1. Calling the project function

The calling arguments for the *project* function are: *(d,a,n,(c,)p)*, where d is a vector to be tangentially projected at location a, a is the current (near)-manifold location, n is an option switch, *c* is the (optional) cell index, and p is the parameter structure.

The options available in any *project* implementation are defined as:

- *n = 0* returns the tangent vector for testing
- *n = 1* returns the tangential projection of *d* at *a*
- *n = 2* returns the normal projection of *a*, where *d* is not used
- *n = 4* returns the constraint function at *a* for testing

The projections defined in an xSPDE *project* function can be of any type. Arbitrary dimension reduction and manifold geometry is possible. Currently in the examples, dimensionality is reduced by 1. Normal projections use fixed point iterations, defined by *p.iterations*, with a default of *p.iterations=4*.

1.9.2. The predefined manifold geometries

The current manifolds, by setting p.project = @Quadproj ..., are as follows:

1. Quadratic - *Quadproj* - needs a user defined matrix *p.qc* such that $f = \sum [qc]_{ij} x^i x^j - 1 = 0$

2. Polynomial - *Polproj* - needs a user defined column vector *p.vc* defined by $f = \sum [vc]_i (x^i)^p - 1 = 0$, where the power p is a user defined integer, *p.orderpol.* This reduces to the quadratic case if $p.orderpol = 2$.

3. Catenoid - *Catproj* - uses fixed coefficients defined by $f = (x_1)^2 + (x_2)^2 - (sinh(x_3))^2 - 1 = 0$

Any other manifold can be used by replacing these predefined manifolds with an appropriate *project* function.

As a typical example of a quadratic projection, the following code simulates surface diffusion on on a ten-dimensional hypersphere:

```
function [e] = Sphere10()
% e = SPHERE10() simulates 10d spherical diffusion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
p.name = 'Sphere10:  Diffusion on 10D hypersphere';
p.C = 1;
p.qc = eye(10);
p.X0 = [1,0,0,0,0,0,0,0,0,0]';
p.fields = 10;
p.ranges = 5;
p.points = 51;
p.ensembles = [40, 10, 10];
p.project = @Quadproj;
p.compare{1} = @(r) 2*(1-exp(-4.5*r.t));
p.compare{2} = @(r) 1+0*r.t;
p.deriv = @(a, w, r) w;
p.initial = @(w, r) r.X0;
p.observe{1} = @(a, r) sum((r.X0-a).^2,1);
p.observe{2} = @(a, r) (abs(sum(a.*(r.qc*a),1)));
p.diffplot = {1,0};
p.olabels = {'< R^2 >','< |f| >'};
p.method = @MPnproj;
e = xspde(p);
end
```

1.9.3. Cell array projections

Cell arrays of fields can be integrated with projections, provided the *project* function is suitably defined. With this approach, there are separate projections for each field cell. In the case of the predefined *project* functions, there must be a cell array of the projection parameters. If these are empty arrays, nno projection takes place, and the cell field is integrated as normally.

1.10. Weighted equations

In some types of stochastic equation, there is a weight associated with each trajectory, which is used to weight the probability of the trajectory [37]. This type of equation is sometimes found when dealing with quantum trajectories [38, 39] and feedback [40].

The equations still have the standard form of Eq (2.1), with an extra weight equation, Eq (2.6), which is expected in xSPDE to be the last cell of a cell array of variables. These weights are only included automatically if a special user-defined function, *p.expect,* is used to define the quantity whose weighted expectation is computed. By comparison, *p.observe* always computes an unweighted quantity. These can be mixed in a simulation, provided their cell indices are distinct.

The results for mean values using *p.expect* are automatically weighted by a term $\exp\left(\Omega\left(t\right)\right)$, so that:

$$\left\langle \mathbf{O}\right\rangle_{\Omega}=\frac{\sum_{n}\mathbf{O}\left(\mathbf{a}^{(n)}\right)\exp\left(\Omega^{(n)}\left(t\right)\right)}{\sum_{n}\exp\left(\Omega^{(n)}\left(t\right)\right)}. \tag{1.20}$$

This reduces to the standard *p.observe* expression of Eq (2.5) in the case that $\Omega\left(t\right)=0$. To simulate these equations, the weight exponent Ω is integrated as the *last* field in cell array, which must therefore have at least two components. A nonzero threshold weight, *thresholdw*, must be entered to allow calculation of breeding, to improve efficiency.

With these changes, averages in each vector ensemble are calculated using Eq (1.20). After each step in the calculation, a breeding calculation is carried out. During breeding, any weight such that $\exp\left(\Omega^{(n)}\right)<p.thresholdw/\left\langle \exp\left(\Omega\right)\right\rangle$ is removed.

The most probable trajectory is then duplicated to replace the low-weight trajectory. Both exponential weights are halved, so the total weight of the remaining trajectories is unchanged. If they are complex, weights such that $\exp\left(Re\left(\Omega^{(n)}\right)\right)<p.thresholdw/\left\langle \exp\left(Re\left(\Omega\right)\right)\right\rangle$ are removed, and the real weight of the bred trajectory is reduced, which removes any low-weight trajectories that don't contribute. When used, the internal variable *p.breedw* is updated at each step to allow the fraction of trajectories that are bred per step to be monitored. For weighted SPDEs, the spatial weights $\Omega(x_j)$ are summed over space points to obtain Ω.

1.10.1. Example

The following example shows how weighted stochastic equations are implemented.

```
function [e] = Weightcheck()
p.name = 'Weightcheck';
p.ensembles = [10000,10,1];
p.fields = {1,1};
p.points = 9;
p.order = 2;
p.thresholdw = 0.1;
p.diffplot = 1;
p.initial = {@(w,~,~) 1+w,@(~,~,~) 0};
p.deriv{1} = @(a,o,w,v,~) -a+w;
p.deriv{2} = @(a,o,w,v,~) -o+v;
p.expect{1} = @(a,o,~) a;
p.observe{2} = @(~,~,p) p.breedw;
p.compare{1} = @(p) exp(-p.t);
p.olabels{1} = '<a>';
p.olabels{2} = '<fractional-breeds/step>';
e = xcheck(2,p);
end
```

This algorithm converges with second-order accuracy for this exercise, due to the structure of the equation. The example also demonstrates how to use the *xcheck* function instead of *xspde*, to check convergence.

1.11. Forward-backward stochastic equations

The xSPDE program implements an iterative stochastic method to solve forward-backward (FB) stochastic differential equations, which propagate both forward and backward in time. These are equivalent to a Fredholm integral equation, and are solved by iteration.

The general FB equations have the following structure, written as an integral equation to make it clear what the relevant boundary conditions are:

$$\begin{aligned}
\mathbf{x}\left(t\right) &= \mathbf{x}\left(t_0, \mathbf{y}\left(t_0\right)\right) + \int_{t_0}^{t} \left\{ \mathbf{A}^x \left[\mathbf{a}\left(t'\right)\right] dt' + \underline{\mathbf{B}}^x \left[\mathbf{a}\left(t'\right)\right] \cdot d\mathbf{w}(t') \right\} \\
\mathbf{y}\left(t\right) &= \mathbf{y}\left(t_f, \mathbf{x}\left(t_f\right)\right) + \int_{t}^{t_f} \left\{ \mathbf{A}^y \left[\mathbf{a}\left(t'\right)\right] dt' + \underline{\mathbf{B}}^y \left[\mathbf{a}\left(t'\right)\right] \cdot d\mathbf{w}(t') \right\} . \qquad (1.21)
\end{aligned}$$

Here, $\mathbf{a} = \{\mathbf{x}, \mathbf{y}\}$ is a cell array including forward components $\mathbf{x}$ and backwards components $\mathbf{y}$. These have “initial” conditions in the past and the future, respectively, just as with ordinary stochastic equations. The forward fields have $p.fields$ components as usual, whilw the backward fields have $p.fieldsb$ components.

To solve these iteratively, an additional input parameter *p.iterfb* is needed, to specify the number of iterations, and this must have an integer value. The iteration converges in many cases, but this is not guaranteed, and convergence must be checked. The starting point of the iteration is given by the initial trajectory function $firstfb$.

The library includes the *xpathfb* function which replaces *xpath*, and is used automatically. The *initialfb* function initialises the forward and backward fields, each of which requires a different cell index. This function requires an additional field argument, since the initial condition may depend on the last value of the counterpropagating field.

The FB equation is solved in differential form, where $t_{-} = T - t$, as:

$$\begin{aligned}\frac{\partial \mathbf{x}}{\partial t} &= \mathbf{A}^{x}\left[\mathbf{a}\right] + \underline{\mathbf{B}}^{x}\left[\mathbf{a}\right]\cdot\mathbf{w}(t)\\ \frac{\partial \mathbf{y}}{\partial t_{-}} &= \mathbf{A}^{y}\left[\mathbf{a}\right] + \underline{\mathbf{B}}^{y}\left[\mathbf{a}\right]\cdot\mathbf{w}(t)\,.\end{aligned} \tag{1.22}$$

One must specify $firstfb$, but the default value of zeros can also be used. A negative value of *iterfb* can be used to indicate that the iterations should commence with the backward field *y*, rather than *x*, since this can give faster convergence where x(1) depends on the backward field y. Additional inputs used are as follows:

Label	Type	Default value	Description
fieldsb	integer	0	Backward field cells and dimensions
iterfb	integer	1	Forward-backward iterations
firstfb	function handle	$\{@(p)\ 0,...\}$	First trajectory estimate of all fields
initial	function handle	$\{@(y\{:\},w\{:\},p)\ 0,..\}$	Initial values in each field cell

The output of *firstfb{n}(p)* gives the first iterative path estimate of a in cell n with a default value of 0. An array size cell array, *p.fbsize*, indexed by a cell index in the range $1 : p.fbcells$ is available so the initial forward-backward field trajectories will be correctly sized. This cell array is indexed by the forward-backward field cell index.

The *initial(y{:},w{:},p)* function is used to initialize the forward components at t_0, and the backwards components at t_f. It requires an extra argument compared to the usual *initial* function. Forward cells $\mathbf{x}^{(n)}(0)$ use the last time-point of the previous backward component iteration as the first argument(s), i.e., $\mathbf{y}^{(n-1)}(0)$. Similarly, backward cells $\mathbf{y}^{(n)}(T)$ use the last time-point of the previous forward component as the first arguments. i.e., $\mathbf{x}^{(n-1)}(T)$.

To give better accuracy, the *pathfb* algorithm uses the mean value in time of the previous trajectory, so it is equivalent to a Stratonovich differential equation in each direction. This requires appropriate correction terms if there is multiplicative noise and the original equations are in the Ito formalism, as described in the following SDE theory section (2.2.1).

This method uses Picard iterations of a Fredholm integral equation, and is not always convergent. To improve convergence, $linear$ terms should be integrated using the linear interaction picture of xSPDE, which is exact. The *deriv* functions have their usual behavior. For more than one iteration an iteration error is reported as the difference of the last two estimates.

1.11.1. Example:

This example illustrates the input structure. It has two counter-propagating stochastic processes, one decaying in the forward time direction, and one decaying in the backward time direction:

$$\begin{aligned}\frac{\partial x}{\partial t} &= -x + w_1(t) \\ \frac{\partial y}{\partial t_-} &= -y + w_2(t).\end{aligned} \tag{1.23}$$

They are not coupled together here, but they have opposite initial conditions, with gaussian distributed final and initial values, each with unit variance:

$$\begin{aligned}x\left(0\right) &= 4 + u_1 \\ y\left(T\right) &= -4 + u_2\end{aligned} \tag{1.24}$$

In the numerical code, the default values are used so that $fields = 1$, $points = 51$, $ranges = 51$, $iterfb = 1$, $inrandoms = \{1, 1\}$, $noises = \{1, 1\}$, and the linear interaction picture is used in each direction. The results are computed and compared with exact results for the means and variances.

These are easily computed, to give:

$$\begin{aligned}\left\langle x\left(t\right)\right\rangle &= 4e^{-t} \\ \left\langle y\left(t\right)\right\rangle &= -4e^{t-T} \\ \sigma_x^2\left(t\right) &= \left(1 + e^{-2t}\right)/2 \\ \sigma_y^2\left(t\right) &= \left(1 + e^{2(t-T)}\right)/2\end{aligned} \tag{1.25}$$

```
p.fieldsb = 1;
T = 10;
p.ranges = T;
p.initial = {@(y,u,v,p) 4+u,@(x,u,v,p) -4+v};
p.ensembles = [200,10,1];
p.observe = {@(x,y,p) [x;y];@(x,y,p) [x.^2;y.^2]};
p.output{2} = @(o,p) o{2}-o{1}.^2;
p.compare{1} = @(p) [4*exp(-p.t);-4*exp(p.t-T)];
p.compare{2}= @(p) [(1+exp(-2*p.t))/2;(1+exp(2*(p.t-T)))/2];
p.legends = {{'<x>', '<y>'},{'\sigma_x^2',
'\sigma_y^2'}};
p.linear = {@(p) -1, @(p) -1};
p.deriv = {@(x,y,w,v,p) w, @(x,y,w,v,p) v};
e = xspde(p);
```

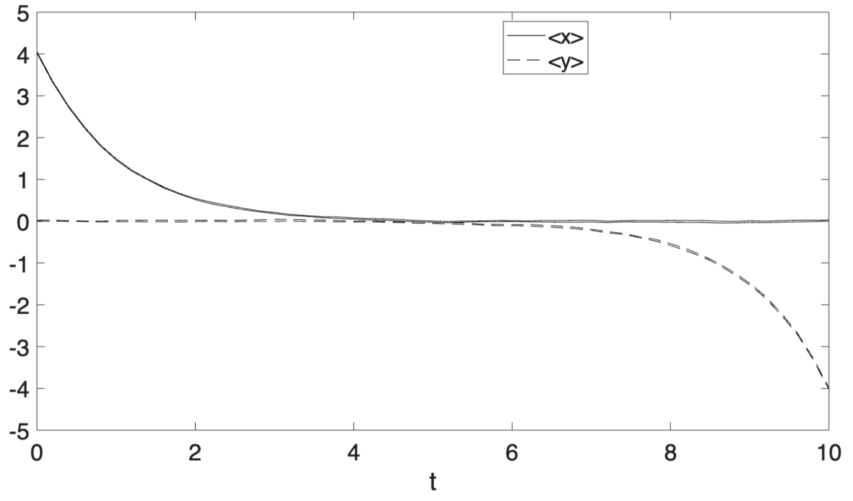


Figure 1.4.: *Forward-backward mean value solution, with an initial condition for x, a final condition for y, and independent noises.*

2. SDE theory

This chapter describes the basics of stochastic differential equation (SDE) theory, in order to explain the background to the numerical methods.

2.1. General form

A stochastic differential equation (SDE) is an equation with random noise terms. These were introduced by Langevin to treat small particles in fluids [3], and extended by Wiener, Ito and Stratonovich [41–43]. The theory and its applications to biology, chemistry, engineering, economics, physics, meteorology and other disciplines are treated in many texts [1, 2, 4, 44–46].

An ordinary stochastic differential equation in one time dimension is,

$$\frac{\partial \mathbf{a}}{\partial t} = \mathbf{A}\left(\mathbf{a},t\right) + \underline{\mathbf{B}}\left(\mathbf{a},t\right)\cdot \boldsymbol{\xi}(t)\,. \tag{2.1}$$

Here $\mathbf{a}$ is a real or complex vector, $\mathbf{A}$ is a vector function, $\underline{\mathbf{B}}$ a matrix function and $\mathbf{w}$ is usually a delta-correlated real Gaussian noise vector such that:

$$\left\langle \xi_i\left(t\right)\xi_j\left(t'\right)\right\rangle \;=\; \delta\left(t-t'\right)\delta_{ij}. \tag{2.2}$$

One can also have non-Gaussian noise or noise that is not delta-correlated. Although these are somewhat less commonly treated, these alternatives are often found in real applications.

These equations can alternatively be written in the equivalent form that:

$$d\mathbf{a} = \mathbf{A}\left(\mathbf{a},t\right)dt + \underline{\mathbf{B}}\left(\mathbf{a},t\right)\cdot d\boldsymbol{w}\,. \tag{2.3}$$

This leads to the relation that:

$$\frac{d\boldsymbol{w}}{dt} = \boldsymbol{\xi}(t). \tag{2.4}$$

In a finite time interval dt, one can approximately replace the delta-correlated noise by a fixed noise with variance $\left\langle \xi^2\right\rangle = 1/dt$, but if the alternative notation is used, then $\left\langle dw^2\right\rangle = dt$. The first notation is the one mostly used in xSPDE.

2.1.1. Observables

In all cases, there are multiple independent trajectories, and one is interested in probabilistic averages, where the unweighted average of an observable $\mathbf{O}\left(\mathbf{a}\right)$, for N_s trajectories

$\mathbf{a}^{(n)}$ is:

$$\langle\mathbf{O}\rangle_{N_s} = \frac{1}{N_s}\sum_n \mathbf{O}\left(\mathbf{a}^{(n)}\right). \tag{2.5}$$

In other types of stochastic equation [47,48], there is a weight $\Omega(t)$ for each trajectory. This has an additional equation of motion, where:

$$\frac{\partial\Omega}{\partial t} = A_\Omega(\mathbf{a},\Omega,t) + \underline{B}_\Omega(\mathbf{a},\Omega,t)\cdot\boldsymbol{\xi}(t). \tag{2.6}$$

The results for all mean values are then weighted by the term $\exp(\Omega(t))$, so that:

$$\langle\mathbf{O}\rangle_\Omega = \frac{\sum_n \mathbf{O}\left(\mathbf{a}^{(n)}\right)\exp\left(\Omega^{(n)}(t)\right)}{\sum_n \exp\left(\Omega^{(n)}(t)\right)}. \tag{2.7}$$

This expression reduces to the usual average if the weights are zero, i.e, $\Omega = 0$. Apart from the way that averages are treated, the weight can simply be regarded as an additional term in the stochastic differential equations. This simply means that one now has an equation with an extra random field, so that $\mathbf{a} \to [\mathbf{a},\Omega]$, together with a modified expression for the averages. This, in fact, is how these equations are solved.

For reasons of efficiency, it is best to use "breeding" algorithms to treat these numerically. This replicates highly weighted trajectories with $\Omega^{(n)}(t) \gg 0$ and removes trajectories with $\Omega^{(n)} \ll 0$, that have negligible weight. The numerical method is described in section 13. The remainder of this chapter will focus on the most commonly treated case of unweighted, Gaussian, delta-correlated noise.

2.2. Stochastic calculus

In the case of delta-correlated noise, the trajectories are not differentiable. As a result, there are two main variants of stochastic calculus used to define the derivatives, called Ito or Stratonovich [1,46], and xSPDE can be used for either type. The default algorithms are designed for Stratonovich cases, since this is just ordinary calculus. Ito calculus can be treated also, either using the directly applicable Euler method, or else by appropriate transformations to a Stratonovich form. One can also have a time-reversed or implicit Ito calculus [28], which is directly solved using an implicit Ito-Euler method.

A single step in time of duration Δt uses finite noises $\boldsymbol{\xi}$ which are defined to be delta-correlated in the small time-step limit, so that $\langle\xi_i\xi_j\rangle = \delta_{ij}/\Delta t$.

2.2.1. Types of stochastic calculus

The limits as $\Delta t \to 0$ are taken differently for the different types of stochastic calculus. Let $\mathbf{a}_0 = \mathbf{a}(t_0)$, $t_1 = t_0 + \Delta t$, $\mathbf{a}_1 = \mathbf{a}(t_1)$, $\bar{\mathbf{a}} = (\mathbf{a}_1 + \mathbf{a}_0)/2$, and $\bar{t} = t + \Delta t/2$, then the next step in time is:

- Ito calculus - uses **initial-time** derivative evaluations

$$\mathbf{a}_1 = \mathbf{a}_0 + \left[\mathbf{A}^{(I)}\left(\mathbf{a}_0, t_0\right) + \underline{\mathbf{B}}\left(\mathbf{a}_0, t_0\right) \cdot \boldsymbol{\xi}\right] \Delta t \,. \tag{2.8}$$

- Stratonovich calculus - uses **midpoint** derivative evaluations

$$\mathbf{a}_1 = \mathbf{a}_0 + \left[\mathbf{A}\left(\bar{\mathbf{a}}, \bar{t}\right) + \underline{\mathbf{B}}\left(\bar{\mathbf{a}}, \bar{t}\right) \cdot \boldsymbol{\xi}\right] \Delta t \,. \tag{2.9}$$

- Backward Ito calculus - uses **final-time** derivative evaluations

$$\mathbf{a}_1 = \mathbf{a}_0 + \left[\mathbf{A}^{(I+)}\left(\mathbf{a}_1, t_1\right) + \underline{\mathbf{B}}\left(\mathbf{a}_1, t_1\right) \cdot \boldsymbol{\xi}\right] \Delta t \,. \tag{2.10}$$

The drift term **A** is changed in Ito or implicit Ito calculus, if the noise coefficient B depends on the stochastic variable. Defining $\partial_n \equiv \partial/\partial a_n$ and using the Einstein convention of summing over repeated indices, one has the following relationships:

$$\begin{aligned} A_i^{(I)} &= A_i + \frac{1}{2} B_{jk} \partial_j B_{ik}, \\ A_i^{(I+)} &= A_i - \frac{1}{2} B_{jk} \partial_j B_{ik}. \end{aligned} \tag{2.11}$$

Methods used for solving stochastic equations depend on the type of stochastic calculus. The default methods used in xSPDE are for Stratonovich calculus. Other methods are available as well, for both forward and backward Ito calculus. Alternatively, one can use the conversion formulae to change the equation.

2.3. Example: random walk

The first example of an SDE is the simplest possible stochastic equation or Wiener process:

$$\dot{a} = w(t) \,. \tag{2.12}$$

This has the solution that

$$a\left(t\right) = a\left(0\right) + \int_0^t w\left(\tau\right) d\tau, \tag{2.13}$$

which means that the initial mean value does not change in time:

$$\left\langle a\left(t\right)\right\rangle = \left\langle a\left(0\right)\right\rangle . \tag{2.14}$$

2.3.1. Variance solution

The noise correlation is non-vanishing from Eq (2.2), so the variance must increase with time:

$$\begin{aligned} \left\langle a^2\left(t\right)\right\rangle &= \left\langle a^2\left(0\right)\right\rangle + \int_0^t \int_0^t \left\langle w\left(\tau\right) w\left(\tau'\right)\right\rangle d\tau d\tau' \\ &= \left\langle a^2\left(0\right)\right\rangle + \int_0^t \int_0^t \delta\left(\tau - \tau'\right) d\tau d\tau'. \end{aligned} \tag{2.15}$$

Integrating the delta function gives unity, which means that the second moment and the variance both increase linearly with time:

$$\begin{aligned}\left\langle a^2(t)\right\rangle &= \left\langle a^2(0)\right\rangle + \int_0^t d\tau \\ &= \left\langle a^2(0)\right\rangle + t. \end{aligned} \tag{2.16}$$

The probability follows an elementary diffusion equation:

$$\frac{\partial P}{\partial t} = \frac{1}{2}\frac{\partial^2 P}{\partial a^2}\,, \tag{2.17}$$

which is an example of Eq (2.27). From this equation and using Eq (2.29), the first two corresponding moment equations in this case are

$$\begin{aligned}\frac{\partial}{\partial t}\langle a\rangle &= \left\langle \frac{1}{2}\frac{\partial^2}{\partial a^2}a\right\rangle = 0 \\ \frac{\partial}{\partial t}\left\langle a^2\right\rangle &= \left\langle \frac{1}{2}\frac{\partial^2}{\partial a^2}a^2\right\rangle = 1. \end{aligned} \tag{2.18}$$

These differential equations are satisfied by the solutions obtained directly from the stochastic equations, namely Eq (2.14) and Eq (2.16).

2.4. Interaction picture

The interaction picture allows one to eliminate linear terms in the time derivatives. It is especially useful for stochastic partial differential equations, but it is applicable to stochastic equations as well. Suppose there are linear terms $\underline{\mathbf{L}}$, so that $\mathbf{A}(\mathbf{a},t) = \mathbf{A}_1(\mathbf{a},t) + \underline{\mathbf{L}}\cdot\mathbf{a}$, where $\underline{\mathbf{L}}$ is a constant matrix. The interaction picture defines local variables $\tilde{\mathbf{a}}$ for the fields $\mathbf{a}$.

It is convenient to introduce an abbreviated notation as:

$$D(\mathbf{a}) = \mathbf{A}_1(\mathbf{a},t) + \underline{\mathbf{B}}(\mathbf{a},t)\cdot\mathbf{w}(t), \tag{2.19}$$

so that one can write the differential equation as:

$$\frac{\partial \mathbf{a}}{\partial t} = D(\mathbf{a}) + \underline{\mathbf{L}}\cdot\mathbf{a}. \tag{2.20}$$

2.4.1. Linear propagator

Next, we define a linear propagator. This is given formally by:

$$\underline{\mathbf{P}}\left(t,\tilde{t}\right) = \exp\left(\left(t-\tilde{t}\right)\underline{\mathbf{L}}\right). \tag{2.21}$$

where $\tilde{t}$ is the interaction picture origin. Transforming the field $\mathbf{a}$ to an interaction picture is achieved on defining:

$$\tilde{\mathbf{a}} = \underline{\mathbf{P}}\left(\tilde{t}, t\right)\mathbf{a}. \tag{2.22}$$

As a result, the equation of motion is:

$$\begin{aligned}\frac{\partial \tilde{\mathbf{a}}}{\partial t} &= \underline{\mathbf{P}}\left(\tilde{t}, t\right) D\left(\underline{\mathbf{P}}\left(t, \tilde{t}\right)\tilde{\mathbf{a}}, t\right) \\ &\equiv \tilde{D}\left(\tilde{\mathbf{a}}, t\right).\end{aligned} \tag{2.23}$$

This removes linear terms, which can cause stiffness in the equations, increasing the discretization error. Given the case of a completely linear ODE or SDE, the trajectory solutions will be exact up to round-off errors.

2.5. Stochastic equations with jumps

Many stochastic equations involve a discrete Poisson or jump process, which xSPDE can also solve. These are common in many fields, from financial modeling to open quantum systems. The fundamental noise is then a discrete jump or Poisson process, dN, which in our applications has the integer values 0 or 1.

Including this, a combined jump-diffusion Ito SDE can be written [49]:

$$\Delta \boldsymbol{a} = \left[\mathbf{A}\left(\mathbf{a}, t\right) + \underline{\mathbf{B}}\left(\mathbf{a}, t\right)\cdot\boldsymbol{\xi}(t)\right]\Delta t + \underline{\boldsymbol{C}}\left(\mathbf{a}, t\right)\cdot\Delta\boldsymbol{N}_{\lambda}(t), \tag{2.24}$$

where the $i-th$ jump process intensity is $\lambda_i\left(\mathbf{a}, t\right)$. This is defined such that:

$$\lambda_i\left(\mathbf{a}, t\right) = \lim_{\Delta t\to 0}\frac{1}{\Delta t}P(\Delta N_{i\lambda}(t) = 1). \tag{2.25}$$

Defining this general equation requires two additional parameters, $\underline{\boldsymbol{C}}\left(\mathbf{a}, t\right)$ and $\boldsymbol{\lambda}\left(\mathbf{a}, t\right)$, to specify the jump rate and its effect on the independent variable $\mathbf{a}$. Such equations are treated in the mathematics and numerical literature, but quantum physics applications require a variable jump rate, which is sometimes ignored.

Such equations are often specified in the Ito picture in the usual mathematical literature. The fact that $\underline{\boldsymbol{C}}\left(\mathbf{a}, t\right)$ and $\boldsymbol{\lambda}\left(\mathbf{a}, t\right)$ can depend on the field $\mathbf{a}$ means that the equations do not follow standard calculus, just as with continuous SDE equations.

2.6. Probability distributions

Stochastic equations generate trajectories distributed with a probability density $P\left(\mathbf{a}\right)$. These can be defined as an average and hence can be evaluated stochastically, since:

$$P\left(\mathbf{a}'\right) = \left\langle\delta\left(\mathbf{a}' - \mathbf{a}\right)\right\rangle. \tag{2.26}$$

Here $\langle ..\rangle \equiv \langle ..\rangle_\infty$ is the infinite ensemble limit of the average over many trajectories. The probability can be shown to follow a Fokker-Planck equation (FPE) with positive semi-definite diffusion matrix, [1,50]:

$$\frac{\partial P}{\partial t} = \mathcal{L}P = \left[-\partial_n A_n^{(I)} + \frac{1}{2}\partial_n \partial_m B_{nk} B_{mk} \right] P \,, \tag{2.27}$$

where the differential operators act on all terms to their right.

2.6.1. Distribution averages

The average of any observable $\mathbf{O}(\mathbf{a})$ is obtained either by averaging over the stochastic trajectories numerically, or by analytic calculations, using:

$$\langle \mathbf{O} \rangle \;=\; \int \mathbf{O}(\mathbf{a})\, P(\mathbf{a})\, d\mathbf{a}. \tag{2.28}$$

The dynamics of an observable or moment follows an adjoint equation, where $\tilde{\mathcal{L}}$ is the adjoint of $\mathcal{L}$:

$$\left\langle \frac{\partial \mathbf{O}}{\partial t} \right\rangle = \left\langle \tilde{\mathcal{L}} \mathbf{O} \right\rangle , \tag{2.29}$$

where:

$$\left\langle \tilde{\mathcal{L}} \mathbf{O} \right\rangle = \left\langle \left[A_n^{(I)} \partial_n + \frac{1}{2} B_{nk} B_{mk} \partial_n \partial_m \right] \mathbf{O} \right\rangle . \tag{2.30}$$

This equation allows the time-evolution of averages to be calculated analytically in simple cases, given an initial distribution. However, in more complex cases, a numerical simulation of the stochastic equations is more practical, and this can be carried out with xSPDE or other software.

2.7. Probability of a Wiener process

The Wiener process with an arbitrary noise strength has the stochastic equation:

$$\dot{a} = bw(t) \,. \tag{2.31}$$

The probability density satisfies the Fokker-Planck equation for diffusion,

$$\frac{\partial P}{\partial t} = \frac{b^2}{2} \frac{\partial^2}{\partial a^2} P \,. \tag{2.32}$$

Then, if x initially is Gaussian distributed, this has a Gaussian distribution at time t with:

$$P(a) = \frac{1}{\sqrt{2\pi\sigma^2(t)}} \exp\left[-\frac{(a - \bar{a}(t))^2}{2\sigma^2(t)} \right] . \tag{2.33}$$

Here:

$$\begin{aligned} \bar{a}(t) &= \bar{a}(0) \\ \sigma^2(t) &= \sigma^2(0) + b^2 t. \end{aligned} \tag{2.34}$$

2.7.1. Distributions of functions

Any function of the stochastic variables has a corresponding probability density. For example, the distribution of a^2 has a χ^2 distribution with a single degree of freedom, such that if $y = \left(a - \bar{a}\left(t\right)\right)^2 / \sigma^2\left(t\right)$, then:

$$P\left(y\right) = \frac{1}{\sqrt{2\pi y}} \exp\left[-\frac{y}{2}\right]. \tag{2.35}$$

Hence:

$$P\left(a^2\right) = \frac{1}{\left|a - \bar{a}\left(t\right)\right| \sqrt{2\pi\sigma^2\left(t\right)}} \exp\left[-\frac{\left(a - \bar{a}\left(t\right)\right)^2}{2\sigma^2\left(t\right)}\right]. \tag{2.36}$$

More generally, it is often not known what the exact analytic solutions are, and a numerical solution is employed. This can either use the stochastic equation directly, or the Fokker-Planck equation, although it is generally difficult to scale this to many variables or to partial differential equations,

That is why we focus on the stochastic equation approach here, which can be used to numerically calculate either the mean values or the probability distributions in general cases.

2.8. Fourier transforms

Frequency spectra have many uses, especially for understanding the steady-state fluctuations of any physical system in the presence of noise, typically either thermal or quantum-mechanical, although the noise could have other sources.

The time-domain spectral definition used here is:

$$\begin{aligned}\tilde{a}(\omega) &= \frac{1}{\sqrt{2\pi}} \int e^{i\omega t} a(t) dt \\ a(t) &= \frac{1}{\sqrt{2\pi}} \int e^{-i\omega t} \tilde{a}(\omega) d\omega.\end{aligned} \tag{2.37}$$

As a simple example, a sinusoidal oscillation in the form

$$a(t) = \cos\left(\omega_0 t\right). \tag{2.38}$$

between $t = -T/2$ and $t = T/2$ has a Fourier transform given by:

$$\begin{aligned}\tilde{a}(\omega) &= \frac{1}{2\sqrt{2\pi}} \int_{-T/2}^{T/2} \left[e^{i(\omega-\omega_0)t} + e^{i(\omega+\omega_0)t}\right] dt \\ &= \frac{T}{2\sqrt{2\pi}} \left[sinc\left(\left(\omega - \omega_0\right)\frac{T}{2}\right) + sinc\left(\left(\omega + \omega_0\right)\frac{T}{2}\right)\right].\end{aligned} \tag{2.39}$$

2.9. Forward-backward stochastic equations

This type of problem is found in financial mathematics and control theory [51], and also in a more time-symmetric form in quantum phase-space dynamics [52].

The general FB equations have the following structure, written as an integral equation to make it clear what the relevant boundary conditions are:

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{x}(t_0, \mathbf{y}(t_0)) + \int_{t_0}^{t} \left\{ \mathbf{A}^x \left[\mathbf{a}(t') \right] dt' + \underline{\mathbf{B}}^x \left[\mathbf{a}(t') \right] \cdot d\mathbf{w}(t') \right\} \\ \mathbf{y}(t) &= \mathbf{y}(t_f, \mathbf{x}(t_f)) + \int_{t}^{t_f} \left\{ \mathbf{A}^y \left[\mathbf{a}(t') \right] dt' + \underline{\mathbf{B}}^y \left[\mathbf{a}(t') \right] \cdot d\mathbf{w}(t') \right\}. \qquad (2.40)\end{aligned}$$

Here, $\mathbf{a} = \{\mathbf{x}, \mathbf{y}\}$ is a cell array including forward components $\mathbf{x}$ and backwards components $\mathbf{y}$. These have "initial" conditions in the past and the future, respectively, just as with ordinary stochastic equations. Control fields, called $\mathbf{z}$ in the literature, are included in the noise coefficients if required.

The noise terms $\mathbf{w}$ are uncorrelated real Gaussian noises:

$$\langle dw_i dw_j \rangle = \delta_{ij} dt. \qquad (2.41)$$

These are equivalent to a Fredholm integral equation, where the two fields are regarded as successive time instances of a single vector field, and are solved by iteration. The previous field, ie $\mathbf{a}^{(n-1)}$, is used to solve first for $\mathbf{x}^{(n)}$, then for $\mathbf{y}^{(n)}$, in step n, since the current value is not yet known.

That is, the algorithm is:

$$\begin{aligned}\frac{\partial \mathbf{x}^{(n)}}{\partial t} &= \mathbf{A}^x \left[\mathbf{a}^{(n-1)}(t) \right] + \underline{\mathbf{B}}^x \left[\mathbf{a}^{(n-1)}(t) \right] \cdot \mathbf{w}(t) \qquad (2.42) \\ \frac{\partial \mathbf{y}^{(n)}}{\partial t_-} &= \mathbf{A}^y \left[\mathbf{a}^{(n-1)}(t) \right] + \underline{\mathbf{B}}^y \left[\mathbf{a}^{(n-1)}(t_-) \right] \cdot \mathbf{w}(t_-) \,,\end{aligned}$$

Convergence is the responsibility of the user, and can depend on the choice of the first or $n = 0$ trajectory, defined by $firstfb$ in the code.

If a control field or "z" term is present in the drift and noise coefficients [51], this must be supplied in analytic form to the algorithm.

3. SDE Examples

3.1. Complex damped spectrum

Consider the spectrum of Eq (1.4), with a complex noise,

$$\left\langle w\left(t\right)w^{*}\left(t'\right)\right\rangle = 2\delta\left(t-t'\right), \tag{3.1}$$

Th script below solves an SDE with a complex Gaussian initial condition having $\left\langle \left|a\left(0\right)\right|^{2}\right\rangle = 1$, so it is in the steady-state initially:

$$\frac{\partial a}{\partial t} = -a + w_1(t) + iw_2(t)\,. \tag{3.2}$$

The equation is such that the initial distribution is also the equilibrium probability distribution so the numerical simulation uses a random initial equation near the equilibrium value, and a range of $t = 100$, with 640 points. Here there are two real noises.

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

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

Note that p.`transforms = 1` tells xSPDE to Fourier transform the field over the time coordinate before averaging, to give a spectrum. Both observe and transforms could be cell arrays, but the this is not needed with a single observable. The first argument v of the initial function is a random field, used to initialize the stochastic variable.

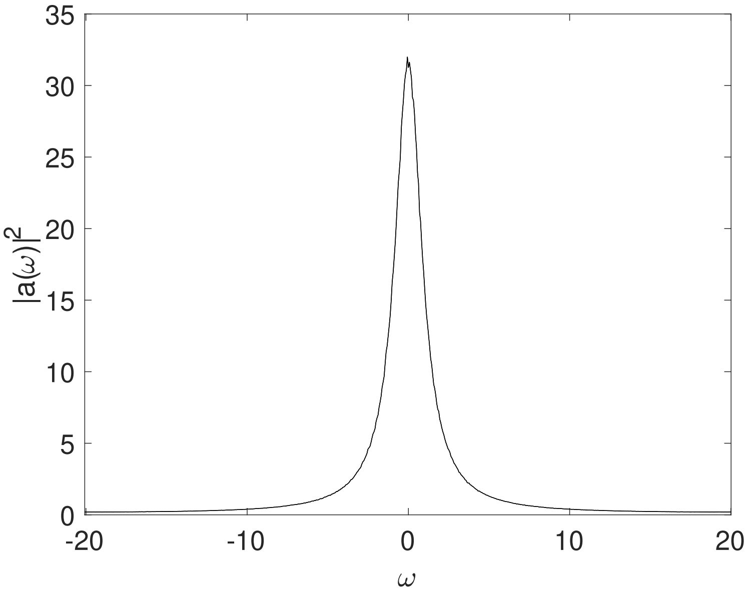


Figure 3.1.: *Complex damped spectrum obtained from the interactive script.*

To define as many observables as you like, use a cell array of observe function handles:

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

To learn more, try the following:

- **Simulate over a range of $t = 200$. What changes do you see? Why?**
- **Change the equation to the laser noise equations introduced in the next section (Laser quantum noise). Why is the spectrum much narrower?**

3.1.1. Reducing the frequency cut-off

If the number of time-points is reduced, the maximum spectral frequency is reduced. In the example below, a comparison is included, to compare with the exact result, and both the time domain and freqency domain outputs are plotted.

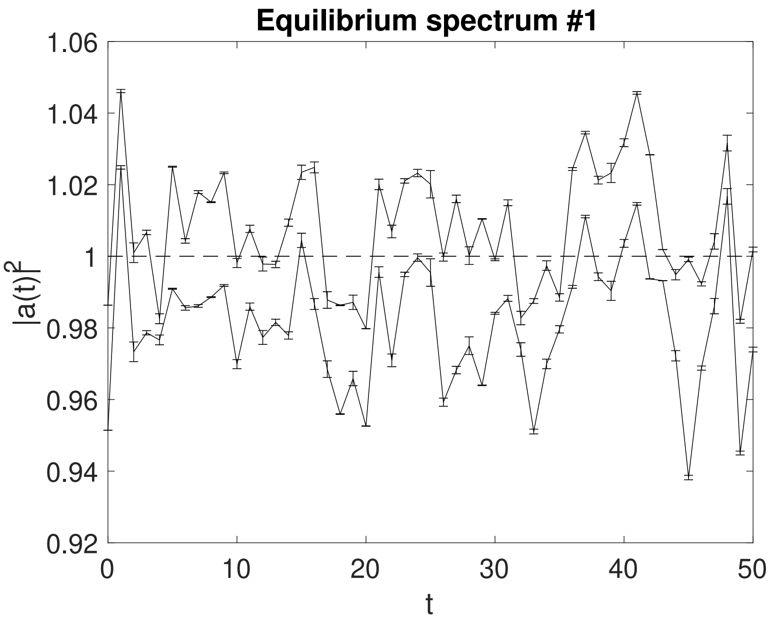
The computed ordinary and spectral variances are compared with exact solutions and graphed, where

$$\lim_{t\to\infty}\left\langle |a(t)|^2\right\rangle = 1.$$
$$\left\langle |a(\omega)|^2\right\rangle = \frac{T}{\pi(1+\omega^2)}. \tag{3.3}$$

```
function [e] = Equilibrium()
p.name = 'Equilibrium spectrum';
p.points = 50;
p.steps = 4;
p.ranges = 50;
p.seed = 241;
p.noises = 2;
p.ensembles = [100,50];
p.initial = @(w,~) (w(1,:)+1i*w(2,:))/sqrt(2);
p.deriv = @(a,w,~) -a + w(1,:)+1i*w(2,:);
p.observe{1} = @(a,~) a.*conj(a);
p.observe{2} = @(a,~) a.*conj(a);
p.transforms = {0,1};
p.olabels = {'|a(t)|^2','|a(\omega)|^2'};
p.compare = {@(p) 1, @(p)p.ranges(1)./(pi*(1+p.w.^2))};
e = xspde(p);
end
```

Notes

- A fixed random seed is input using the p.seed parameter.
- The p.transforms cell array gives a Fourier transform for p.observe{2} only.
- A small number of ensembles and time-steps is used to improve error visibility.



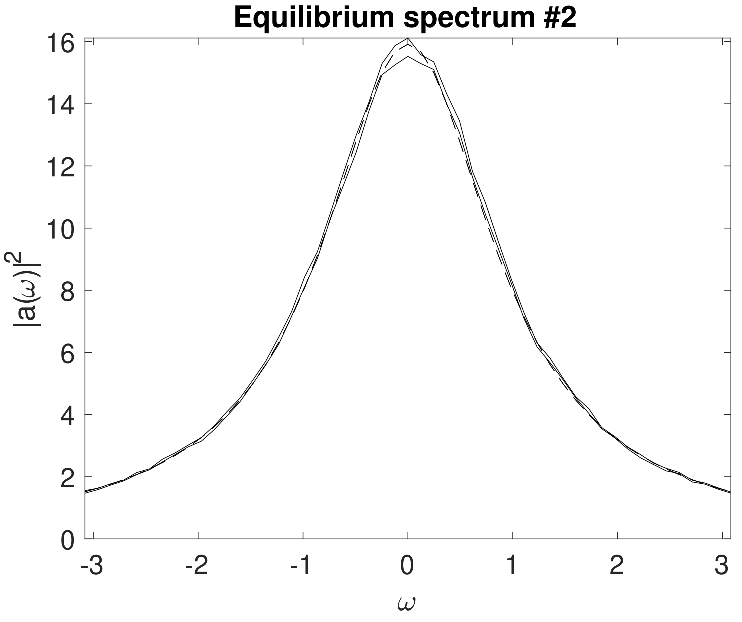


Figure 3.2.: *Top figure: Mean amplitude squared, showing invariant behavior with time, apart from sampling errors. Bottom figure: Mean spectrum as a function of frequency. The dashed lines are exact results, solid lines are upper and lower sampling error bounds ($\pm\sigma$), from sampling the stochastic equations, the error-bars are errors due to the step-size. Error bars are less than the minimum size for graphics display in the bottom figure.*

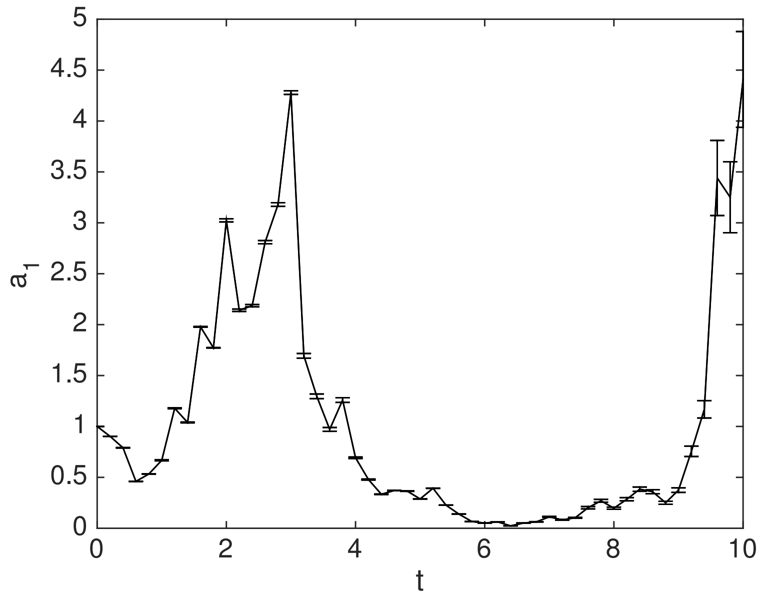


Figure 3.3.: *Simulation of the Black-Scholes equation describing stock prices.*

3.2. The Black-Scholes equation

A well-known Ito-type stochastic equation is called the Black-Scholes equation [53], used to price financial options. It describes the fluctuations in a stock or commodity value:

$$da = \mu a \, dt + a\sigma \, dw, \tag{3.4}$$

where $\left\langle dw^2 \right\rangle = dt$. As the noise is multiplicative, the equation is different in Ito and Stratonovich calculus. The corresponding Stratonovich equation, as used in xSPDE for the standard default integration routine is:

$$\dot{a} = \left(\mu - \sigma^2/2\right) a + a\sigma w(t). \tag{3.5}$$

An interactive xSPDE script in Matlab is given below with an output graph in Fig (3.3). This is for a startup with a volatile stock having $\mu = 0.1$, $\sigma = 1$. The spiky behavior is typical of multiplicative noise, and also of the more risky stocks in the small capitalization portions of the stock market.

```
clear
p.initial = @(v,p) 1;
p.deriv = @(a,w,p) -0.4*a+a.*w;
xspde(p);
```

Here *p.initial* describes the initialization function. The first argument of $@(v, p)$ 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.

To learn more, try the following:

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

3.3. Kubo oscillator

The Kubo oscillator is widely used to model environmental noise in solid-state environments. The function below solves the relevant multiplicative SDE with initial condition $a(0) = 1$ and:

$$\frac{\partial a}{\partial t} = iaw(t)\,. \tag{3.6}$$

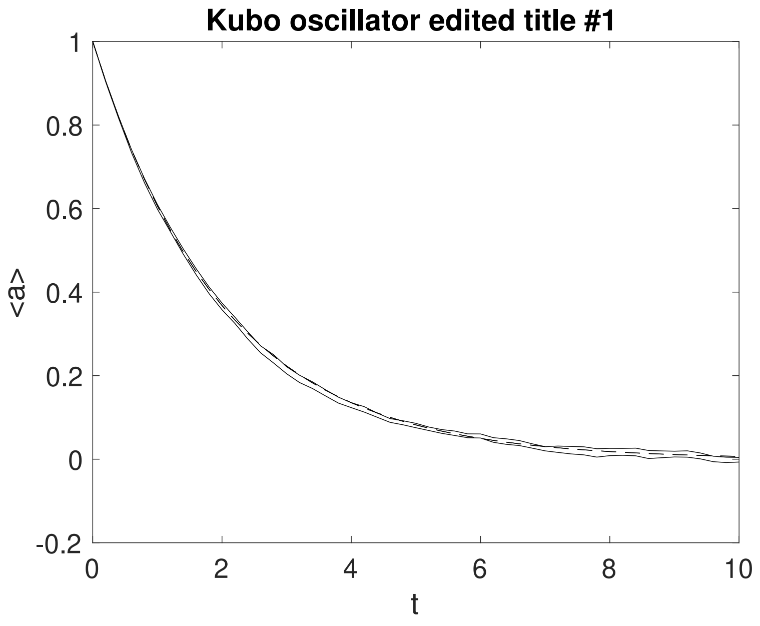
The function employs the RK4 method, although other algorithms can be used instead. It has both vector and series ensembles, then stores the computed averages with a comparison of the variance and an exact solution,

$$\langle a^n \rangle = e^{-tn^2/2}. \tag{3.7}$$

```
function [e] = Kubo()
p.name = 'Kubo oscillator';
p.ensembles = [1000,8];
p.method = @RK4;
p.initial = @(w,p) 1;
p.deriv = @(a,w,p) 1i*w.*a(1,:)  ;
p.file = 'Kubo.mat';
p.observe{1} = @(a,p) a;
p.olabels{1} = {'< a >'};
p.observe{2} = @(a,p) a.^2;
p.olabels{2} = {'< a^2>'};
p.compare = {@(p) exp(-p.t/2),@(p) exp(-2*p.t)};
e = xsim(p);
p2.name = 'Kubo oscillator edited title';
xgraph(p.file,p2);
end
```

Notes

- The algorithm is changed from the default to RK4.
- The data is stored to 'Kubo.mat'.
- This is re-read and edited using a second parameter structure, p2.



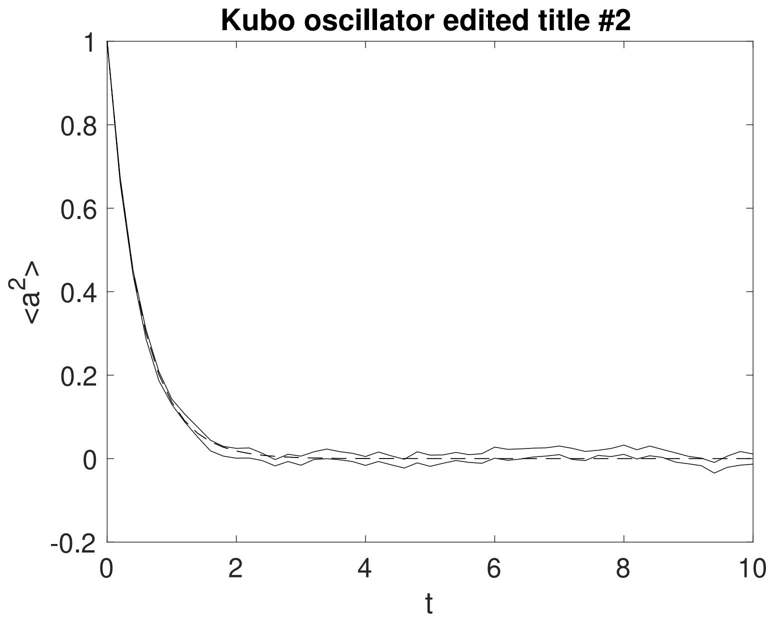


Figure 3.4.: *Example: Kubo oscillator. The graph shows the sampling error-bars as two parallel lines. The discretization error-bars are less than the minimum, and are not shown.*

Exercises

- Simulate the Kubo oscillator with three ensemble levels to allow parallel computation.
- Increase the first ensemble size to check how it modifies the sampling errors.
- Try a higher order moment as an observable.
- This function generates a data file, `kubo.mat`. If you run this twice, note that the new data overwrites the old data.
- Try including modified graphics parameters when running `xGRAPH,` if the first graphs you generate need changes.

3.4. Loss and gain with noise

This solves an SDE with a complex Gaussian distributed initial condition having $\left\langle |a\left(0\right)|^2 \right\rangle =$ 1 and a sequence of SDE equations, such that

$$\frac{\partial a}{\partial t} = \begin{cases} -a + w_1(t) + iw_2(t) & 0 < t < 4 \\ a + w_1(t) + iw_2(t) & 4 < t < 8 \end{cases}. \tag{3.8}$$

The computed variance is compared with an exact solution,

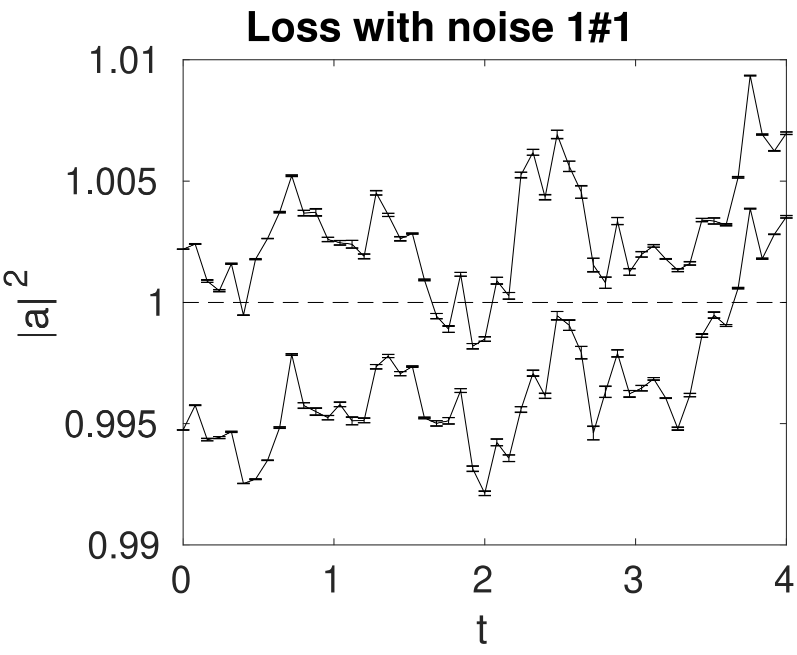
$$\left\langle a^2 \right\rangle = \begin{cases} 1 & 0 < t < 4 \\ 2e^{2(t-4)t} - 1 & 4 < t < 8 \end{cases}. \tag{3.9}$$

.

```
function [e] = Gain()
p.name = 'Loss with noise';
p.ranges = 4;
p.noises = 2;
p.ensembles = [10000,1,10];
p.initial = @(w,~) (w(1,:)+1i*w(2,:))/sqrt(2);
p.deriv = @(a,w,p) -a + w(1,:)+1i*w(2,:);
p.observe = @(a,~) a.*conj(a);
p.olabels = '|a|^2';
p.compare = @(p) 1;
p2 = p;
p2.steps = 2;
p2.name = 'Gain with noise';
p2.deriv = @(a,w,~) a + w(1,:)+1i*w(2,:);
p2.compare = @(p) 2*exp(2*(p.t-4))-1;
e = xspde(p,p2);
end
```

Notes

- Low and high level parallel ensembles optimize use of multi-core vector hardware.
- Two distinct simulations are run in series, with a change in the equation.
- The simulation name is changed in sequence 2, to distinguish the graphical outputs



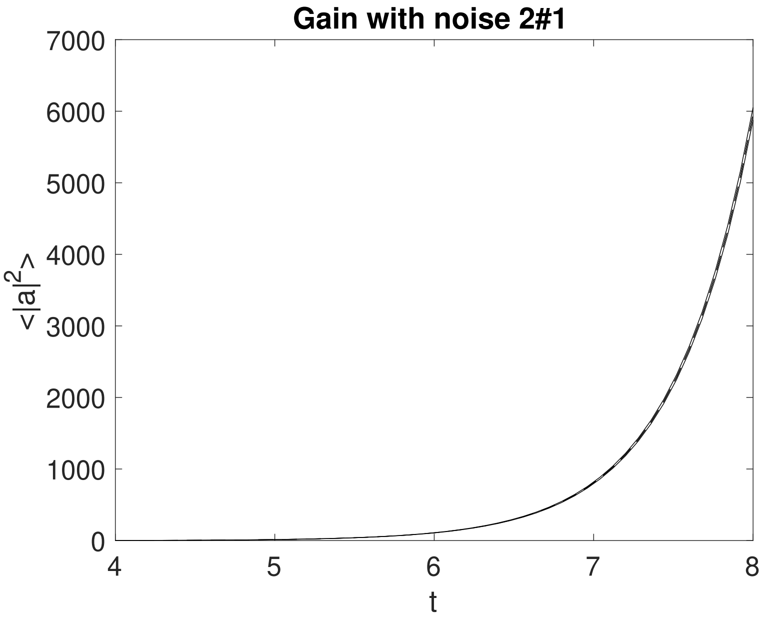


Figure 3.5.: *Top figure: amplitude squared with loss balanced by noise. Bottom figure, amplitude squared with gain. Graphs show excellent agreement with theory up to the sampling errors of less than* ± 0.005 *in the initial phase, shown by the parallel lines, with step errors of order* ± 0.001 *indicated by error-bars.*

3.5. Probability of a Wiener process

The script below solves an SDE with an initial condition $\langle a(0)\rangle^2 = \frac{1}{4}$ and

$$\dot{a} = w(t)\,. \tag{3.10}$$

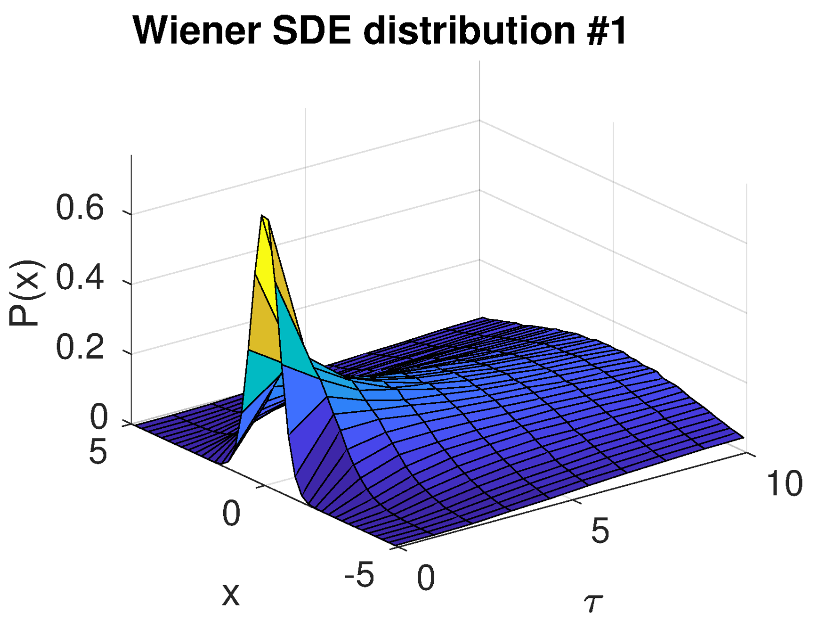
It saves the probability density and compares this with an exact solution:

$$\begin{aligned} P(x,t) &= \frac{1}{\sqrt{2\pi\sigma^2(t)}} e^{-\frac{x^2}{2\sigma^2(t)}} \\ \sigma^2(t) &= \frac{1}{4} + t\,. \end{aligned} \tag{3.11}$$

Notes

- The script outputs a 3D plot of $P(x,t)$, together with the time evolution of $P(0,t)$
- There are 5 “transverse” plots of transient probabilities at intermediate times.
- Legends are plotted to identify the simulated and the analytic comparison lines.

```
function e = Wienerprob()
p.name = 'Wiener SDE distribution';
p.noises = 1;
p.points = 10;
p.ensembles = [10000,10];
p.initial = @(v,p) v/2;
p.sig = @(p) .25 + p.r{1};
p.deriv = @(a,w,p) w;
p.observe{1} = @(a,p) a;
p.compare{1} = @gaussprob;
p.transverse{1} = 5;
p.olabels{1} = 'P(x)';
p.binranges{1} = {-5:0.25:5};
p.legends{1} = {'Sampled P(x,\tau) \pm \sigma',...
'Exact P(x,\tau)'};
p.xlabels = {'\tau','x'};
e = xspde(p);
end
%
function p = gaussprob(p)
p = exp(-(p.r{2}.^2)./(2*p.sig(p)))./sqrt(2*pi*p.sig(p));
end
```



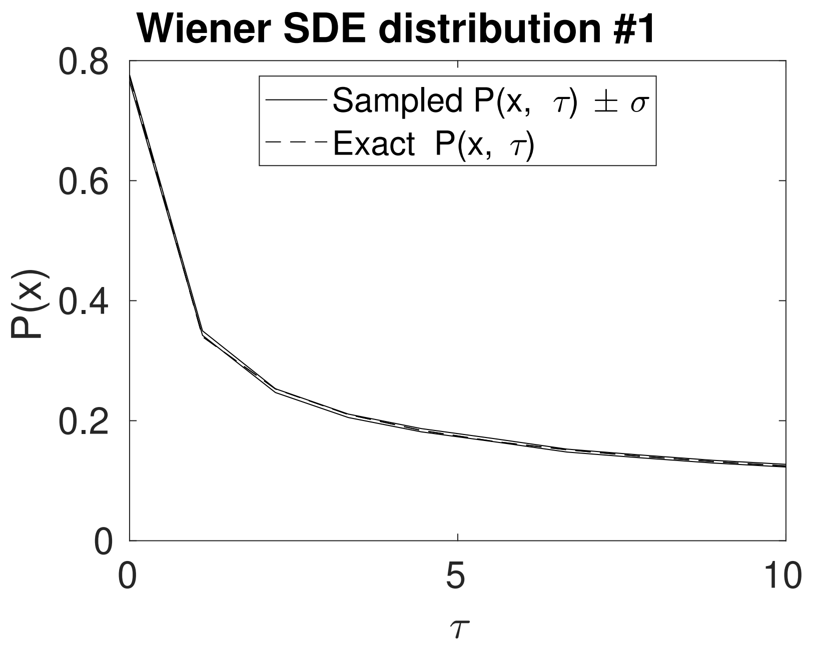
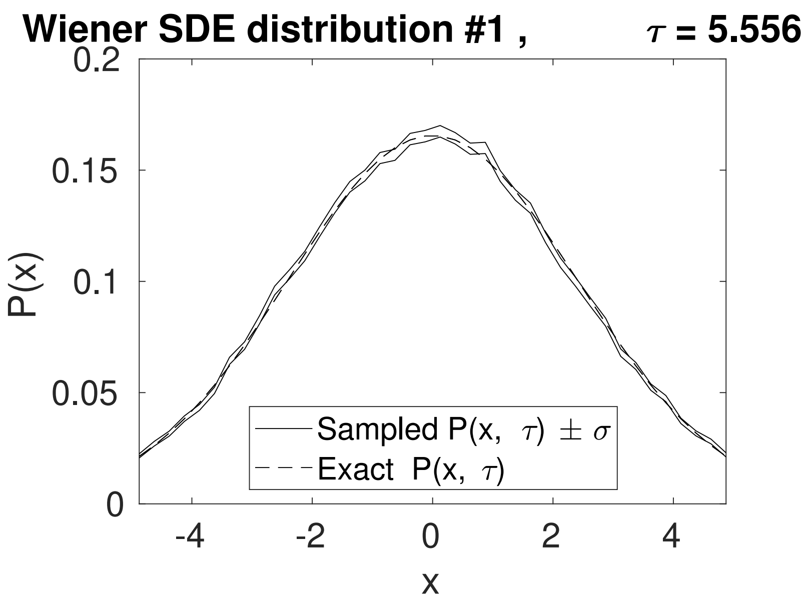


Figure 3.6.: *Top figure: 3D plot of the computed probability density of the simulated Wiener process as a function of time (τ) and "position" (x). Bottom figure: Time evolution of the computed probability density for $x = 0$. The solid lines indicate upper and lower sampling error bounds, while the dashed line indicates theoretical predictions.*



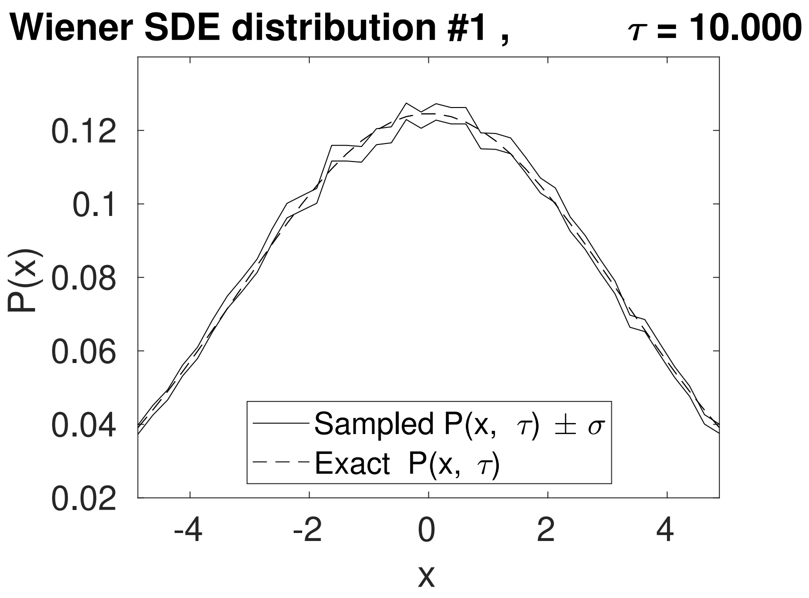


Figure 3.7.: *Top and bottom figure: Computed probability densities of the simulated Wiener process at $\tau = 5.556$ and $\tau = 10$, respectively. In total, 5 of these transverse plots are generated, however, only 2 are presented here.*

3.6. Projected SDE on a catenoid

This solves an SDE with 3 field variables $\mathbf{a} = (a_1, a_2, a_3)^T$. The Stratonovich diffusion equation is

$$\frac{\partial \mathbf{a}}{\partial t} = \mathcal{P}^{\parallel}_{\mathbf{a}}[\mathbf{w}] \,, \tag{3.12}$$

where $\mathcal{P}^{\parallel}_{\mathbf{a}}[\cdot]$ indicates a projected onto the surface of a catenoid manifold defined by

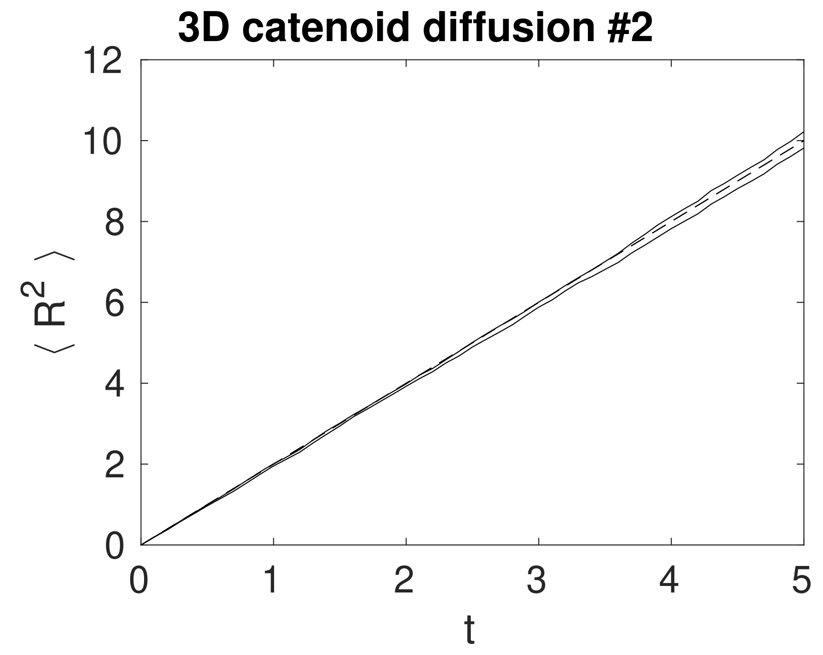
$$f \quad = x_1^2 + x_2^2 - \sinh^2(x_3) - 1 \quad = 0 \,. \tag{3.13}$$

The initial condition is given by $\mathbf{a}(0) = (1, 0, 0)^T$. Here $\mathbf{w} = (w_1, w_2, w_3)^T$ consists of 3 independent noise variables

Notes

- This is a projected sde case
- The Euclidean distance from the initial point is computed
- This is compared with the predicted analytic value $\langle R^2 \rangle = 2t$.

```
function [e] = Catenoid
p.name = '3D Catenoid diffusion';
p.X0 = [1,0,0]';
p.fields = 3;
p.ranges = 5;
p.points = 51;
p.ensembles = [400, 10];
p.compare{2} = @(p) 2*p.t;
p.deriv = @(a, w, p) w;
p.initial = @(w, p) p.X0;
p.observe{2} = @(a, p) sum((p.X0-a).^2,1);
p.diffplot{2} = 1;
p.function{1} = @(o, p) o{2}.^2;
p.olabels = {'\langle R^2 \rangle^2','\langle R^2 \rangle'};
p.project = @Catproj;
p.method = @MPnproj;
e = xspde(p);
end
```



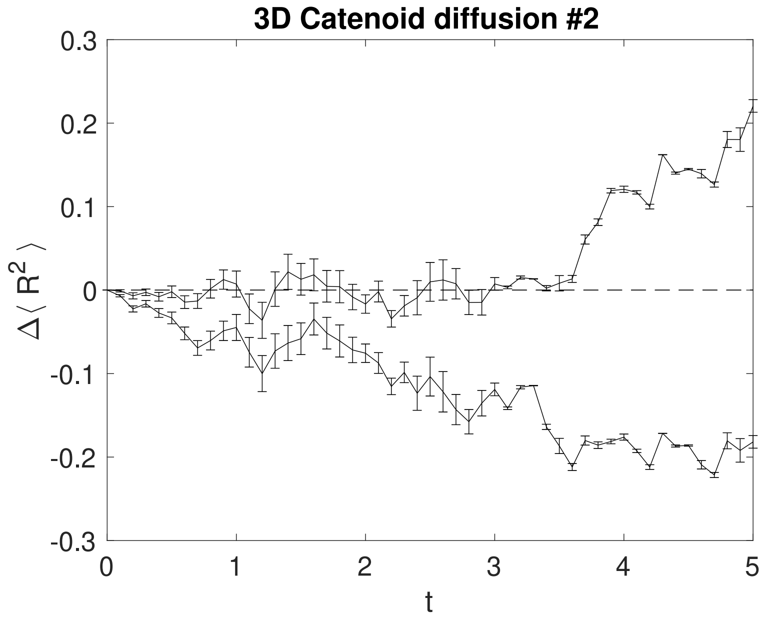


Figure 3.8.: *(Top) Tme evolution of the catenoid squared Euclidean diffusion distance* $|\mathbf{x}_0 - \mathbf{x}(t)|^2$*, where* $\mathbf{x}_0 = (1,0,0)^T$*, as a function of time. The solid lines are stochastic error bounds. The dashed line is the theoretical prediction. (Bottom) Differences between the distance* $|\mathbf{x}_0 - \mathbf{x}(t)|^2$ *and the exact result.*

3.7. Cell array coupled SDE

This solves an SDE with two different variables, one a vector and another a scalar. The equation is

$$\begin{aligned}\frac{\partial \mathbf{a}}{\partial t} &= \begin{bmatrix} 2 \\ 1 \end{bmatrix} - a + \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \times \begin{bmatrix} w(1) \\ w(2) \end{bmatrix} \\ \frac{\partial b}{\partial t} &= -b + a\,(1)\ . \end{aligned} \tag{3.14}$$

The initial condition is given by $\mathbf{a}\,(0) = (0,2)^T$, $b\,(0) = 2$. Here $\mathbf{w} = (w_1, w_2)^T$ has 2 independent noise variables. The output consists of one graph of the mean. and a second of the variance of all three variables. The b variable is effectively driven by colored noise, consisting of the finite bandwidth $a(1)$ variable. The $\boldsymbol{a}$ vector has two white noise inputs of different size.

One can prove the following are the solutions to the stochastic equations:

$$\begin{aligned} a_i\,(t) &= e^{-t}\left(a_i\,(0) + \int_0^t e^{\tau}\left[e_i + w_i(\tau)\right] d\tau\right) \\ &= e^{-t}\left(a_i\,(0) + e_i\left(e^t - 1\right) + \int_0^t e^{\tau} w_i(\tau) d\tau\right) \\ b\,(t) &= e^{-t}\left(2 + \int_0^t \left[2\left(e^t - 1\right) + \int_0^{\tau} e^{\tau'} w_1(\tau') d\tau'\right] d\tau\right) \\ &= e^{-t}\left(2\left(e^t - t\right) + \int_0^t \int_0^{\tau} e^{\tau'} w_1(\tau') d\tau d\tau'\right) \end{aligned}$$

Here, $\boldsymbol{e} = [2,1]^T$ are the two driving terms, and the resulting mean values are:

$$\begin{aligned} \bar{a}_1\,(t) &= 2(1 - e^{-t}) \\ \bar{a}_2\,(t) &= 1 + e^{-t} \\ \bar{b}\,(t) &= 2\left(1 - te^{-t}\right) \end{aligned}$$

Defining the diffusion terms as $\boldsymbol{D} = [1, 0.25]^T$, the two stochastic equation variances are:

$$\begin{aligned} \Delta a_i^2\,(t) &= e^{-2t} \int_0^t \int_0^t e^{\tau+\tau'} \left\langle w_i(\tau) w_i(\tau')\right\rangle d\tau d\tau' \\ &= e^{-2t} \int_0^t e^{2\tau} d_i d\tau = \frac{D_i}{2}\left(1 - e^{-2t}\right) \end{aligned}$$

The additional equation variance is more complex. One must solve for the a_1 variable, and use this as an external colored noise term driving the last equation. The result for the variance is as follows:

$$\begin{aligned}
\Delta b^2(t) &= e^{-2t}\int_0^t\int_0^{\tau_1}\int_0^t\int_0^{\tau_2} e^{\tau_3+\tau_4}\left\langle w_1(\tau_3)w_1(\tau_4)\right\rangle d\tau_1..d\tau_4 \\
&= e^{-2t}\int_0^t d\tau_1\int_0^t d\tau_2\left(\int_0^{\min(\tau_1,\tau_2)} e^{2\tau_3}d\tau_3\right) \\
&= \frac{1}{2}e^{-2t}\int_0^t d\tau\left(\left(e^{2\tau}-1\right)\left(t+\frac{1}{2}\right)-e^{2\tau}\tau\right) \\
&= \frac{1}{4}\left[1-\left(1+2t+2t^2\right)e^{-2t}\right]
\end{aligned}$$

```
function e = Cellarraysde()
p.fields = {2,1};
p.noises = 2;
p.initial = {@(u,v,p) [0;2],@(u,v,p) 2;};
p.ensembles = [100,100];
p.deriv{1} = @(a,b,w,p) [2;1] - a + [1,0;0,0.5]*w;
p.deriv{2} = @(a,b,w,p) - b + a(1,:);
p.observe = {@(a,b,p) [a;b],@(a,b,p) [a.^2;b.^2]};
p.output{2} = @(o,p) o{2} - o{1}.^2;
p.compare{1} = @(p) [2*(1-exp(-p.t));1+exp(-p.t);2*(1-p.t.*exp(-p.t))];
p.compare{2} = @(p) [0.5*(1-exp(-2*p.t));...
0.125*(1-exp(-2*p.t));0.25*(1(1+2*p.t+2*p.t.^2).*exp(-2*p.t))];
p.olabels{1} = '<a(i)>, <b>';
p.olabels{2} = '<[\Delta a(i)]^2>, <[\Delta b]^2>';
e = xspde(p);
```

Notes

- This uses cell arrays to define vector and scalar variables
- Cell arrays of initial and deriv functions are also required
- The output is first an observe average, then an output function.
- Two vector compare functions are used to check results
- The RMS errors in the observables are reported as a normalised error, compared to the maximum value of the corresponding output. See Section 14.5 for explanations.
- RMS errors are: Step=0.000518 Samp=0.00645 Diff=0.00896

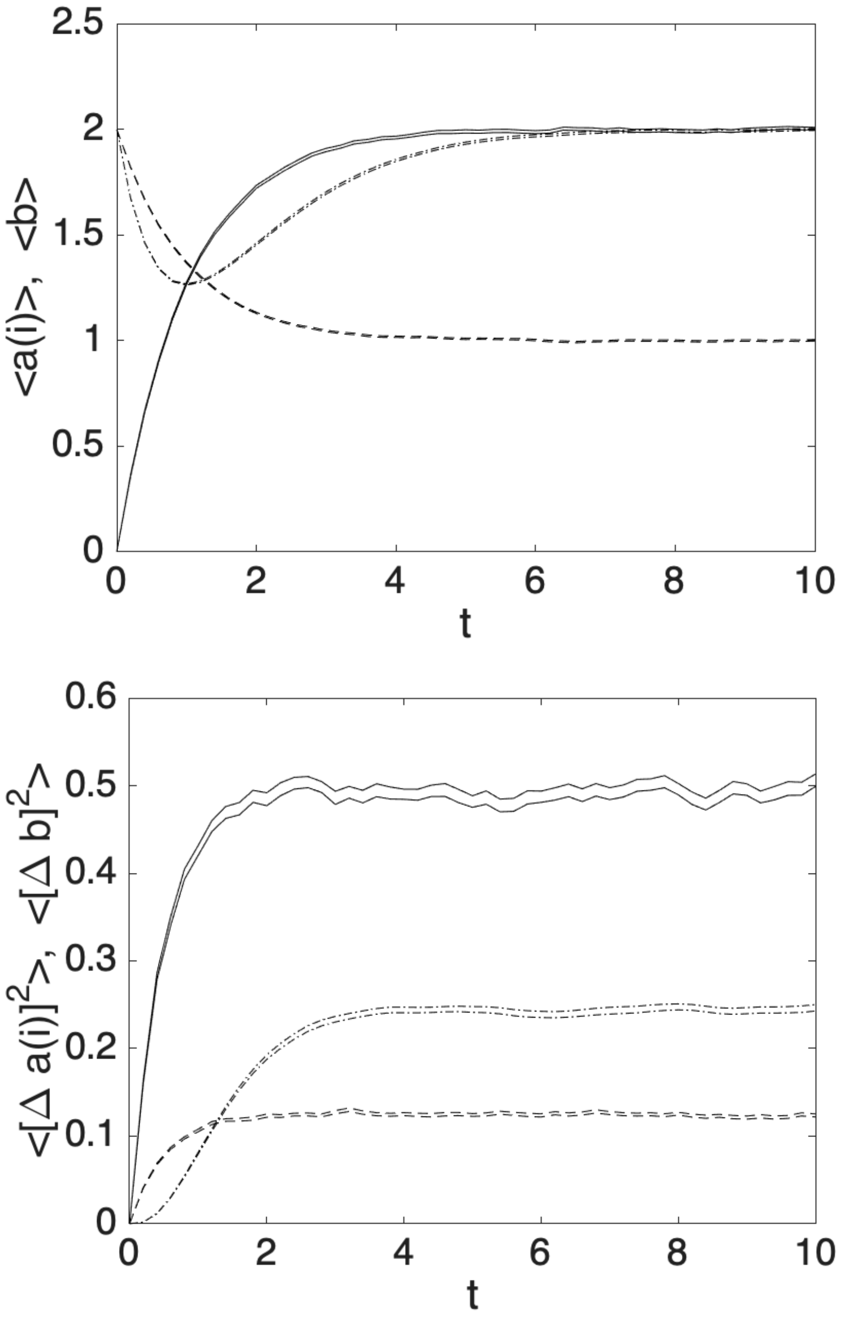


Figure 3.9.: *Time evolution of the mean (top) and variance (bottom) of each variable. The two lines indicate the stochastic error bounds. The solid lines are* $a(1)$*, the dashed lines are* $a(2)$*, the dot-dash lines are* b*. For clarity, the comparison lines are not graphed.*

3.8. Additive FBSDE

This is a soluble example of an additive FBSDE, with the equations

$$\begin{aligned} x\left(t\right) &= x_0 + \int_0^t \frac{kx}{(x-y)^2+1} d\tau + \int_0^t dw \\ y\left(t\right) &= f\left(x\left(T\right)\right) - \int_t^T \frac{ky}{(x-y)^2+1} d\tau - \int_t^T dw \end{aligned}$$

A similar example with a multiplicative noise term is treated in a standard FBSDE textbook [51].

3.8.1. Solutions and boundary values

If the boundary values are defined so that:

$$f\left(x\left(T\right)\right) = x\left(T\right)$$

then the equation has the analytic solution that $x\left(t\right) = y\left(t\right)$, and hence that the forward solution obeys the simpler forward SDE,

$$\dot{x} = kx + \xi\left(t\right)$$

where:

$$< \xi(\tau)\xi(\tau') >= \delta\left(\tau - \tau'\right)$$

This leads to the overall solution that

$$x = y = e^{kt}\left[x_0 + \int_0^t e^{-k\tau}\xi(\tau)d\tau\right]$$

and hence the mean value is given by:

$$< x\left(t\right) > =< y\left(t\right) >= x_0 e^{kt}$$

Similarly, the variance is obtained as:

$$\begin{aligned} < x^2\left(t\right) >=< y^2\left(t\right) > &= e^{2kt}\left[x_0^2 + < \int_0^t e^{-k\tau}\xi(\tau)d\tau \int_0^t e^{-k\tau'}\xi(\tau')d\tau' >\right] \\ &= e^{2kt}\left[x_0^2 + \int_0^t e^{-2k\tau}d\tau\right] \\ &= e^{2kt}\left[x_0^2 + \frac{1}{2k}\right] - \frac{1}{2k} \end{aligned}$$

3.8.2. Code with verification

```
function [e] = fbsdec()
%   e  =  FBESDEC() uses xSPDE4.2 to solve fbsde
%   Licensed by Peter D. Drummond and Margaret D Reid, (2025)

p.X           =  1;              %initial x-value
p.ranges      =  1;              %maximum real time
p.K           =  1;              %gain
p.name        =  'FBSDEc';
p.fieldsb     =  1;
p.noises      =  1;
p.inrandoms   =  0;
p.initial     =  {@(~,~,p) p.X,  @(x,~,p) x};
p.ensembles   =  [100,100,1];
p.deriv{1}    =  @(x,y,w,r)  p.K*x./((x-y).^2+1) + w;
p.deriv{2}    =  @(x,y,w,r) -p.K*y./((x-y).^2+1) - w;
p.olabels     =  {'<x>','<y>','<x^2>','<y^2>'};
p.compare{1}      =  @(p) p.X*exp(p.t*p.K);
p.compare{2}      =  @(p) p.X*exp(p.t*p.K);
p.compare{3}      =  @(p) (p.X^2+1/(2*p.K))*exp(2*p.t*p.K)-1/(2*p.K);
p.compare{4}      =  @(p) (p.X^2+1/(2*p.K))*exp(2*p.t*p.K)-1/(2*p.K);
p.observe{1}      =  @(x,y,r) x;
p.observe{2}      =  @(x,y,r) y;
p.observe{3}      =  @(x,y,r) x.^2;
p.observe{4}      =  @(x,y,r) y.^2;
p.iterfb          =  5;
p.firstfb         =  {@(p) randn(p.fbsize{1}),@(p) randn(p.fbsize{2})};
e                 =  xspde(p);
```

3.8.3. Graphs

The four graphs generated are given below.

3.8.3.1.

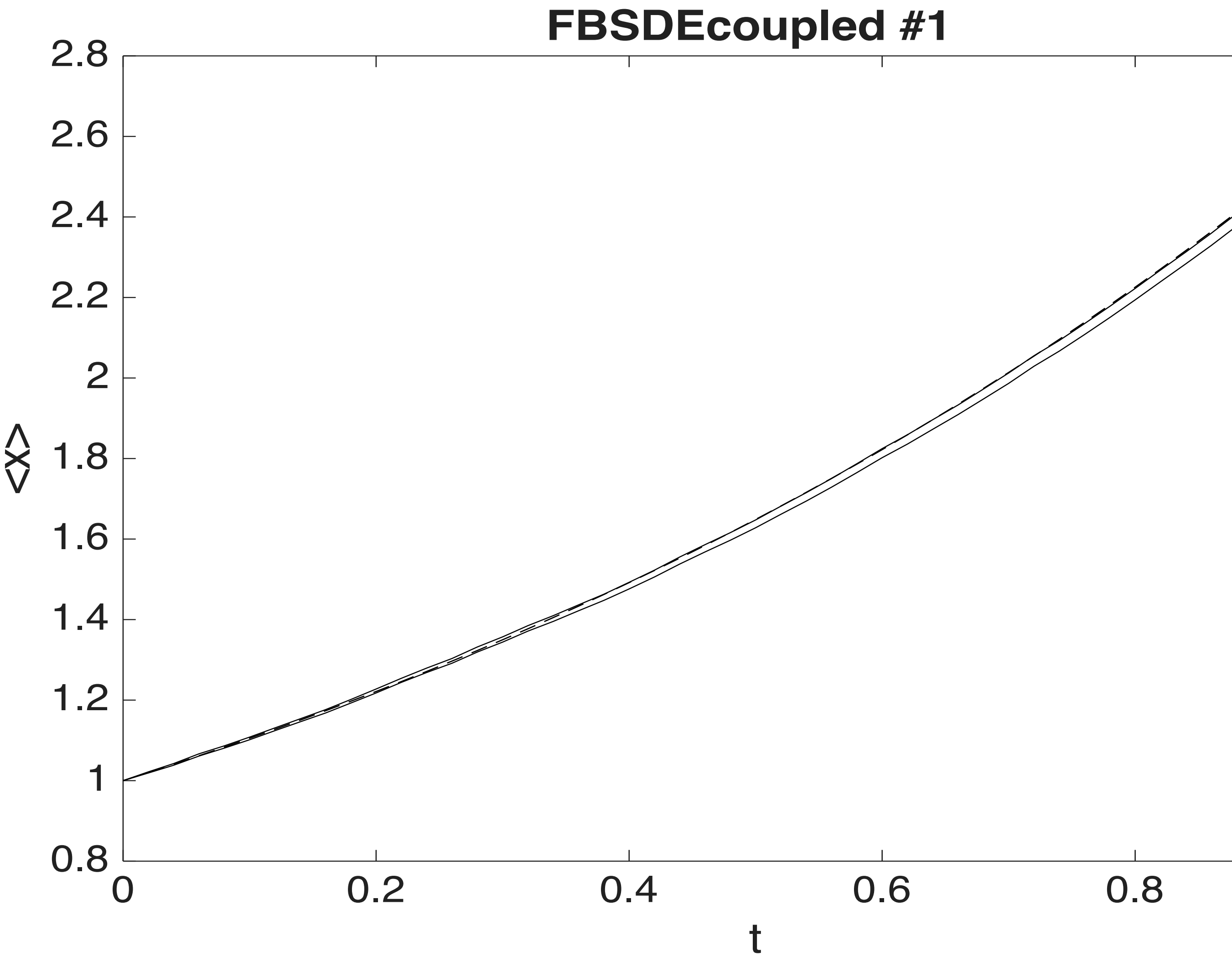


Figure 3.10.: Forward-backward SDE example

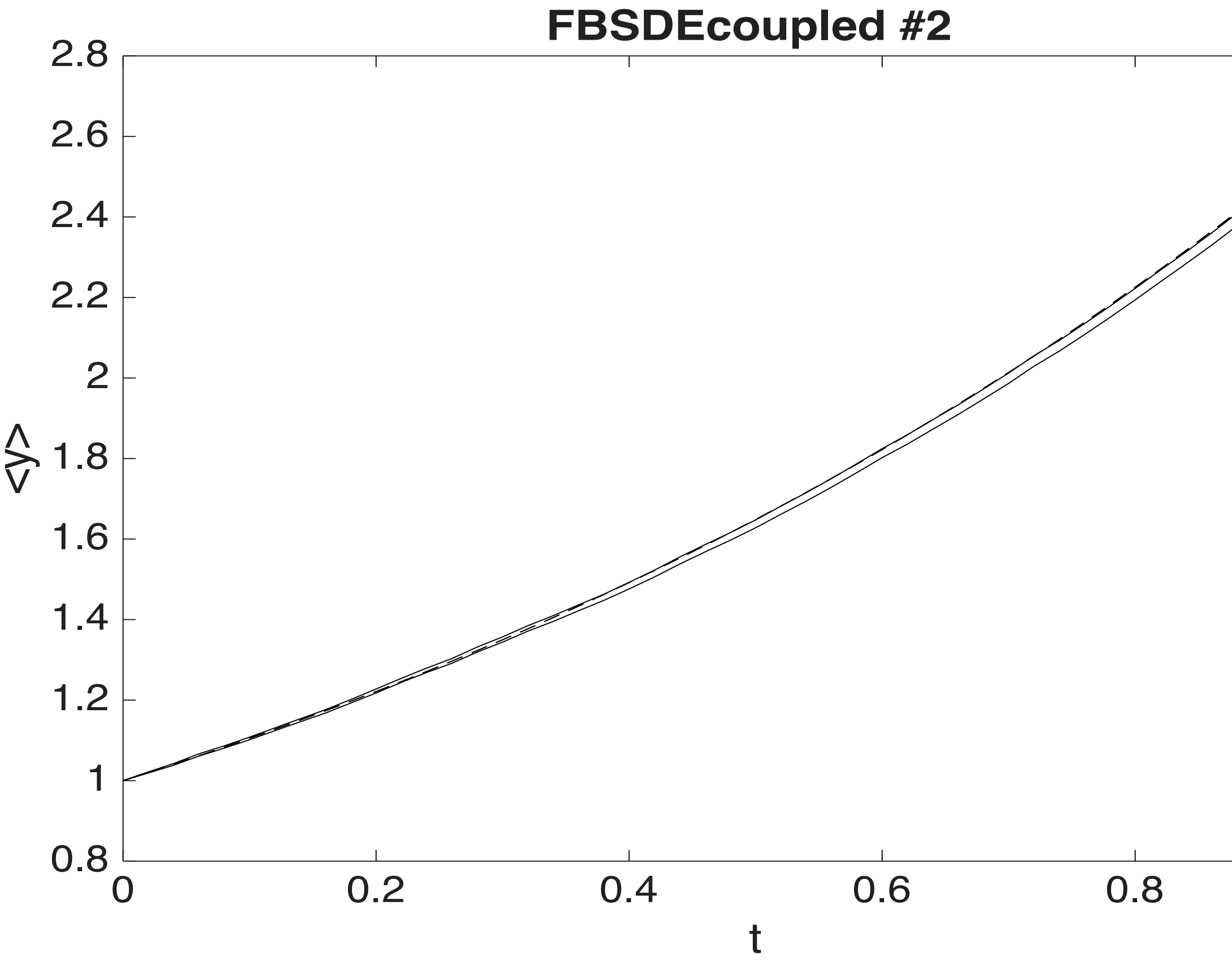


Figure 3.11.: Forward-backward SDE example

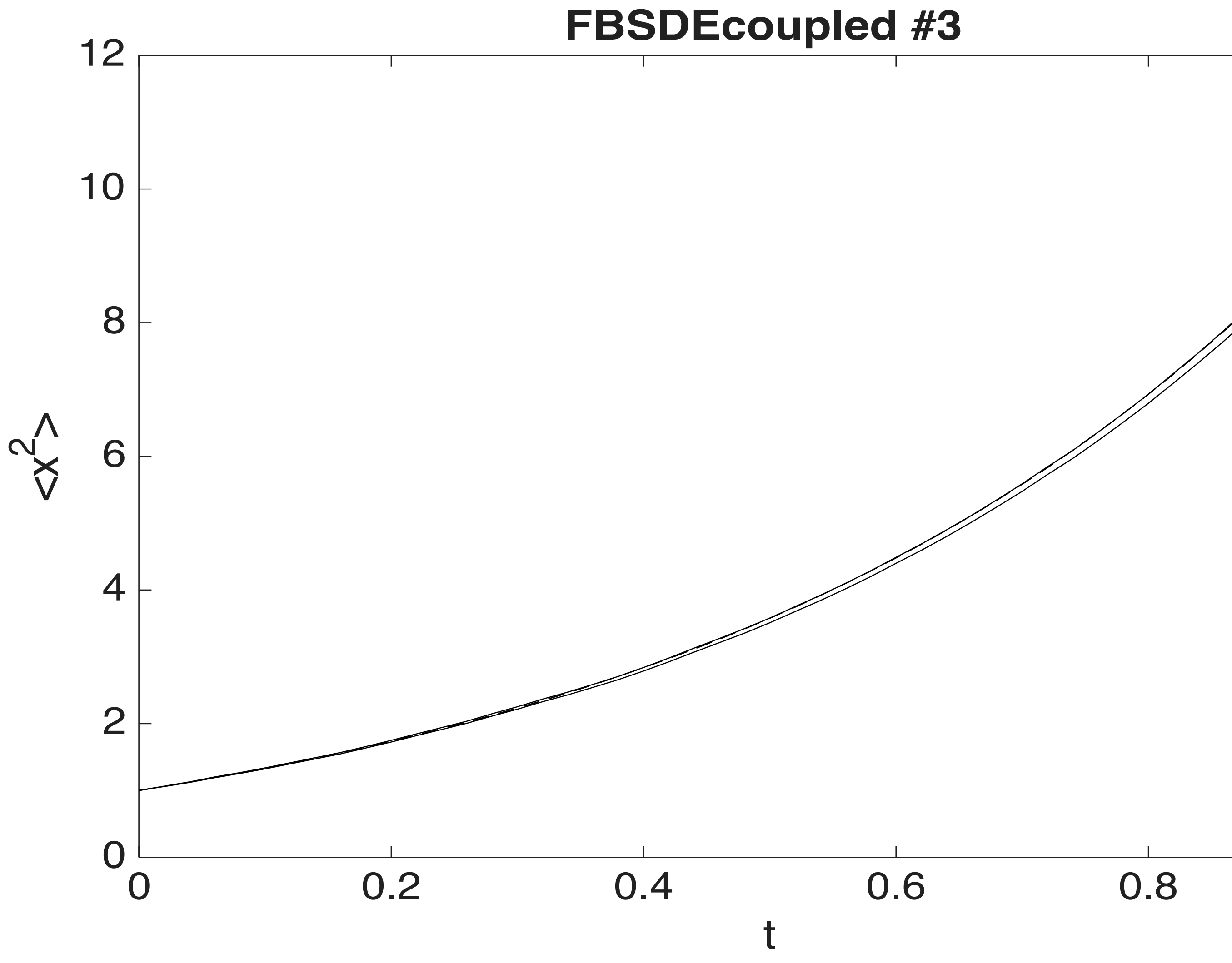


Figure 3.12.: Forward-backward SDE example

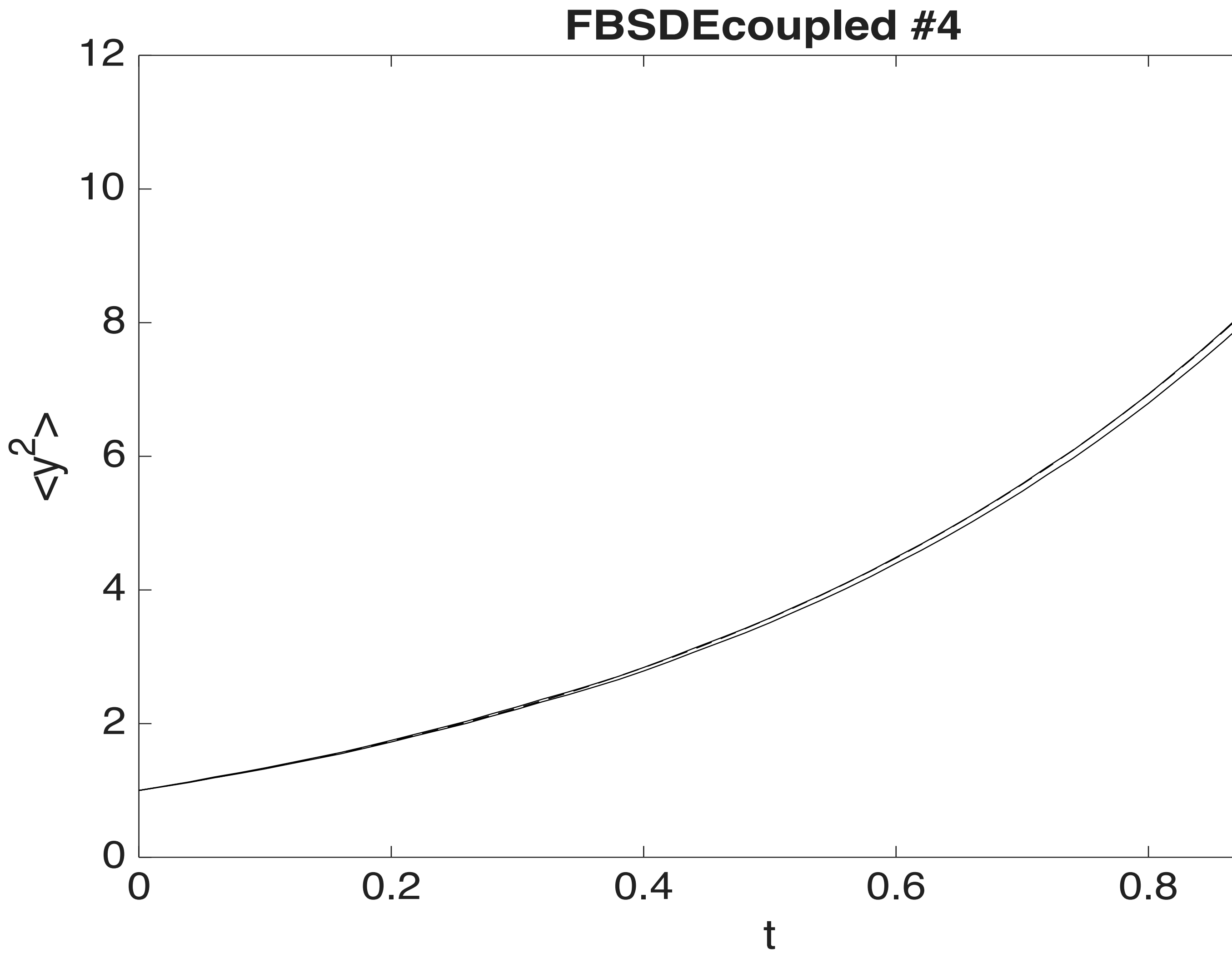


Figure 3.13.: Forward-backward SDE example

Part III.

Stochastic partial differential equations

4. SPDE toolbox

This chapter describes how to simulate a PDE or SPDE, including choosing spectral or finite difference methods and specifying boundary conditions. For theoretical background, see Chapter (5). For detailed examples, see Chapter (6).

4.1. SPDE parameters

A stochastic partial differential equation or *SPDE* for a complex vector field is defined in both time t and space dimension(s) $\mathbf{x}$. The total dimensions includes both time and space. To solve a stochastic partial differential equation xSPDE involves a similar procedure to the case of the SDE, covered in section 1.

The numerical solutions require additional parameters to define the spatial grid, and to define the linear transformations in an interaction picture, if spectral methods are used. The SPDE input parameters extend those already introduced in (1.2.1). Some new and extended parameters are listed in the table below:

Label	Type	Typical value	Description
dimensions	integer	2	Space-time dimensions
linear{c}	function	@(p) p.Dx	Linear interaction picture function
ranges	real vector	[10,10,...]	Ranges in time and space
transforms{c,d}	integer vector	$[1, 0, 1, ..]$	Space-time transform switch
points{c}	integer vector	[51,35,..]	Output lattice points in [t,x,y,z,..]
origins	real vector	[0,-5,..]	Space-time integration origin
boundaries{c,d}	integer array	$[0, 0; 0, 0; ..]$	Boundary type per field index
boundval{c,d}	cell array	$\{0, 0; 0, 0; ..\}$	Boundary value per field index
boundfun	function	@(a,c,d,p) ...	Boundary value function

Setting *dimensions* > 1 defines an (S)PDE as opposed to an ordinary (S)DE. In the xSPDE implementation, the space-time dimensions are unlimited, but large space-time dimensions become memory-intensive and slow. There is a practical limit of less than ten space-time dimensions with current digital computers, owing to exponential growth of memory and corresponding CPU time requirements at large space dimensionality.

The cell index c can be omitted in cell arguments like boundval{c,d} if there is only one field cell. Using boundval will only specify boundary values that are static in time. These can be any combinations of Dirichlet and/or Neumann/Robin. Using boundfun allows boundary values that can vary in time or are dynamic functions of the field cells.

Definitions of boundfun have four arguments, with the first one a field cell array: see (15.3).

4.2. Multidimensional Wiener process

To solve for a single four-dimensional trajectory with three space dimensions, as in Eq (5.8) , just type in:

```
p.dimensions = 4;
p.deriv = @(a,w,p) w;
xspde(p);
```

Here *p.deriv* defines the time derivative $\dot{a}$ in the input parameter structure p, while w is a delta-correlated Gaussian noise generated internally. Apart from the dimensions, there are no other parameters, so default values are used. This produces the graph shown in Fig (4.1), which gives a single trajectory using the default lattice settings.

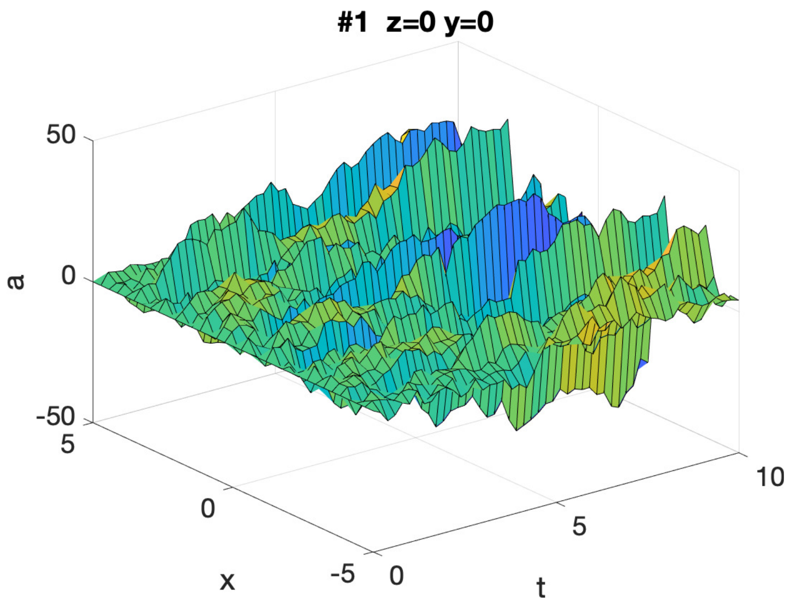


Figure 4.1.: *A multidimensional random walk of a three-dimensional field projected onto* $y = z = 0$.

For more interesting problems than this, more parameters are needed, as explained next.

4.2.1. Initial conditions

Initial conditions in the previous example have their default value of zero. For other values, they are set at the initial time of $t = O_1$ with a user-defined function p.initial, so that:

$$a(O_1) = initial(v, p) \tag{4.1}$$

The initial function includes initial random fields $v = \left[v^x, v^k\right]$. Their correlations are either delta correlated or spatially correlated. To allow this, the input parameter *randoms* is a vector such that: *randoms*(1) is the number of delta-correlated random fields, v^x, and *randoms*(2) is the number of correlated random fields, v^k. All random fields in the initial function, even if correlated using filters in momentum space, are transformed to position space before use. If there is no filtering, v^x and v^k have the same correlations.

4.2.2. Additional damping term

As another very simple example, consider the SPDE

$$\frac{\partial a}{\partial t} = -\frac{1}{4}a + x \cdot w \tag{4.2}$$

The system has one spatial dimension, or $d = 2$ space-time dimensions, one field and one noise variable. We suppose that the initial noise variance is Gaussian, with:

$$a(0, x) = 10v(x). \tag{4.3}$$

We want to consider $10,000$ stochastic trajectories per sub-ensemble with10 sub-ensembles. We will set the origin for x to 0. The variable a will be initialized as delta-correlated in space with a gaussian standard deviation on the lattice of $\sigma = 10/\sqrt{\Delta V}$. As our observable, we consider the second moment of a.

This is simulated through the following xSPDE code:

```
clear;
p.name = 'simple SPDE';
p.dimensions = 2;
p.ensembles = [10000,10];
p.origins = [0,0];
p.noises = 1;
p.initial = @(v,p) 10*v;
p.observe = @(a,~) a.^2;
p.olabels = '<a^2>';
p.deriv = @(a,w,p) -0.25*a + p.x .* w;
xspde(p);
```

With this input, Matlab produces two output graphs:

The second graph shows the time evolution for x at the mid-point, $x = 5$. The variances are larger than they would be in the SDE case, where one might expect an

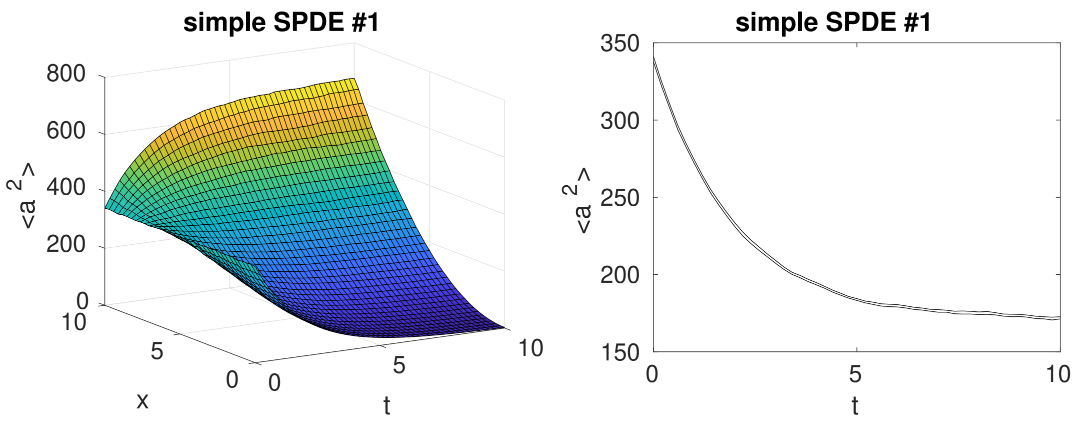


Figure 4.2.: *Example: simple SPDE output graphs.*

initial variance of $\langle a^2(0)\rangle = 100$. The reason for this is that the initial and propagating random noise fields are replaced by a lattice of noise terms with a variance of $1/\Delta V$. This causes an increase in each local noise.

4.3. Differential operators

xSPDE has finite difference and spectral methods for direct differentiation. These derivatives are obtained through function calls $D1$ and $D2$ respectively for first and second derivatives, which use a fixed grid spacing. As elsewhere, they can be replaced by user-written functions if preferred. Generally they require smaller steps in time than spectral methods, when used to define the derivative.

4.3.1. Finite difference first derivatives

The code to take a first order spatial derivative with finite difference methods is carried out using the xSPDE function D1() with arguments (o, [d, c, ind ,] p).

This takes a scalar or vector o and returns a first derivative in an axis direction d. Set d = 2 for an x-derivative, d = 3 for a y-derivative, and so on. Time derivatives are ignored at present. Derivatives are returned at all lattice locations.

If the direction d is omitted, an x-derivative is returned. The next optional input is c, the cell index, which is needed to identify the boundary conditions. If the cell index is omitted, $c = 1$ is assumed. Finally ind, which is a vector of one or more field indices can be input. If omitted, all indices are differentiated.

These derivatives can be used both in calculating propagation and in calculating observables. The boundary condition is set by the boundaries input. Any boundary of any dimension, cell or index can be made periodic, which is the default, or Neumann, or Robin/Dirichlet. Boundary values and/or a boundary function can also be input, as described in the next subsection.

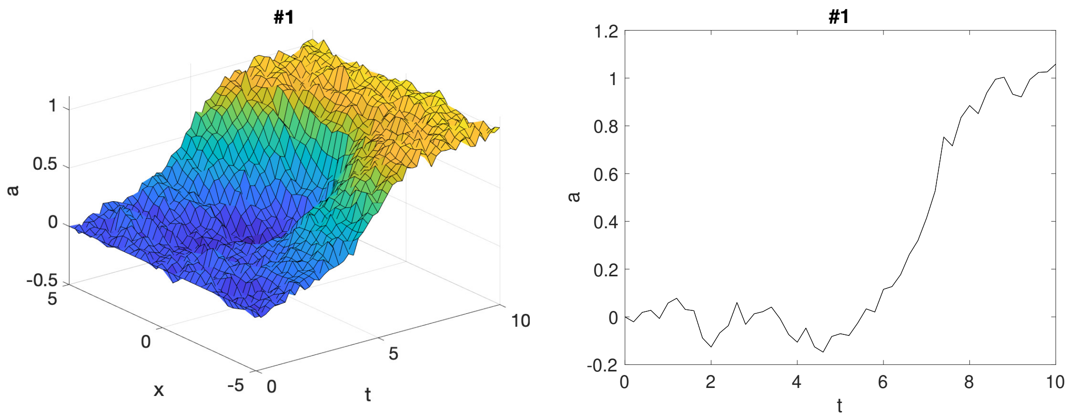


Figure 4.3.: *Two space-dimensional example graphs, direct differentiation.*

4.3.2. Finite difference second derivatives

The code to take a second order spatial derivative with finite difference methods is carried out using the xSPDE D2 function with arguments (o, [d, c, ind ,] p).

This takes a scalar or vector o and returns the second derivative in axis direction d. Set d = 2 for an x-derivative, d = 3 for a y-derivative and so on. All other properties are exactly the same as D1.

Without using the interaction picture, the stochastic equation of Eq (4.11) is specified in xSPDE using finite differences as

```
p.dimensions = 3;
p.steps = 50;
p.deriv = @(a,w,p) D2(a,2,p)+D2(a,3,p)+a - a.^3 +...
w/10;
xspde(p);
```

This gives the same result as with the linear propagator, although requiring smaller step-sizes for numerical stability, with an output graph shown in Fig (4.3). Note that the parameters and noises are slightly different!

4.3.3. Spectral derivatives

The code to take n-th order spatial derivative with spectral methods is carried out using the xSPDE DS function with arguments DS(o, [n, d, c, ind ,] p).

This takes a scalar or vector o and returns the n-th derivative in direction d. If n is omitted, a first spectral derivative is assumed. Set d = 2 for an x-derivative, d = 3 for a y-derivative and so on. All other properties are the same as D1,D2. These operators can only be used for propagation, so that they act on a field.

The spectral methods used in xSPDE use Fourier and trigonometric methods, as described next.

4.4. Spectral propagators

Using a linear spectral propagator in an SPDE can give gives better accuracy, and allows use of the interaction picture. This is included for all built-in xSPDE algorithms, provided the linear function is defined in the parameter structure. Variables $p.D\{i\}$ (with placeholders $p.Dx, p.Dy, p.Dz$ for the first 3 spatial dimensions) provide access to the operator. Higher-order derivatives are found through potentiating $p.Dx$ accordingly.

For example, the 2-dimensional Laplacian operator

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{4.4}$$

corresponds to a linear differential operator specified as:

$$p.linear = @(p)\ \ p.Dx.^2 + p.Dy^2; \tag{4.5}$$

For a comprehensive list of variables accessible through the p-structure, refer to sec. 15.8.3.

As explained in section 5.9, the general equation solved can be written in differential form as

$$\frac{\partial \mathbf{a}}{\partial t} = \mathbf{A}\left[\mathbf{a}\right] + \underline{\mathbf{B}}\left[\mathbf{a}\right] \cdot \mathbf{w}(t) + \underline{\mathbf{L}}\left[\nabla, \mathbf{a}\right]\ . \tag{4.6}$$

The linear function L can be input either inside the derivative function using finite difference operators described below, or as a separate linear function, to allow for an interaction picture in which case:

$$\underline{\mathbf{L}}\left[\nabla, \mathbf{a}\right] = \underline{\mathbf{L}}\left[\nabla\right]\mathbf{a}\,. \tag{4.7}$$

This depends on momentum space coordinates, which involves Fourier transforms so that no space dependence is allowed. It is also possible to use finite differences, in which case the derivative terms are included as part of the derivative function deriv.

The usual FFT spectral methods require periodicity, and may have either even or odd linear derivatives. The four other boundary combinations must be used with an interaction picture derivative that only has even powers of linear derivatives. Odd derivatives and nonlinear derivative terms can also be included by using finite difference derivatives in the deriv functions.

The field $p.x$ is provided by the parameter structure, and corresponds to the variable x in Eq (4.2). All parameters are preceded by the structure label, p. For two or three space dimensional problems, x, y, z are placeholders for $r\{2\}, r\{3\}, r\{4\}$, and spatial variables of even higher dimensional problems can be accessed through $r\{n\}$.

Note that where numerical labels or indices are used, the convention is that time is the first dimension.

4.4.1. One space-dimensional example

A famous partial differential equation is an exactly soluble equation for a soliton, the nonlinear Schrödinger equation (NLSE):

$$\frac{da}{dt} = \frac{i}{2}\left[\nabla^2 a - a\right] + ia\left|a\right|^2 . \tag{4.8}$$

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

$$a(t,x) \quad = \quad sech(x). \tag{4.9}$$

The spatial integral is simply:

$$\int sech(x)dx \quad = \quad \pi. \tag{4.10}$$

An xSPDE code that solves this using periodic boundary conditions is given below, together with code that compares the numerical solution with the exact solutions for the soliton and the integral:

```
p.name = 'NLS soliton';
p.dimensions = 2;
p.initial = @(v,p) sech(p.x);
p.deriv = @(a,~,p) 1i*a.*(conj(a).*a);
p.linear = @(p) 0.5*1i*(p.Dx.^2-1.0);
p.olabels = {'a(x)','\int a(x) dx'};
p.observe{2} = @(a,p) Int(a, p);
p.compare{1} = @(p) sech(p.x);
p.compare{2} = @(p) pi;
e = xspde(p);
```

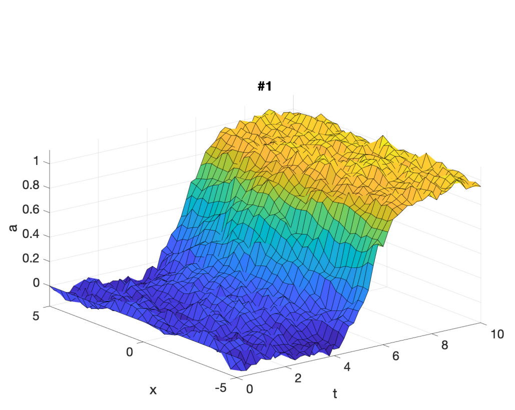
Due to finite boundaries and discrete spatial lattice, the agreement is not perfect. The errors can be reduced by increasing the range of the integration domain and improving the resolution with more points.

4.4.2. Two space-dimensional example

As another example, consider the two-dimensional nonlinear stochastic equation, with periodic boundary conditions:

$$\frac{\partial a}{\partial t} \quad = \quad \nabla^2 a\left(\mathbf{x},t\right) + a\left(\mathbf{x},t\right) - a\left(\mathbf{x},t\right)^3 + \eta\left(\mathbf{x},t\right). \tag{4.11}$$

Using the interaction picture allows for the absorption of both the Laplacian and the first-order term by the *p.linear* parameter, which results in



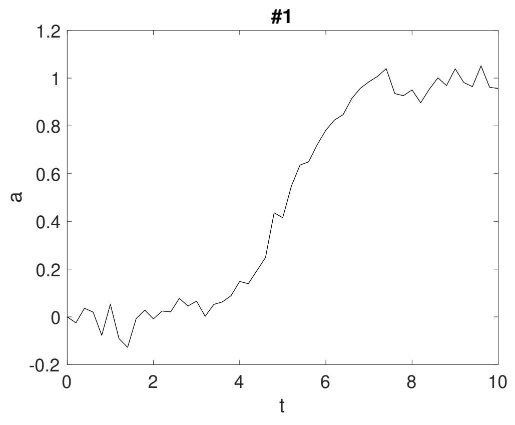


Figure 4.4.: *Two space-dimensional example graphs.*

```
...
p.linear = @(p) (p.Dx.^2+p.Dy.^2) + 1;
p.deriv = @(a,w,~) -a.^3 + w;
xspde(p);
```

With this input, Matlab produces two output graphs as shown in Fig (4.4):

4.5. Transverse lattice

4.5.1. SPDE spatial lattice

Stochastic fields in an SPDE are stored in a cell array that can include one or more real or complex arrays, $a(f, \mathbf{i}, e)$. Here f is the internal field index, $\boldsymbol{i}$ is a $d-1$ dimensional spatial lattice index for d space-time dimensions, and e is the ensemble index. For a cell array of multiple fields, a, b, c .., these must each have either the same number of points or one point per dimension. When specifying the spatial lattice, one must define:

dimensions The dimensionality in time and space. The default is an SDE: $d = 1$.

points The number of integration points. The first cell default points is $\mathbf{N} = [51, 35, 35..]$. This can have a cell index if there are multiple field cells, provided the indices are compatible for broadcasting, eg *points* $= \{[35, 25], [35, 1]\}$, to allow different dimensions to be input. After the first or base cell dimension, the default is the previous cell dimension, with ones in the higher dimensions unless a value is already present in the previous cell.

steps The number of intermediate steps per plotted point. The default is $\mathbf{N} = [1, 1, 1..]$. This is the same for all cells, and can modify any space-time dimension.

ranges The integration ranges in each dimension. The default is $\mathbf{R} = [10, 10, 10..]$. This is the same for all cells.

origins The origins of the space-time integration domains. By default, the origin is $O(1) = 0$ for the time coordinate and $\mathbf{O} = -\mathbf{R}/2$ for the space coordinates ($\mathbf{R}$ is the *ranges* variable) such that the spatial grid is symmetric around $\mathbf{r} = 0$.

There is a restriction, which is that the first cell must have the maximum number of points in each dimension.

The spatial points in the n-th dimension are at $r\{n\} = O(n), O(n) + dx(n), \ldots O(n) + R(n)$, so the spacing for $N(n)$ points is $dx(n) = R(n)/(N(n) - 1)$. For periodic boundaries, the boundaries where fields and derivatives are equal are at $O(n) - dx(n)/2$ and $O(n) + R(n) + dx(n)/2$. For other boundary types, the boundary values are defined at the first and last points.

4.5.2. Field indices

In the functions *deriv*, *initial* and *observe*, the field and noise variables a and w have extended dimensionality compared to the 1-dimensional case, to index the transverse lattice. The indices are $a(f, \mathbf{i}, e)$, where the:

field index f corresponds to the field index for a and the noise index for w.

intermediate indices $\mathbf{i}$, which are absent in the 1-dimensional case, correspond to the spatial grid and have the same structure. For example, in the case with dimensions $= 3$, indicating one time index and two spatial dimension, $\mathbf{i}$ corresponds to the two space indices.

last index e corresponds to the stochastic trajectory.

For storing space coordinates like $p.x$, the first and last index are $f = e = 1$. Where Fourier transforms are used internally, the momentum arrays have zero momentum as the first index to follow standard discrete Fourier transform conventions. This is changed to a symmetric convention in all stored graphics data outputs that are functions of momentum space.

4.5.3. Integrals and averages

There are functions available in xSPDE for spatial grid averages and integrals, to handle the spatial grid. These are **Ave** and **Int,** which are used to calculate observables for plotting. They operate in parallel over the lattice dimensions, by taking a vector or scalar quantity, for example a single field component, and returning an average or a space integral. In each case the first argument is the field, the second argument is a vector defining the type of operation, and the last argument is the parameter structure. If there are two arguments, the operation vector is replaced by its default value.

Int(o, [dx,] p) Integrates over the spatial grid to allow calculation of global quantities. To take an integral over the spatial grid, use the xSPDE function Int with arguments (o, [dx,] p).

This function takes a scalar or vector quantity o, and returns a trapezoidal space integral over selected dimensions with vector measure dx. If $dx(j) > 0$ an integral is taken over dimension j. Dimensions are labelled from j = 1,2,3 ... as in all xSPDE standards. Time integrals are ignored at present. To integrate over an entire lattice, set dx = p.dx, which is the default value if dx is omitted, otherwise set dx(j) = p.dx(j) for selected dimensions j.

If momentum-space integrals are needed, first use the transforms switch to make sure that the field is Fourier transformed before being averaged, and **always** input dk instead of dx.

Warning: if dk is omitted, xSPDE assumes that you want to use the default of dx! This will generate incorrect results for momentum integrals.

Ave(o, [av,] p) Spatial grid averages can be used to obtain stochastic results with reduced sampling errors if the overall grid is homogeneous. An average is carried out using the built in xSPDE function Ave() with arguments (o, [av,] p).

This takes a vector or scalar field or observable, defined on the lattice, and returns an average over the spatial lattice. The input is a field a or observable o, and an optional averaging switch av. If $av(j) > 0$, an average is taken over dimension j. Space dimensions are labelled from j = 2,3... as elsewhere. If the av vector is omitted, the average is taken over all space directions.

4.6. Boundary conditions

4.6.1. Transverse boundary types

Transverse boundary conditions must be given for all partial differential equations. Common transverse boundary types are of three types: Neumann (specified derivative), periodic, or Dirichlet (specified field). These are obtained using $boundaries\{c,d\} = -1, 0, 1$, which is specified for each cell c, space dimension $d > 1$, field index i and boundary j.

If boundaries are omitted for any dimension the default is 0, which gives periodic boundaries in that dimension for all field indices, and permits the use of Fourier transforms and an interaction picture as described above.

The value of $boundaries\{c,d\}$ is a matrix whose column index (i) is the field index, and whose row index (j) is given by $j = 1, 2$ for the lower and upper boundary type respectively.

Spatial derivatives or other functions linking different spatial points can be specified either in the functionals $\boldsymbol{A}[\mathbf{a},\mathbf{r}]$, $\underline{\mathbf{B}}[\mathbf{a},\mathbf{r}]$ or else in the *linear* function, provided the derivative terms are linear functions of the fields. Use of the *linear* function allows an interaction picture algorithm, with increased efficiency. The *linear* function is available with all boundary conditions, but works best in periodic cases.

The default boundary conditions are periodic. The implicit setting of this is that periodicity is enforced such that $a\left(o_i - dx_i/2\right) = a\left(o_i + r_i + dx_i/2\right)$, which is the usual discrete Fourier transform requirement.

Otherwise, the differential equation boundaries are specified at $a\left(o_i\right)$, $a\left(o_i + r_i\right)$, using the cell-array input $boundaries\{c,d\}(i,j)$, which is defined per field cell, space dimension $(d = 2,3..)$, field index $(i = 1,2..)$ and boundary $j = (1,2)$. Here $d > 1$ is the transverse dimension, not including time, which only has an initial condition.

In summary the available boundary types are:

Neumann: For specified derivative boundaries, $boundaries\{c,d\}(i,j) = -1$

Periodic: For periodic boundaries, $boundaries\{c,d\}(i,j) = 0$

Dirichlet: For specified field boundaries, $boundaries\{c,d\}(i,j) = 1$

These are specified in a cell array: $boundaries\{c,d\}(i,1)$ sets the lower boundary type in dimension d for the i-th field component, while $boundaries\{c,d\}(i,2)$ gives the upper boundary type. Each space dimension, variable and boundary is set independently. In xSPDE, the equations are always initial value problems in time, so the time dimension boundary specification for $d = 1$ is not included.

Example: boundary types in a 2-dimensional PDE Suppose there is one field cell with two field indices, and mixed boundaries in space: Dirichlet for a lower boundary at $x = 0$, and and Neumann for an upper boundary at $x = 1$, for the first field $a(1,:)$, with the opposite combination in the second field component, $a(2,:)$, hence

```
p.boundaries{2} = [1,-1;-1,1];
```

The field cell index is $c = 1$, which can be omitted here.

4.6.2. Transverse boundary values

For non-vanishing, specified boundary conditions, the boundary values can be entered either using $boundval\{c,d\}(a,p)$, if they are constant in time, \or else, if they are dynamical, the function $boundfun(a,c,d,p)$ is specified. This returns the boundary values used for the fields or derivatives in a particular cell c and dimension $d > 1$ as an array of dimension $b(\mathbf{j},e))$, where $\mathbf{j} = i,\mathbf{k}$.

Here a is the current field cell array, $i = j_1$ is the field index, and $\mathbf{k}$ is the space index, where j_d is the index of the dimension whose boundary values are specified. For this dimension, only two values are needed: $j_d = 1,2$ for the lower and upper boundary values, which could either be field values or their derivatives. An ensemble index e is also needed if the boundary values are stochastic.

Boundary values can be a function of both the fields (a) and internal variables like the current time (t). These may have stochastic initial values at $t = 0$ which are calculated only once. In such cases the boundary values must first be initialized, so the routine $boundfun(a,c,d,p)$ is first internally initialized with time $t < origin(1)$, and with random Gaussian values in the input field a. These are delta-correlated in space, i.e., with the same definition as "inrandoms". The xSDPE program stores the returned values b for the boundaries in an internal cell array, *boundval*$\{c,d\}$, for later use if required.

The default boundary value is zero, if not specified.

4.6.3. Example: boundaries in a 2-dimensional PDE

Suppose there are two fields, and we wish to set boundary values.

We take boundary values as Dirichlet for $x = 0$ and Neumann for $x = 1$ in field variable 1, and Neumann for $x = 0$ and Dirichlet for $x = 1$ in field variable 2. Suppose the boundary values are different from the default values of $a = 0$, $\partial_x a = 0$, so that:

$$\begin{aligned} a_1\left(x=0\right) &= 1, \\ \partial_x a_1\left(x=1\right) &= a_1\left(x=1\right). \\ \partial_x a_2\left(x=0\right) &= -a_2\left(x=0\right) \\ a_2\left(x=1\right) &= -1. \end{aligned} \tag{4.12}$$

These are set in the following code:

```
p.boundfun = @mybfun
p.boundaries{2} = [1,-1;-1,1];
...
function b = mybfun(a,~,~,p)
% b = mybfun(a,c,d,p) calculates boundary values
b(1,2,:)  = a(1,end,:);
b(2,1,:)  = -a(2,end,:);
b(1,1,:)  = 1+0*a(1,end,:);
b(2,2,:)  = -1+0*a(1,end,:);
end
```

4.6.4. Transverse plots

A number of plots at equally spaced points in time can be generated through. For example, adding the line below creates 3 time-sliced plots at $t = 0, 5, 10$:

```
p.transverse{1} = 3;
```

4.7. Output transforms

For graphical output, Fourier transforms involve a sum over the lattice points using a discrete Fourier transform at the lattice points x_i, so that:

$$\tilde{a}(\omega_i, \mathbf{k}_i) = \frac{dtd\mathbf{x}}{[2\pi]^{d/2}} \sum_{j_1 \cdots j_d} \exp\left[i\left(\omega_{i_1} t_{j_1} - \mathbf{k_i} \cdot \mathbf{x_j}\right)\right] a(t_{j_1}, \mathbf{x_j}) \tag{4.13}$$

The momenta k_i have an interval of

$$dk_i = \frac{2\pi}{n_i dx_i} \tag{4.14}$$

with k_i values given for even n by:

$$k_i = \left(1 - \frac{n_i}{2}\right) dk_i, \dots \frac{n_i}{2} dk_i \tag{4.15}$$

and for odd n by:

$$k_i = \frac{1 - n_i}{2} dk_i, \dots \frac{n_i - 1}{2} dk_i \tag{4.16}$$

Once Fourier transformed, the *observe* function can be used to take any further functions or combinations of Fourier transformed fields prior to averaging. Important points to keep in mind are as follows:

- Fourier transforms are specified for the k-th observe function independently of all other functions, by specifying $transforms\{k\} = [\ell_1, \dots \ell_{d,}]$.
- Here $\ell_j = 0, 1$ is a logical switch, set to $\ell_j = 1$ if the $j - th$ dimension requires a Fourier transform, and to $\ell_j = 0$ if there is no Fourier transform.
- The internal fields $p.k\{1\}, \dots p.k\{d\}$ are available for use in making functions of momentum for use with observations.
- In propagation calculations, the momentum lattice values start with $k = 0, \dots$, following standard Matlab and FFT conventions.
- For storing and graphing, momentum lattice values are reordered to start with $k = -k_{max}, \dots$, following standard graphics and mathematical conventions.

4.8. Initial random fields

When *randoms* $\sim= 0$, an initial Gaussian random field $\mathbf{v}^x$ is generated with delta-correlations in x-space. This corresponds to a variance of $1/\Delta V$ for a lattice volume of $\Delta V = \Delta x \Delta y \Delta z$ for three space dimensions, and similarly for other cases.

When *krandoms* $\sim= 0$, an initial random field $\tilde{\mathbf{v}}^k$ is generated with delta-correlations in k-space. This can be filtered with a user-specified filter function to give $\tilde{\mathbf{v}}^{kf}$, then inverse Fourier transformed to give v^k. Both random fields are passed to the *initial* function as an extended vector $\left[v^x, v^k\right]$, for field initialization in space.

When *urandoms* $\sim= 0$, an initial field of uniform random numbers is generated for jump processes.

These can all be specified as cells of multiple random fields, and passed to the initial function in the order of [randoms,krandoms,urandoms].

For multiple fields, one may combine random fields of different dimensionality using different cell-array indices. The "missing" dimension has only one point. This has a corresponding reduced lattice volume of $\Delta V_r = \Delta x \Delta y$, if the third dimension has only a single point for this field. If all space dimensions are missing, then $\Delta V_r = 1$.

There is a user specified filter function available, to modify random fields $\tilde{v}^k$, that are delta-correlated in momentum space using a filter function, 'rfilter' so that $v_i^{kf}(\mathbf{k}) = f_i^{(r)}\left(\mathbf{v}^k(\mathbf{k})\right)$, before being used. The corresponding correlations are:

$$\begin{aligned}
\left\langle v_i^x(\mathbf{x})\, v_j^x(\mathbf{x}')\right\rangle &= \delta\left(\mathbf{x}-\mathbf{x}'\right)\delta_{ij} \sim \frac{1}{\Delta V}\delta_{\mathbf{x},\mathbf{x}'}\delta_{ij} \\
\left\langle \tilde{v}_i^k(\mathbf{k})\, \tilde{v}_j^k(\mathbf{k}')\right\rangle &= \delta\left(\mathbf{k}-\mathbf{k}'\right)\delta_{ij} \sim \frac{1}{\Delta K}\delta_{\mathbf{k},\mathbf{k}'}\delta_{ij} \\
\left\langle \tilde{v}_i^{kf}(\mathbf{k})\, \tilde{v}_j^{kf}(\mathbf{k}')\right\rangle &= \left\langle f_i^{(r)}\left(\tilde{\mathbf{v}}^k(\mathbf{k})\right) f_j^{(r)}\left(\tilde{\mathbf{v}}^k(\mathbf{k}')\right)\right\rangle .
\end{aligned} \tag{4.17}$$

On a lattice, we replace the Dirac continuous delta-function by a discrete Kronecker delta function scaled by an inverse volume element either in space (ΔV) or momentum (ΔK) . The xSPDE Fourier transforms are given by a symmetric Fourier transform, so that if we inverse Fourier-transform the $k-$space inrandoms, without filtering, then:

$$v^k(\mathbf{x}) = \frac{1}{[2\pi]^{(d-1)/2}} \int e^{i\mathbf{k}\cdot\mathbf{x}} \tilde{v}^k(\mathbf{k}) d\mathbf{k} \tag{4.18}$$

These have random initial values that are real and delta-correlated in space, so that:

$$\left\langle v^x(\mathbf{x})\, v^x(\mathbf{x}')\right\rangle = \delta\left(\mathbf{x}-\mathbf{x}'\right). \tag{4.19}$$

The corresponding noises in position space are correlated according to:

$$\begin{aligned}
\left\langle v^k(\mathbf{x})\left(v^k(\mathbf{x}')\right)^*\right\rangle &= \frac{1}{[2\pi]^{(d-1)}} \int e^{i(\mathbf{k}\cdot\mathbf{x}-\mathbf{k}'\cdot\mathbf{x}')}\left\langle \tilde{v}^k(\mathbf{k})\, \tilde{v}^k(\mathbf{k}')\right\rangle d\mathbf{k}d\mathbf{k}' \\
&= \frac{1}{[2\pi]^{(d-1)}} \int e^{i(\mathbf{x}-\mathbf{x}')\cdot\mathbf{k}} d\mathbf{k} \\
&= \delta\left(\mathbf{x}-\mathbf{x}'\right).
\end{aligned} \tag{4.20}$$

Similarly, if we don't conjugate the k-noise, then:

$$\left\langle v^k(\mathbf{x})\, v^k(\mathbf{x}')\right\rangle = \delta\left(\mathbf{x}+\mathbf{x}'\right). \tag{4.21}$$

However, if we define $\tilde{v}^c(\mathbf{k}) = \left[\tilde{v}_1^k(\mathbf{k}) + i\tilde{v}_2^k(\mathbf{k})\right]/\sqrt{2}$, then we obtain complex noise that is only delta correlated when conjugated.

$$\begin{aligned}
\left\langle v^c(\mathbf{x})\left(v^c(\mathbf{x}')\right)^*\right\rangle &= \delta\left(\mathbf{x}-\mathbf{x}'\right) \\
\left\langle v^c(\mathbf{x})\, v^c(\mathbf{x}')\right\rangle &= 0.
\end{aligned} \tag{4.22}$$

This is obtainable with the x-space noise as well, but the utility of the k-space noise is that it can be filtered to have nonlocal correlations in space if required.

4.9. Noise fields

During propagation in time, noises are Gaussian noise fields delta-correlated in space-time. They are calculated in an analogous way, except with an additional factor of $1/\sqrt{\Delta t}$ because they are delta correlated in time. They have a variance of $\sigma^2 = 1/(\Delta t \Delta V)$. Reduced dimension cells with volume V_r have a noise variance $\sigma^2 = 1/(\Delta t \Delta V_r)$.

There is a user specified scaling function available, to take random knoises w^k in momentum space that are then scaled using a filter function, 'nfilter' so that $w_i^{kf}(\mathbf{k}) = f_i^{(n)}\left(\mathbf{w}^k(\mathbf{k})\right)$, before being used:

$$
\begin{aligned}
\left\langle w_i^x(t,\mathbf{x})\, w_j^x\left(t,\mathbf{x}'\right)\right\rangle &= \delta\left(\mathbf{x}-\mathbf{x}'\right)\delta\left(t-t'\right)\delta_{ij} \\
\left\langle \tilde{w}_i^k(t,\mathbf{k})\, \tilde{w}_j^k\left(t,\mathbf{k}'\right)\right\rangle &= \delta\left(\mathbf{k}-\mathbf{k}'\right)\delta\left(t-t'\right)\delta_{ij} \\
\left\langle \tilde{w}_i^{kf}(t,\mathbf{k})\, \tilde{w}_j^{kf}\left(t',\mathbf{k}'\right)\right\rangle &= \left\langle f_i^{(n)}\left(\tilde{\mathbf{w}}^k(t,\mathbf{k})\right) f_j^{(n)}\left(\tilde{\mathbf{w}}^k\left(t',\mathbf{k}'\right)\right)\right\rangle . \qquad (4.23)
\end{aligned}
$$

When *unoises* $\sim= 0$, an initial field of uniform random numbers is generated for jump processes.

All noises can all be specified as cells of multiple noise fields, and are passed to the deriv function in the order of [noises,knoises,unoises].

5. SPDE theory

This chapter describes the basics of stochastic partial differential equation (SPDE) theory, in order to explain the background to the numerical methods.

5.1. SPDE definitions

A stochastic partial differential equation or SPDE is defined in both time t and one or more space dimensions $\mathbf{x}$. We suppose there are d total space-time dimensions. The space-time coordinate is denoted as $\mathbf{r} = \left(r^1, \dots r^d\right) = (t, \mathbf{x}) = (t, x, y, z, ...)$.

The stochastic partial differential equation solved is written in differential form as

$$\frac{\partial \mathbf{a}}{\partial t} = \mathbf{A}\left[\nabla, \mathbf{a}, \mathbf{r}\right] + \underline{\mathbf{B}}\left[\nabla, \mathbf{a}, \mathbf{r}\right] \cdot \mathbf{w}(\mathbf{r}) + \mathbf{L}\left[\nabla, \mathbf{a}, \mathbf{r}\right] \cdot \mathbf{a}. \tag{5.1}$$

Here, $\mathbf{a} = [a_1, \dots a_f]$ is a real or complex vector field, $\mathbf{A}$ is a vector function of fields and space and $\underline{\mathbf{B}}$ a matrix function. The new feature is that terms can now include the operator ∇, which is a differential term in a real space $\mathbf{x}$. The exact structure of these terms is important, and not all such equations have well-behaved solutions [54, 55].

In many common cases, the noise term $\mathbf{w}$ is delta-correlated in time and space:

$$\left\langle w_i\left(\mathbf{r}\right) w_j\left(\mathbf{r}'\right)\right\rangle = \delta\left(t - t'\right)\delta\left(\mathbf{x} - \mathbf{x}'\right)\delta_{ij}. \tag{5.2}$$

One can also have noise with a finite correlation length defined by a noise correlation function $N_{ij}\left(\mathbf{x} - \mathbf{x}'\right)$ in space so that:

$$\left\langle w_i\left(\mathbf{r}\right) w_j\left(\mathbf{r}'\right)\right\rangle = \delta\left(t - t'\right) N_{ij}\left(\mathbf{x} - \mathbf{x}'\right). \tag{5.3}$$

It is even possible to have noise with a finite correlation time. Currently, these are not directly treated in xSPDE, although user definitions of this are possible by adding a customized noise function.

Additionally, the initial field has a probability distribution. In most examples, we suppose that this initial random field distribution can be generated as a function of Gaussian distributed initial random fields $\mathbf{v}\left(\mathbf{x}\right)$, where:

$$\left\langle v_i\left(\mathbf{x}\right) v_j\left(\mathbf{x}'\right)\right\rangle = \delta\left(\mathbf{x} - \mathbf{x}'\right)\delta_{ij}. \tag{5.4}$$

However, it is also possible that the initial random fields are also not delta-correlated, so that

$$\left\langle v_i\left(\mathbf{x}\right) v_j\left(\mathbf{x}'\right)\right\rangle = R_{ij}\left(\mathbf{x} - \mathbf{x}'\right). \tag{5.5}$$

Both finite correlation length and delta-correlated noise and random terms can be used in xSPDE simulations, with finite correlation lengths defined through a Fourier transform method.

5.2. Boundary conditions

There are three types of boundaries that are commonly used. They are specified independently for each space dimension $j = 2, \ldots d$, field component $i = 1, \ldots f$, and lower or upper location $\ell = 1, 2$. Each has a specific boundary type. These are described with a numerical code bt, as:

Dirichlet (specified value, $bt = 1$): $a_i\left(r^1, r^2, \ldots \hat{r}^j_\ell, \ldots\right) = f_{ij\ell}\left(\mathbf{r}, \mathbf{a}\right)$.

Periodic ($bt = 0$): $a_i\left(r^1, r^2, \ldots \hat{r}^j_\ell, \ldots\right) = a_i\left(r^1, r^2, \ldots \hat{r}^j_{3-\ell}, \ldots\right)$.

Robin/Neumann (specified derivative, $bt = -1$): $\frac{\partial}{\partial r^j} a_i\left(r^1, r^2, \ldots \hat{r}^j_\ell, \ldots\right) = g_{ij\ell}\left(\mathbf{r}, \mathbf{a}\right)$.

The coordinates $\hat{r}^j_\ell = \left(r^j_1, r^j_2\right)$ are locations where boundary conditions are enforced. There are five types of boundary combinations of these for each dimension and field variable. Note that the boundary type can change the error stability properties of an equation.

Periodic boundaries can't be combined with other types, as this defines both boundaries:

a) periodic-periodic- P-P: "0,0"

b) Dirichlet-Dirichlet- D-D: "1,1"

c) Robin-Robin- R-R: "-1,-1"

d) Robin-Dirichlet- R-D: "-1,1"

e) Dirichlet-Robin- D-R: "1,-1"

Just as with the derivative term, each of these types can change with dimension and field component. Specified field or derivative values can be any user-defined functions of space, time, and field amplitude or simply have fixed values. Currently, all combinations of boundaries can be treated in xSPDE.

5.3. Spatial grid and boundaries

The location of the boundary at $\hat{r}^j_\ell$ is important in solving (S)PDEs, especially if high accuracy is required, or if field values at the boundary are needed.

Suppose the spatial grid spacing is Δx and the number of grid points in a particular dimension d is $points(d) = N$, then the maximum range from the first to last computed point is:

$$R = (N-1)\Delta x = ranges(d). \tag{5.6}$$

Noting that $\mathbf{r} = (t, \mathbf{x})$, and $\Delta\mathbf{r} = (\Delta t, \Delta\mathbf{x})$,this means that the space-time points for an origin vector $\boldsymbol{O}$are at:

$$r_i = O_i + (i-1)\Delta r_i. \tag{5.7}$$

There are two slightly different spatial boundary locations used in xSPDE, depending on the type of boundary conditions specified, as follows:

5.3.1. Periodic boundary

For the default case of a periodic boundary, the logical boundary location is arbitrary. The indices are arranged as though on a circle from $1 : N$. It is useful to suppose the boundary is simultaneously at $\hat{r}_1^j = r_1^j - \Delta r^j/2$ and at $\hat{r}_2^j = r_{N_j}^j + \Delta r^j/2$. Neither upper or lower logical 'boundary' is at a grid point. The effective range of the domain is $R^j + \Delta r^j$, due to this displaced boundary.

Only the values at N points are computed, and one must regard the point where the periodicity is enforced as interpolating between the last and first point.

5.3.2. Non-periodic boundary

For the case of a non-periodic boundary, including Dirichlet, Robin and Neumann boundary conditions, the indices are in a line from $1 : N$. The lower and upper lower boundaries are at $\hat{r}_1^j = r_1^j$ and at $\hat{r}_2^j = r_{N_j}^j$. In some PDE methods the logical boundaries are outside the grid boundaries, but that is not the case here. Unlike the periodic case, boundaries are enforced at the first and last point.

This is different to what is found in most trigonometric transform software, but this approach allows for a unified treatment of multiple types of algorithm. For finite difference derivatives at the boundaries, this leads to the usual result that the central difference approximation to the second derivative is of first order (in Δr) at the boundaries, while it is of second order elsewhere.

With spectral methods, the derivative boundaries are obtained with a combination of an interaction picture transform and additional polynomial terms. For simplicity, only linear, even order space derivatives are included in the linear propagator for the interaction picture (see 5.5), which means that other space derivatives must be included using finite differences.

5.4. Multidimensional walk

The simplest example of an SPDE is the multidimensional Wiener process:

$$\dot{a} = w(t, \mathbf{x}). \tag{5.8}$$

This has a solution that is identical in appearance to an SDE:

$$a(t,\mathbf{x}) = a(0,\mathbf{x}) + \int_0^t w(\tau,\mathbf{x})\, d\tau. \tag{5.9}$$

Just as for an SDE, this means that the initial mean value does not change in time:

$$\langle a(t,\mathbf{x})\rangle = \langle a(0,\mathbf{x})\rangle . \tag{5.10}$$

Since there are no spatial derivatives here, boundary values are not important. One can regard this as having periodic boundaries, which by the xSPDE conventions means that no boundary conditions are enforced - since periodic boundaries do not alter computed values when there are no derivatives.

5.4.1. Variance solution

The noise correlation is non-vanishing from Eq (2.2), so the variance must increase with time:

$$\begin{aligned}\left\langle a^2(t,\mathbf{x})\right\rangle &= \left\langle a^2(0,\mathbf{x})\right\rangle + \int_0^t \int_0^t \left\langle w(\tau,\mathbf{x})\, w\left(\tau',\mathbf{x}\right)\right\rangle d\tau d\tau' \\ &= \left\langle a^2(0,\mathbf{x})\right\rangle + \delta^{d-1}(0) \int_0^t \int_0^t \delta\left(\tau-\tau'\right) d\tau d\tau'.\end{aligned} \tag{5.11}$$

Integrating the temporal delta function gives unity. The spatial delta-function is replaced by $1/\Delta V$ in a discretized lattice calculation at points $\mathbf{x}_j$ with cell volume $\Delta V = \prod \Delta x_j$, which means that the second moment and the variance both increase linearly with time:

$$\left\langle a^2(t,\mathbf{x}_j)\right\rangle = \left\langle a^2(0,\mathbf{x}_j)\right\rangle + t/\Delta V. \tag{5.12}$$

The probability on the lattice for observing lattice field values a_j follows an elementary diffusion equation:

$$\frac{\partial P}{\partial t} = \frac{1}{2\Delta V} \sum_j \frac{\partial^2 P}{\partial a_j^2}, \tag{5.13}$$

which is an example of Eq (2.27). From this equation and using Eq (2.29), the first two corresponding moment equations in this case are

$$\begin{aligned}\frac{\partial}{\partial t}\langle a_j\rangle &= \left\langle \frac{1}{2}\frac{\partial^2}{\partial a_j^2} a_j \right\rangle = 0 \\ \frac{\partial}{\partial t}\left\langle a_j^2\right\rangle &= \left\langle \frac{1}{2\Delta V}\frac{\partial^2}{\partial a_j^2} a_j^2 \right\rangle = \frac{1}{\Delta V}.\end{aligned} \tag{5.14}$$

These differential equations are satisfied by the solutions obtained directly from the stochastic equations, but as one can see, the coupling between the lattice points provides more interesting behavior. This requires derivative terms such as Laplacians.

5.5. Interaction picture

To treat Laplacians, spectral or interaction-picture methods can be very efficient, with much lower errors and much faster run-times. This is because they do not have the stability problems of finite difference methods when treating higher-order derivatives, which allows much larger time-steps to be used.

To explain the interaction picture algorithm, SPDEs often contain terms which are linear in the field variables $\mathbf{a}$, including derivative operators acting on $\mathbf{a}$. This can be treated exactly using an *interaction picture*, which leads to dramatically reduced time-step errors and higher stability [30, 56], by using a spectral method to compute derivatives. These methods are also very useful in non-stochastic PDEs.

In summary, the interaction picture provides a means to solve for linear space-derivative terms in the propagation in an efficient way. This is based on introducing local variables $\tilde{\mathbf{a}}$ for the field variables $\mathbf{a}$. It is convenient for the purposes of describing such interaction picture methods to introduce an abbreviated notation as:

$$\mathcal{D}\left[\mathbf{a},\mathbf{r}\right]=\mathbf{A}\left[\nabla,\mathbf{a},\mathbf{r}\right]+\underline{\mathbf{B}}\left[\nabla,\mathbf{a},\mathbf{r}\right]\cdot\mathbf{w}(\mathbf{r}) \tag{5.15}$$

Hence, we can write the differential equation as:

$$\frac{\partial \boldsymbol{a}}{\partial t}=\mathcal{D}\left[\mathbf{a},\mathbf{r}\right]+\underline{\mathbf{L}}\left[\boldsymbol{\nabla}\right]\cdot\boldsymbol{a}. \tag{5.16}$$

Here $\underline{\mathbf{L}}\left[\boldsymbol{\nabla}\right]$ should include the highest order derivatives, as these have the largest eigenvalues, but lower-order derivative terms may occur in the other terms.

5.5.1. Linear propagator

Next, we define a linear propagator. This is given formally by:

$$\mathcal{P}\left(t,\tilde{t}\right)=\exp\left(\left(t-\tilde{t}\right)\underline{\mathbf{L}}\left[\boldsymbol{\nabla}\right]\right). \tag{5.17}$$

where $\tilde{t}$ is the interaction picture origin, and the notation includes setting boundary values. Transforming the field $\mathbf{a}$ to an interaction picture is achieved on defining:

$$\tilde{\mathbf{a}}=\mathcal{P}^{-1}\left(t,\tilde{t}\right)\mathbf{a}. \tag{5.18}$$

As a result, the equation of motion is:

$$\frac{\partial\tilde{\mathbf{a}}}{\partial t}=\mathcal{P}^{-1}\left(t,\tilde{t}\right)\mathcal{D}\left[\mathcal{P}\left(t,\tilde{t}\right)\tilde{\mathbf{a}},t\right]. \tag{5.19}$$

This allows an SPDE to be treated with transformations using Fourier or discrete sine/cosine transforms. Our implementation uses a diagonal linear operator L without space-dependence. The linear operator can have any derivative in the periodic case, but only even order derivatives in the Dirichlet and Neumann case.

As well as the linear term, derivatives and nonlinear functions that are not tractable with spectral methods can appear in the residual term $\mathcal{D}\left[\mathbf{a},\mathbf{r}\right]$, where they are treated using finite difference techniques. As a result, while the interaction picture does not handle all possible derivative terms, it also does not restrict them from being used elsewhere in the equations.

Other methods exist in the literature. Improved convergence properties are obtained for some problems in a spectral picture using an exact solution of a linear part of the drift term [57, 58], or stochastic noise terms [59], as well as the Laplacian terms. The xSPDE code has user-definable functions that can be adapted to include these.

5.6. Fourier transforms

It is often useful to transform a field to implement the interaction picture, or to extract nonlocal correlation properties in space. The Fourier transforms or spectrum definitions used in xSPDE are given by the symmetric Fourier transform definition:

$$\begin{aligned}\tilde{a}(\mathbf{k}) &= \mathcal{F}\left(a(\mathbf{x})\right)\\ &= \frac{1}{[2\pi]^{(d-1)/2}}\int e^{-i\mathbf{k}\cdot\mathbf{x}}a(\mathbf{x})d\mathbf{x}\,. \end{aligned} \tag{5.20}$$

The inverse Fourier transform is the function:

$$\begin{aligned}a(\mathbf{x}) &= \mathcal{F}^{-1}\left(\tilde{a}\right)\\ &= \frac{1}{[2\pi]^{(D-1)/2}}\int e^{i\mathbf{k}\cdot\mathbf{x}}\tilde{a}(\mathbf{k})d\mathbf{k}\,. \end{aligned} \tag{5.21}$$

In simulations, this is not combined with any time (or space) averaging as in the temporal Fourier transforms. The reason for this is that the interaction picture transformations must be invertible, which is the case for a point-based discrete Fourier transform.

5.6.1. Normalization

During propagation, we define temporary internal fields $A(\mathbf{k_n})$, that are normalized using FFT conventions:

$$\begin{aligned}A(\mathbf{k_n}) &= \sum_{j_2=1}^{N_2}\ldots\sum_{j_d=1}^{N_d} e^{-i\mathbf{k}_n\cdot\mathbf{x_j}}a(\mathbf{x_j})\\ a(\mathbf{x_j}) &= \frac{1}{\prod_{k=2}^{D}N_k}\sum_{n_2=1}^{N_2}\ldots\sum_{n_D=1}^{N_D} e^{i\mathbf{k_n}\cdot\mathbf{x_j}}A(\mathbf{k_n})\,. \end{aligned} \tag{5.22}$$

Otherwise, for graphical and output averages, we define Fourier transforms using physics and mathematics conventions:

$$\begin{aligned}\tilde{a}\left(\mathbf{k_n}\right) &= \prod_{d=2}^{D}\left[\frac{\Delta x_d}{\sqrt{2\pi}}\right]\sum_{j_2=1}^{N_2}\dots\sum_{j_d=1}^{N_D} e^{-i\mathbf{k}_n\cdot\mathbf{x_j}} a\left(\mathbf{x_j}\right)\\ a\left(\mathbf{x_j}\right) &= \prod_{d=2}^{D}\left[\frac{\Delta k_d}{\sqrt{2\pi}}\right]\sum_{n_2=1}^{N_2}\dots\sum_{n_d=1}^{N_D} e^{i\mathbf{k_n}\cdot\mathbf{x_j}}\tilde{a}\left(\mathbf{k_n}\right)\ . \end{aligned} \tag{5.23}$$

Note that this rescaling is consistent, because

$$\Delta x_d \Delta k_d = \frac{2\pi}{N_d}. \tag{5.24}$$

5.7. Trigonometric transforms

Taking the interaction picture approach, we now consider other types of boundary conditions, which we initially assume here are either a zero field (Dirichlet) or a zero derivative (Neumann). We will only treat cases of even order derivatives, which do not change the trigonometric function. Any odd order derivatives are taken to be included in the finite difference ($\mathcal{D}$) term.

5.7.1. Zero boundary cases

In the spectral transform method in one space dimension, with zero boundaries, one uses a trigonometric function, $T\left(kx\right) = T_1 \sin\left(kx\right) + T_2 \cos\left(kx\right)$ to expand as:

$$a_i\left(t,x\right) = \sum_n a_{i,n}(t) T(k_{i,n}x), \tag{5.25}$$

The discrete inverse transform allows evaluation at sample points x_j, in order to satisfy the boundary conditions:

$$a_{i,n}(t) = \sum_j a_i(t,x_j)\tilde{T}(k_n x_j), \tag{5.26}$$

The trigonometrical function is defined such that:

$$\partial_x^{2p} T(kx) = \left(-k^2\right)^p T(kx). \tag{5.27}$$

The propagated equation is exactly soluble for the sampled points, since for each component

$$\begin{aligned}\mathcal{L}\cdot a(t,x_j) &= \sum_{ijn} \mathcal{L} a_n(t) T(k_n x_j),\\ &= -\sum_{ijnp} L_p\left(-k_n^2\right)^p a_n(t) T(k_n x_j)). \end{aligned} \tag{5.28}$$

Hence,

$$a_n(t) = \exp\left(\sum L_p \left(-k_n^2 t\right)^p t\right) a_n(0). \tag{5.29}$$

This is an exact solution, provided the initial condition has the given expansion. This of course is usually an approximation itself, which should be checked by changing the grid. There are no other approximations made on the transverse derivative. Provided the k values are the same, this propagator is identical for all types of trigonometric and Fourier transforms.

As explained in (5.2), there are five boundary combinations that are possible in each dimension and field component. Each has a corresponding xSPDE boundary type and spectral integrator. Each boundary type is specified to depend on the space dimension and the field component, as well as having boundary values depending on time and any field value.

Currently, all can be treated in xSPDE using finite differences, and each type of boundary also has a spectral method that preserves the boundary requirement. In principle one can define the trigonometric transforms to correspond to whole symmetries whose boundary is at a grid point, as used in xSPDE, or half symmetries which are halfway between two grid points.

All spectral methods used in xSPDE make use of boundaries at a grid point, in order to compute the relevant terms, which means that there is greater compatibility with the finite difference methods, when the boundaries are at the grid points. Differential equations can also have first order terms, which currently require using either finite differences or periodic boundaries.

It is possible to compute first-order derivatives with spectral methods, but these turn sine transforms into cosine transforms. This is not compatible with trigonometric interaction picture transformations used in XSPDE. As a result, any odd-order derivative terms must be computed using finite differences in all cases , except for the periodic case, where either method can be used.

In summary, spectral transforms can all be implemented using fast FTT, discrete sine (DST) or cosine (DCT) transforms. The spectral method used is specific to the boundary type. The definitions used here correspond to the standard definitions [60,61], except for one-based indexing, normalization, and extra points at the boundaries, explained below.

5.7.2. Finite boundary values

For the case of finite boundaries, a combination of trigonometric and polynomial functions are used to expand the fields, so that:

$$\begin{aligned} a_i(t,x) &= b_i(t,x) + u_i(t,x) \\ &= b_i(t,x) + \sum_n u_{i,n}(t) T(k_{i,n}x), \end{aligned} \tag{5.30}$$

Here, the functions $b_i(t,x)$ are inhomogeneous polynomial terms specified to satisfy the non-vanishing boundary conditions such that:

$$\dot{b} = \mathcal{L} \cdot b, \tag{5.31}$$

while the trigonometric expansion simply has to satisfy the equation with zero boundaries.

5.8. Transforms and boundaries

For Dirichlet or Neumann/Robin boundaries, the following expansion can be employed in each dimension. We only describe one space dimension for simplicity with:

$$u=\sum_{n=1}^{\infty}\left[S_n\sin\left(k_nx\right)+C_n\cos\left(k_nx\right)\right]e^{\sum L_p\left(-k_n^2\right)^pt}, \tag{5.32}$$

where k_n, C_n, S_n are chosen to satisfy the initial and boundary conditions. Boundaries are taken, for the purposes of explanation, as being from $x=0$ to $x=R$. This is not the case in the actual code, which can treat arbitrary boundary locations due to the use of the optional origins input to change the origin.

Unlike fast Fourier transform (FFT) definitions, there are multiple distinct trigonometric transforms . These are generally labeled DST-(n) and DCT-(n), where $n=I..IV$. They correspond to distinct boundary combinations, as explained below.

Suppose there are N computational grid-points. For the spatial grid (1-based), this corresponds to $x_n=(n-1)\,\Delta x$, $n=1,...,N$ with $\Delta x=\frac{R}{N-1}$, so we have $x_1=0$ and $x_N=R$, as elsewhere in the manual.

In carrying out a discrete transform on N_T points, with standard trigonometric transform definitions of N_T, there are **less** transform grid points required if some boundary values are defined due to Dirichlet boundaries, hence $N_T<N$. This is because xSPDE stores the full computational range, N, with boundary values.

Sometimes one may wish to refer to the corresponding periodic Fourier transform size, N_{FT}. This is $N_{FT}=2N_T=2\,(N-1)$, except for DST-I , when it is $N_{FT}=2(N_T+1)=2\,(N-1)$.

An unnormalized inverse gives the original array multiplied by $N_{FT}/4=(N-1)/2$, where $N_{FT}=2\,(N-1)$ is the periodic size, so our definitions include a normalization of $\sqrt{2/\left(N-1\right)}$. Here N_T, the number of points in the standard DST/DCT definitions, differs from **both** the xSPDE computation grid size N that includes both boundaries, and also from the periodic size, which always includes one (periodic) boundary.

Our notation is based on standard discrete sine and cosine transform definitions. Here we use $1-$based indices throughout. For all coordinates, including these examples of discrete Fourier transforms, with an origin at $\boldsymbol{r}=0$ and an integration range of $\boldsymbol{R}$, we define:

$$\begin{aligned} r_n^d &= (n-1)\,\Delta r^d. \\ \Delta r^d &= R^d/(N^d-1). \end{aligned} \tag{5.33}$$

If we regard the transforms as having arguments of form $k_j{\cdot}r_n$, the momentum spacings given below are such that:

$$\begin{aligned}\Delta k &= \frac{\pi}{R}\\ \Delta x \Delta k &= \frac{\pi}{N-1}.\end{aligned} \tag{5.34}$$

The internal momentum definitions used in the propagator calculations are therefore different to those used in external graphs and in periodic boundary cases.

The following lists the trigonometric transforms required to obtain the transform $\tilde{u}_k$ from u_n, and vice-versa, for the four non-periodic boundary types in each dimension and field index.

5.8.1. D-D case: Discrete map (DST-I)

Let $u(0) = u_1 = 0$, and $u(R) = u_N = 0$. The discrete representation of u is:

Forward transform: DST-I

$$\tilde{u}_k = \sqrt{\frac{2}{N-1}} \sum_{n=2}^{N-1} u_n \sin\left(\pi \frac{(k-1)(n-1)}{N-1}\right). \tag{5.35}$$

Inverse transform: DST-I

$$u_n = \sqrt{\frac{2}{N-1}} \sum_{k=2}^{N-1} \tilde{u}_k \sin\left(\pi \frac{(k-1)(n-1)}{N-1}\right). \tag{5.36}$$

The forward transform does **not** require the values at the end-points of $n = 1$ and $n = N$, which are set to zero in this case. This is implicit in the sine expansion, since $\sin(\mathrm{n}\pi) = 0$. Second derivatives are proportional to $(k-1)^2$.

5.8.2. R-R case: Discrete map (DCT-I)

Let $u'(0) = 0$, and $u'(R) = 0$. The discrete representation of u is:

Forward transform: DCT-I

$$\tilde{u}_k = \sqrt{\frac{2}{N-1}} \left(\frac{1}{2}\left(u_1 + (-1)^{n-1} u_N\right) + \sum_{n=2}^{N-1} u_n \cos\left(\pi \frac{(k-1)(n-1)}{N-1}\right)\right). \tag{5.37}$$

Inverse transform: DCT-I

$$u_n = \sqrt{\frac{2}{N-1}} \left(\frac{1}{2}\left(\tilde{u}_1 + (-1)^{n-1} \tilde{u}_N\right) + \sum_{k=2}^{N-1} \tilde{u}_k \cos\left(\pi \frac{(k-1)(n-1)}{N-1}\right)\right). \tag{5.38}$$

The forward transform requires the values at the end-points of $n = 1$ and $n = N$, which are not zero in this case. It is equal (up to a factor) to a discrete Fourier transform of $2(N-1)$ real numbers u_n with even symmetry about $n = 1$ and $n = N$. As a result, the equivalent discrete derivatives at both the end-points are zero. Second derivatives are proportional to $(k-1)^2$.

5.8.3. D-R case: Discrete map (DST-II/III)

Let $u(0) = 0$, and $u'(R) = 0$. The discrete representation of u is:

Forward transform: DST-III

$$\tilde{u}_k(t) = \sqrt{\frac{2}{N-1}} \left((-1)^{(n-1)} u_N/2 + \sum_{n=2}^{N-1} u_n \sin\left[\frac{\pi}{N-1}\left(k-\frac{1}{2}\right)(n-1)\right]\right). \tag{5.39}$$

Inverse transform: DST-II

$$u_n = \sqrt{\frac{2}{N-1}} \left(\sum_{k=1}^{N-1} \tilde{u}_k(t) \sin\left[\frac{\pi}{N-1}\left(k-\frac{1}{2}\right) n\right]\right). \tag{5.40}$$

The forward transform does **not** require the value at $n = 1$, which is zero in this case. This is implicit in the sine expansion, since $\sin(n\pi) = 0$. This transform implies a boundary condition that is odd around $n = 1$, and even around $n = N$. Second derivatives are proportional to $(k - 1/2)^2$.

5.8.4. R-D case Discrete map (DCT-II/III)

Take $u'(0) = u(R) = 0$. The discrete representation of u is:

Forward transform: DCT-III

$$\tilde{u}_k = \sqrt{\frac{2}{N-1}} \left(u_1/2 + \sum_{n=2}^{N-1} u_n \cos\left[\frac{\pi}{N-1}\left(k-\frac{1}{2}\right)(n-1)\right]\right). \tag{5.41}$$

Inverse transform: DCT-II

$$u_n = \sqrt{\frac{2}{N-1}} \sum_{k=1}^{N} \tilde{u}_k \cos\left[\frac{\pi}{N-1}\left(k-\frac{1}{2}\right)(n-1)\right]. \tag{5.42}$$

The forward transform does not require the value at $n = N$, which is zero in this case. This transform implies a boundary condition that is even around $n = 1$, and odd around $n = N$. Second derivatives are proportional to $(k - 1/2)^2$.

5.9. Frequency or momentum grid

The frequency or momentum grid spacing is defined for all output graphs and periodic Fourier transforms as

$$\Delta k = \frac{2\pi}{N\Delta x}. \tag{5.43}$$

The internal momentum grid spacing used can differ from this, depending on the transforms used in the interaction picture. As explained above in Section (5.8), the internal momenta for trigonometric transforms are:

$$\Delta k = \frac{\pi}{(N-1)\,\Delta x}. \tag{5.44}$$

This is because the xSPDE algorithms allow the use of a sequence of interaction pictures. Each successive interaction picture is referenced to $t = t_n$, for the n-th step starting at $t = t_n$, so $\boldsymbol{a}_I(t_n) = \boldsymbol{a}(t_n) \equiv \boldsymbol{a}_n$. It is also possible to solve stochastic partial differential equations in xSPDE using explicit derivatives, but this is less efficient.

A discrete Fourier transform (DFT) using a fast Fourier transform method is employed for the interaction picture (IP) transforms used with periodic boundaries. This is normalized differently to the graphed Fourier transforms, but the difference is not computationally significant. However, the Δk used internally changes with the precise type of trigonometric transform used in other cases.

In one dimension, the DFT is usually defined by a sum over indices starting with zero, rather than the Matlab convention of one. Hence, if $\tilde{m} = m - 1$:

$$A_{\tilde{n}} = \mathcal{F}(a) = \sum_{\tilde{m}=0}^{N-1} a_{\tilde{m}} \exp\left[-2\pi i \tilde{m}\tilde{n}/N\right]. \tag{5.45}$$

For periodic boundaries, the IP Fourier transform can be written in terms of an FFT as

$$\boldsymbol{A}(\boldsymbol{k_n}) = \prod_j \left[\sum_{\tilde{m}_j} \exp\left[-i\left(dk_j dx_j\right)\tilde{m}_j\tilde{n}_j\right]\right]. \tag{5.46}$$

The inverse FFT Fourier transforms divide by the correct factors of $\prod_j N_j$ to ensure invertibility. Due to the periodicity of the exponential function, negative momenta are obtained if we consider an ordered lattice such that:

$$\begin{aligned} k_j &= (j-1)\Delta k \;\; (j \leq N/2) \\ k_j &= (j-1-N)\Delta k \;\; (j > N/2) \end{aligned}. \tag{5.47}$$

This Fourier transform is then multiplied by the appropriate factor to propagate in the interaction picture, then an inverse Fourier transform is applied. While it is not scaled for interaction picture transforms, an additional scaling factor is applied to obtain transformed fields in any averages for output plots.

In other words, in the averages

$$\tilde{a}_n = \frac{\Delta x}{\sqrt{2\pi}} A_{\tilde{n}'}. \tag{5.48}$$

where the indexing change indicates that graphed momenta are stored from negative to positive values. For plotted frequency spectra a **positive** sign is used in the frequency exponent of the transform to frequency space, to agree with common physics conventions.

5.10. Derivatives

5.10.1. Spectral derivatives

For spectral derivatives in the interaction picture, we define $D_x(k)$ to obtain a derivative. To explain, one integrates by parts:

$$D_x^p \tilde{\boldsymbol{a}}(\boldsymbol{k}) = [ik_x]^p \tilde{\boldsymbol{a}}(\boldsymbol{k}) = \frac{1}{(2\pi)^{d/2}} \int d\boldsymbol{x} e^{-i\boldsymbol{k}\cdot\boldsymbol{x}} \left[\frac{\partial}{\partial x}\right]^p \boldsymbol{a}(\boldsymbol{x}). \tag{5.49}$$

This means, for example, that to calculate a one dimensional space derivative in a Fourier interaction picture routine, one uses:

$$\nabla_x \to D_x. \tag{5.50}$$

Here Dx is an array of momenta in cyclic order in dimension d as defined above, suitable for an FFT calculation. The imaginary i is not needed to give the correct sign, as it is included in the derivative array. In two dimensions, a full two-dimensional Laplacian is:

$$\boldsymbol{\nabla}^2 = \nabla_x^2 + \nabla_y^2 \to D_x^2 + D_y^2. \tag{5.51}$$

Then, on inverting the transform

$$\left[\frac{\partial}{\partial x}\right]^p \boldsymbol{a}(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}} \int d\boldsymbol{x} e^{i\boldsymbol{k}\cdot\boldsymbol{x}} [D_x(\boldsymbol{k})]^p \tilde{\boldsymbol{a}}(\boldsymbol{k}). \tag{5.52}$$

5.10.2. Finite difference derivatives

For calculating derivatives using finite differences, the following central differencing method is used, away from the boundaries:

$$\nabla_x a(x_i) \to \frac{1}{2\Delta x}[a(x_{i+1}) - a(x_{i-1})]$$

$$\nabla_x^2 a(x_i) \to \frac{1}{\Delta x^2}[a(x_{i+1}) - 2a(x_i) + a(x_{i-1})]. \tag{5.53}$$

This raises the question of how to calculate derivatives at the boundary, for example at the lower boundary x_1, where $a(x_0)$ is not known, and similarly at the upper boundary. The answer depends on the boundary type [62], and is obtained by extending the boundary to additional points $a\left(x_0\right)$ and $a\left(x_{N+1}\right)$ that are assumed to extend the boundary condition:

Periodic: $a\left(x_0\right) = a\left(x_N\right)$

$$\nabla_x a\left(x_1\right) \to \frac{1}{2\Delta x}\left[a\left(x_2\right) - a\left(x_N\right)\right]$$

$$\nabla_x^2 a\left(x_1\right) \to \frac{1}{\Delta x^2}\left[a\left(x_2\right) - 2a\left(x_2\right) + a\left(x_N\right)\right]. \tag{5.54}$$

Dirichlet: $\tilde{a}\left(x_1\right)$ **specified:** $a\left(x_0\right) = \tilde{a}\left(x_1\right)$

$$\nabla_x a\left(x_1\right) \to \frac{1}{2\Delta x}\left[a\left(x_2\right) - \tilde{a}\left(x_1\right)\right]$$

$$\nabla_x^2 a\left(x_1\right) \to \frac{1}{\Delta x^2}\left[a\left(x_2\right) - \tilde{a}\left(x_1\right)\right]. \tag{5.55}$$

Robin/Neumann: $\tilde{a}'\left(x_1\right)$ **specified:** $a\left(x_0\right) = a\left(x_2\right) - 2\tilde{a}'\left(x_1\right)\Delta x$

$$\nabla_x a\left(x_1\right) \to \tilde{a}'\left(x_1\right)$$

$$\nabla_x^2 a\left(x_1\right) \to \frac{2}{\Delta x^2}\left[a\left(x_2\right) - a\left(x_1\right) - \tilde{a}'\left(x_1\right)\Delta x\right]. \tag{5.56}$$

In all cases the boundary value is evaluated as part of the derivative evaluation, so it can be a nonlinear function of $\mathbf{a}$.

6. SPDE examples

6.1. Gaussian diffraction

Free diffraction and absorption of a Gaussian wave-function in $d-1 = s$ space dimensions, is given by the partial differential equation (PDE):

$$\frac{da}{dt} = -\frac{\gamma}{2}a + \frac{i}{2}D\nabla^2 a. \tag{6.1}$$

The corresponding stochastic partial differential equation (SPDE) includes additional noise, so that:

$$\frac{da}{dt} = -\frac{\gamma}{2}a + \frac{i}{2}D\nabla^2 a + bw(t,x). \tag{6.2}$$

The xSPDE spectral definition in space is:

$$\tilde{a}(t,\mathbf{k}) = \frac{1}{[2\pi]^{s/2}}\int e^{i\mathbf{k}\cdot\mathbf{x}}a(t,\mathbf{x})d\mathbf{x}\,. \tag{6.3}$$

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

$$\begin{aligned} |a(t,\mathbf{x})|^2 &= \frac{1}{\left(1+(Dt)^2\right)^{s/2}} exp\left(-|\mathbf{x}|^2/\left(1+(Dt)^2\right)-\gamma t\right) \\ |\tilde{a}(t,\mathbf{k})|^2 &= exp\left(-|\mathbf{k}|^2-\gamma t\right). \end{aligned} \tag{6.4}$$

Exercises

- Simulate Gaussian diffraction in three dimensions using an xSPDE function
- Check your results against the exact solution
- The example below stores data in a standard Matlab file.

```
function [e] = Gaussian()
p.dimensions = 4;
p.name = 'Gaussian diffraction';
p.initial = @(v,p) exp(-0.5*(p.x.^2+p.y.^2+p.z.^2));
p.linear = @(p) 1i*0.05*(p.Dx.^2+p.Dy.^2+p.Dz.^2);
p.observe = @(a,p) a.*conj(a);
p.olabels = '|a(t,x)|^2';
p.file = 'Gaussian.mat';
p.images = 4;
e = xsim(p);
xgraph(p.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 100 samples.**
- Work out the exact solution and repeat the comparisons.

Note that for this, you'll need to add: $p.deriv = @(a, w, p)\ .. + 0.01 * (w(1, :) + i * w(2, :))$

6.2. Stochastic Ginzburg-Landau equation

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

$$\dot{a} = \left(1 - |a|^2\right) a + bw(t) + c\nabla^2 a \tag{6.5}$$

where

$$\left\langle w(x)w^*(x')\right\rangle = 2\delta\left(t - t'\right)\delta\left(x - x'\right). \tag{6.6}$$

The following new ideas are introduced for this problem:

1. `dimensions` **is the space-time dimension.**
2. **The** 'dot' **notation used for parallel operations over lattices**.
3. `linear` **is the linear operator - a Laplacian in these cases.**
4. `images` **produces movie-style images at discrete time slices.**
5. `Dx` **indicates a derivative operation, $\partial/\partial x$.**
6. $-5 < x < 5$ **is the default xSPDE coordinate range in space.**

Exercises

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

In the first case, you should get the output graphed in Fig (6.1) .

```
clear;
p.name = 'Extended laser gain equation';
p.noises = 2;
p.dimensions = 3;
p.steps = 10;
p.linear = @(p) 1i*0.01*(p.Dx.^2+p.Dy.^2);
p.observe = @(a,~) abs(a).^2;
p.images = 6;
p.olabels = '|a|^2';
p.deriv = @(a,w,~) (1-abs(a(1,:).^2)).*a(1,:)+...
          0.001*(w(1,:)+1i*w(2,:));
xspde(p)
```

Here the notation $a(1,:)$ means that the operation is repeated over all values of the subsequent indices, which are the two spatial lattice indices in this case.

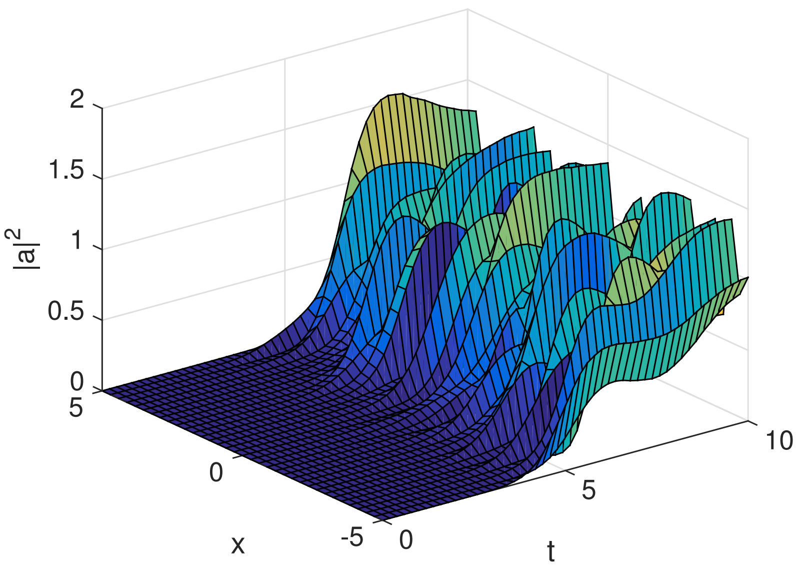


Figure 6.1.: *Simulation of the stochastic equation describing symmetry breaking in two dimensions. Spatial fluctuations are caused by the different phase-domains that interfere. The graph obtained here is projected onto the $y = 0$ plane.*

6.3. NLS soliton

The famous nonlinear Schrödinger equation (NLSE) is:

$$\frac{da}{dt} = \frac{i}{2}\left[\nabla^2 a - a\right] + ia\left|a\right|^2 . \tag{6.7}$$

Together with the initial condition that $a(0,x) = sech(x)$, this has a soliton [63], an exact solution that doesn't change in time:

$$a(t,x) \quad = \quad sech(x). \tag{6.8}$$

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

$$\tilde{a}(t,0) \quad = \quad \frac{1}{\sqrt{2\pi}}\int sech(x)dx = \sqrt{\frac{\pi}{2}}. \tag{6.9}$$

Exercises

- **Solve the NLSE for a soliton using a function instead of a script, then include an additive complex noise of $0.01(w_1 + iw_2)$ to the differential equation, and plot again with an average over 1000 samples.**

6.4. Planar noise

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

$$\frac{d\mathbf{a}}{dt} = \frac{i}{2}\nabla^2\mathbf{a} + \mathbf{w}(t,x). \tag{6.10}$$

where ζ is a delta-correlated complex noise vector field:

$$w_j(t,\mathbf{x}) = \left[w_j^{re}(t,\mathbf{x}) + i\zeta_j^{im}(t,\mathbf{x})\right]/\sqrt{2}, \tag{6.11}$$

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

$$a(0,\mathbf{x}) = \zeta^{(in)}(\mathbf{x}) \tag{6.12}$$

where:

$$\zeta^{(in)}(\mathbf{x}) = \left[\zeta^{re(in)}(\mathbf{x}) + i\zeta^{im(in)}(\mathbf{x})\right]/\sqrt{2} \tag{6.13}$$

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

$$\begin{aligned}\left\langle \left|a_j\left(t,\mathbf{x}\right)\right|^2\right\rangle &= (1+t)/dV\\ \left\langle \left|\tilde{a}_j\left(t,\mathbf{k}\right)\right|^2\right\rangle &= (1+t)/dV_k\\ \left\langle \tilde{a}_1\left(t,\mathbf{k}\right)\tilde{a}_2^*\left(t,\mathbf{k}\right)\right\rangle &= 0.\end{aligned} \tag{6.14}$$

Here, the noise is delta-correlated, and dV, dV_k are the cartesian space and momentum space lattice cell volumes, respectively. Suppose that $N_s = N_xN_y$ is the total number of spatial points, and there are $N_{x(y)}$ points in the x(y)-direction, so then:

$$\begin{aligned}dV &= dxdy\\ dV_k &= dk_xdk_y = \frac{(2\pi)^2}{V}.\end{aligned} \tag{6.15}$$

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 parameter *p.noises* indicates how many noise fields are generated, while *p.knoises* indicates how many of these are spatially correlated, via Fourier transform, filter and inverse Fourier transform. These appear as additional noise cells. The filtered noises have a finite correlation length in general, but in this example are delta-correlated.

Exercises

- **Solve the planar noise growth equation**

```
function [e] = PlanarExample()
p.name = 'Planar noise growth';
p.dimensions = 3;
p.fields = 2;
p.ranges = [1,5,5];
p.steps = 2;
p.noises = 2;
p.knoises = 2;
p.inrandoms = 2;
p.krandoms = 2;
p.ensembles = [10,4,4];
p.initial = @Initial;
p.deriv = @Da;
p.linear = @(p) 0.5*1i*(p.Dx.^2+p.Dy.^2);
p.observe = @(a,p) a(1,:).*conj(a(1,:));
p.olabels = '<|a_1(x)|^2>';
p.compare = @(p) (1+p.t)/p.dv;
p.images = 4;
e = xspde(p);
end
function a0 = Initial(u,v,~)
a0(1,:,:)  = (u(1,:,:)+1i*u(2,:,:))/sqrt(2);
a0(2,:,:)  = (v(1,:,:)+1i*v(2,:,:))/sqrt(2);
end
function da = Da(~,w,z,~)
da(1,:)  = (w(1,:)+1i*w(2,:))/sqrt(2);
da(2,:)  = (z(1,:)+1i*z(2,:))/sqrt(2);
end
```

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

6.5. Gross-Pitaevskii equation

The next example is a stochastic Gross-Pitaevskii (GP) equation [64] in two dimensions,

$$\frac{da}{dt} = \frac{i}{2}\nabla^2 a - ia(V(r) - i\kappa(r) + |a|^2) + \epsilon\eta \tag{6.16}$$

where η is a correlated complex noise vector field:

$$\eta(t, \mathbf{x}) = w_1(t, \mathbf{x}) + iw_2(t, \mathbf{x}), \tag{6.17}$$

with the initial condition that the initial random field and the noise are both filtered in momentum space

$$a(0,\mathbf{x}) = a_0(\mathbf{x}) + \epsilon\zeta^{(in)}(\mathbf{x}) \tag{6.18}$$

where:

$$\zeta^{(in)}(\mathbf{x}) = v_1(\mathbf{x}) + iv_2(\mathbf{x}) \tag{6.19}$$

We add a Gaussian filter in momentum space for both the initial random field and noise so that, if $\tilde{w}(\mathbf{k})$ is a delta-correlated noise in momentum space:

$$\begin{aligned} w(\mathbf{k}) &= \tilde{w}(\mathbf{k}) \exp\left(-|\mathbf{k}|^2\right) \\ v(\mathbf{k}) &= \tilde{v}(\mathbf{k}) \exp\left(-|\mathbf{k}|^2\right) \end{aligned} \tag{6.20}$$

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, $p.noises(1)$, indicates how many noise fields are generated that are delta-correlated in x, while $p.noises(2)$ 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 $p.noises(1)+p.noises(2)$. The filtered noises have a finite correlation length.

Exercises

- **Solve the stochastic GP equation** (6.16)**, with a noise coefficient of** $b = 0.1$**,** $V = 0.01\,|\mathbf{x}|^2$, $\kappa = 0.001\,|\mathbf{x}|^4$**, and a stored output data file.**

```
function [e] = GPE()
p.name = 'GPE';
p.dimensions = 3;
p.points = [101,64,64];
p.ranges = [1,20,20];
p.noises = 0;
p.knoises = 2;
p.inrandoms = 0;
p.krandoms = 2;
p.rfilter = @(w,p) w.*exp(-p.kx.^2-p.ky.^2);
p.nfilter = @(v,p) v.*exp(-p.kx.^2-p.ky.^2);
b = @(xi) .1*(xi(1,:,:)+1i*xi(2,:,:));
p.initial = @(u,v,p) (p.x+1i*p.y)./(1+10*(p.x.^2
+p.y.^2))+b(v);
V = @(p) 0.01*(p.x.^2 + p.y.^2)-0.001*1i*(p.x.^2
+p.y.^2).^2;
p.deriv = @(a,v,w,p) -1i*a.*(V(p)+conj(a).*a)+b(w);
p.linear = @(p) 0.5*1i*(p.Dx.^2+p.Dy.^2);
p.observe{1} = @(a,p) a.*conj(a);
p.images = {2};
p.imagetype = {2};
p.olabels = {'|a|^2'};
p.file = 'GPE.mat';
e = xsim(p);
xgraph(p.file,p);
end
```

6.6. Characteristic equation

The next example is the characteristic equation for a traveling wave at constant velocity [65]. 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 - called apodization - at the boundaries. The equation is:

$$\frac{da}{dt} + \frac{da}{dx} = 0. \tag{6.21}$$

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

$$a(t,x) = sech(2(x-t)+5). \tag{6.22}$$

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

$$a(t,0) = sech(2(t-5/2)). \tag{6.23}$$

Exercises

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

```
function [e] = Characteristic()
p.name = 'Characteristic';
p.dimensions = 2;
p.initial = @(v,p) sech(2.*(p.x+2.5));
p.deriv = @(a,z,p) 0*a;
p.linear = @(p) -p.Dx;
p.olabels = {'a_1(x)'};
p.compare = @(p) sech(2.*(p.t-2.5));
e = xspde(p);
end
```

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

6.7. Nonlinear Anderson localization

A random potential prevents normal wave-packet spreading in quantum-mechanics. This is Anderson localization [66]: 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 [67]. 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

$n = |\psi|^2$, but there is a complication, which is that there are nonlinearities from atomic scattering [68].

This can be treated either using a Schrödinger 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:

$$\frac{\partial \psi}{\partial t} = \frac{1}{i\hbar}\left[-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r}) + g|\psi|^2\right]\psi. \tag{6.24}$$

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 $V(\mathbf{r})$ 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:

$$\frac{\partial a_1}{\partial \tau} = i\left[\frac{\partial}{\partial \zeta^2}^{\,2} - |a_2|^2 - |a_1|^2\right]a_1. \tag{6.25}$$

A convenient initial condition is to use:

$$\begin{aligned} a_1 &= a_0 \exp(-\zeta^2) \\ \left\langle a_2(\zeta)a_2(\zeta')\right\rangle &= v\delta\left(\zeta-\zeta'\right). \end{aligned} \tag{6.26}$$

Exercise

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

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

6.8. Nonlinear Schrödinger equation with Neumann boundary conditions

This solves a (1+1)-dimensional PSDE with an initial condition of $a(t=0,x) = sech(x)$ and

$$\frac{\partial a}{\partial t} = i\cdot\left(a\cdot\left(|a|^2 - \frac{1}{2}\right) + \frac{1}{2}\frac{\partial^2 a}{\partial x^2}\right). \tag{6.27}$$

The solution is subject to Neumann boundary conditions with boundary values at zero

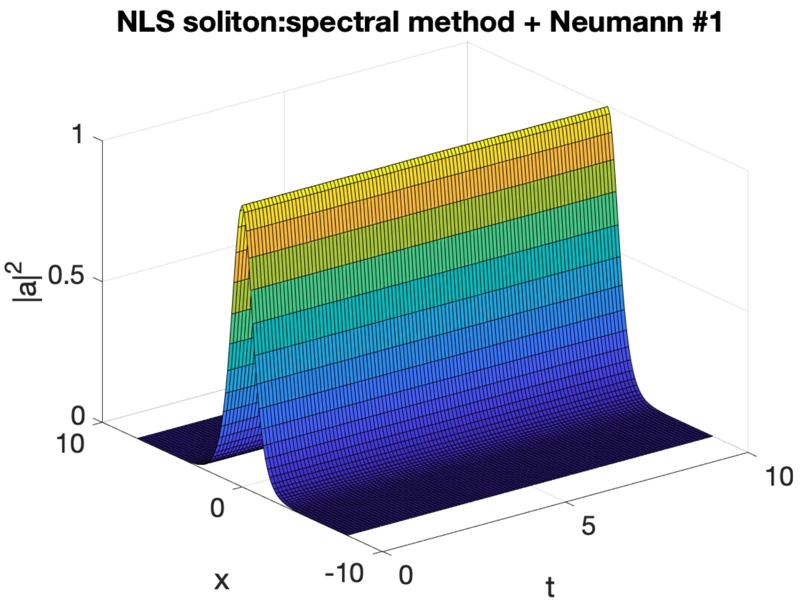
$$\frac{\partial a}{\partial x}(t, \pm x_m) \quad = \quad 0\,. \tag{6.28}$$

The equation is a deterministic nonlinear Schrödinger equation, which applies to nonlinear optics, Bose-Einstein condensates and plasma physics. The observables are $o_1 \equiv |a|^2$ and $o_2 \equiv \int_{-x_m}^{x_m} \left|\frac{\partial}{\partial x} a\right|^2 dx$.

Notes

- The boundary conditions are specified with p.boundaries{2}, which is the x-dimension.
- The integration differential dx does not have to be entered, as this is the default.
- Three transverse graphs were specified, but they aren't reproduced here.
- As there is only one field, which is the default, this does not need to be given.
- Since there is no noise, the default integration method was RK4.

```
function [e] = SolitonDerivN()
p.dimensions = 2;
p.points = [101,101];
p.ranges = [10,15];
p.initial = @(v,p) sech(p.x);
p.observe{1} = @(a,p) a.*conj(a);
p.observe{2} = @(a,p) Int(abs(D1(a,2,p)).^2,p);
p.olabels = {'|a|^2','\int |da/dx|^2 dx'};
p.name = 'NLS soliton:spectral method + Neumann';
p.boundaries{2} = [-1,-1];
p.transverse = {3};
p.deriv = @(a,~,p) 1i*a.*(conj(a).*a);
p.linear = @(p) 0.5*1i*(p.Dx.^2-1);
e = xspde(p);
end
```



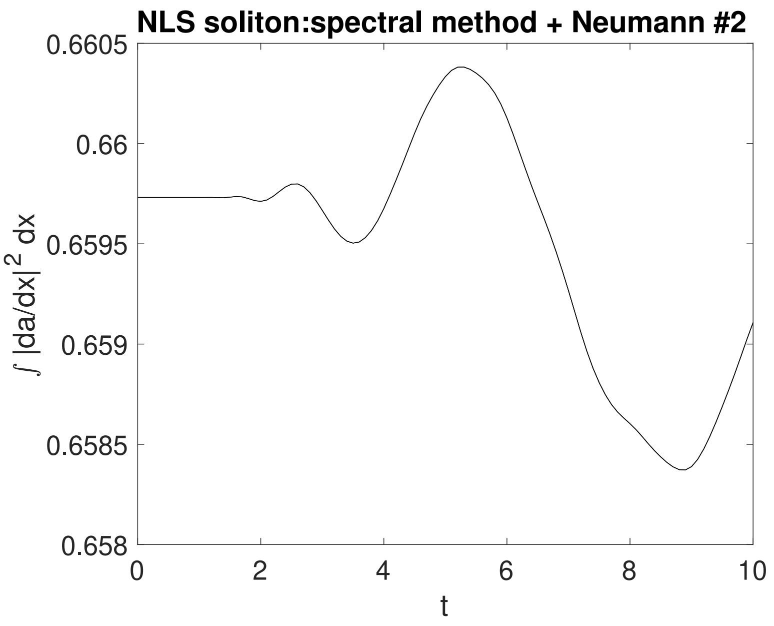


Figure 6.2.: *Top figure: Evolution of the field modulus squared of an NLS soliton with Neumann boundaries.*
Bottom figure: Evolution of the integrated modulus squared of the gradient for an NLS soliton with Neumann boundaries, showing how the reflected fields at the boundaries change the result even though this is not readily visible above.

6.9. Planar noise growth

This solves a (1+2)-dimensional PSDE describing the growth of noise in a planar vector field with a diffraction term giving rise to noise dispersion. The equation is:

$$\frac{\partial \mathbf{a}}{\partial t} = \frac{i}{2}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial x^2}\right)\mathbf{a}+\eta\left(t,x\right) . \tag{6.29}$$

The initial conditions are that $\mathbf{a}=\left(\mathbf{v}_x+i\mathbf{v}_y\right)/\sqrt{2}$, where:

$$\left\langle v_i\left(\mathbf{x}\right)v_j\left(\mathbf{x}'\right)\right\rangle=\delta\left(\mathbf{x}-\mathbf{x}'\right)\delta_{ij} \tag{6.30}$$

the noise correlations are that $\eta=\left(\mathbf{w}_x+i\mathbf{w}_y\right)/\sqrt{2}$, where:

$$\left\langle w_i\left(\mathbf{r}\right)w_j\left(\mathbf{r}'\right)\right\rangle = \delta\left(t-t'\right)\delta_{ij}\left(\mathbf{x}-\mathbf{x}'\right) \tag{6.31}$$

The solution has periodic boundary conditions. The noise correlations for the second field are specified in momentum space. As there are no filters, the noise terms are delta-correlated in both momentum ($\mathbf{k}$) and in space (x). The exact results for comparison within each field are similar in position and momentum space:

$$\begin{aligned}\left\langle\left|a_i\left(t,\mathbf{x}\right)\right|^2\right\rangle&=\left(1+t\right)/\Delta A_x.\\ \left\langle\left|a_i\left(t,\mathbf{k}\right)\right|^2\right\rangle&=\left(1+t\right)/\Delta A_k.\end{aligned} \tag{6.32}$$

Here, $\Delta A_{x,k}$ is the area of a lattice cell in space or momentum space. This is $\Delta A_x = 1/49$ for the parameters used. The correlations are proportional to N_s, the number of points in the spatial lattice, which is $35^2 = 1225$ for the spatial lattice used:

$$\int\left\langle\left|a_i\left(t,\mathbf{x}\right)\right|^2\right\rangle d\mathbf{x}=\int\left\langle\left|a_i\left(t,\mathbf{k}\right)\right|^2\right\rangle d\mathbf{k}=N_s\left(1+t\right). \tag{6.33}$$

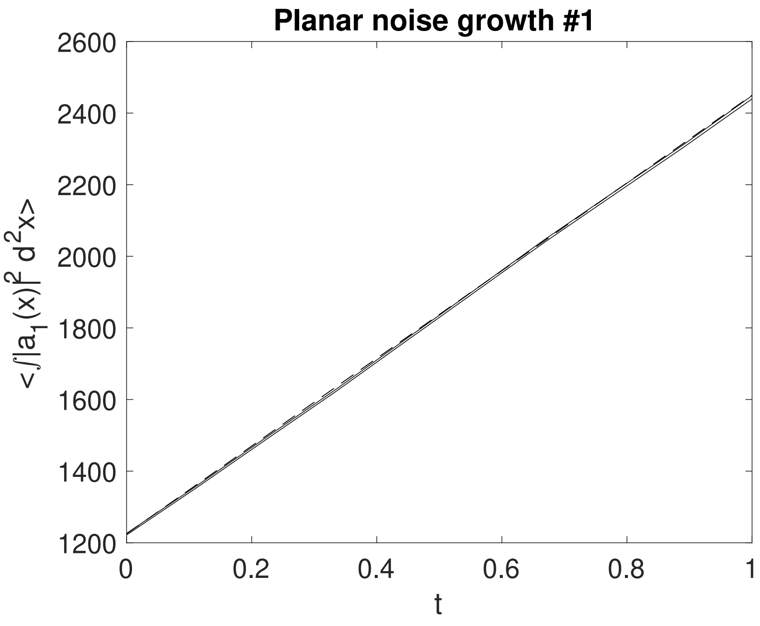
Notes

- All three types of ensemble are used
- The much lower sampling error after integration is evident in the graphs
- Spatially resolved graphs show larger sampling errors
- The integration method is mid-point, as it is stochastic.
- Two k-space noises are specified, but they aren't filtered.
- Under these conditions, x-space and k-space noise are identical.

```
function [e] = Planar()
p.name = 'Planar noise growth';
p.dimensions = 3;
p.fields = 2;
p.ranges = [1,5,5];
p.points = 10;
p.noises = 2;
p.knoises = 2;
p.inrandoms = 2;
p.krandoms = 2;
p.ensembles = [10,2,12];
p.initial = @Initial;
p.deriv = @D_planar;
p.linear = @(p) 1i*0.5*(p.Dx.^2+p.Dy.^2);
p.observe{1} = @(a,p) Int(a(1,:).*conj(a(1,:)),p);
p.observe{2} = @(a,p) Int(a(2,:).*conj(a(2,:)),p.dk,p);
p.observe{3} = @(a,p) real(Ave(a(1,:).*conj(a(2,:)),p));
p.observe{4} = @(a,p) a(2,:).*conj(a(2,:));
p.transforms = {[0,0,0],[0,1,1],[0,1,1]};
p.olabels{1} = '<\int| a_1(x)|^2 d^2x> ';
p.olabels{2} = '<\int| a_2(k)|^2 d^2k> ';
p.olabels{3} = '<< a_1(k) a^*_2(k)>>';
p.olabels{4} = '< |a_2(x)|^2>';
p.compare{1} = @(p) (1+p.t)*p.nspace;
p.compare{2} = @(p) (1+p.t)*p.nspace;
p.compare{3} = @(p) 0.0;
e = xspde(p);
end

function a0 = Initial(u,v,~)
a0(1,:)  = (u(1,:)+1i*u(2,:))/sqrt(2);
a0(2,:)  = (v(1,:)+1i*v(2,:))/sqrt(2);
end

function da = D_planar(~,u,w,~) %%Derivatives
da(1,:)  = (u(1,:)+1i*u(2,:))/sqrt(2);
da(2,:)  = (w(1,:)+1i*w(2,:))/sqrt(2);
end
```



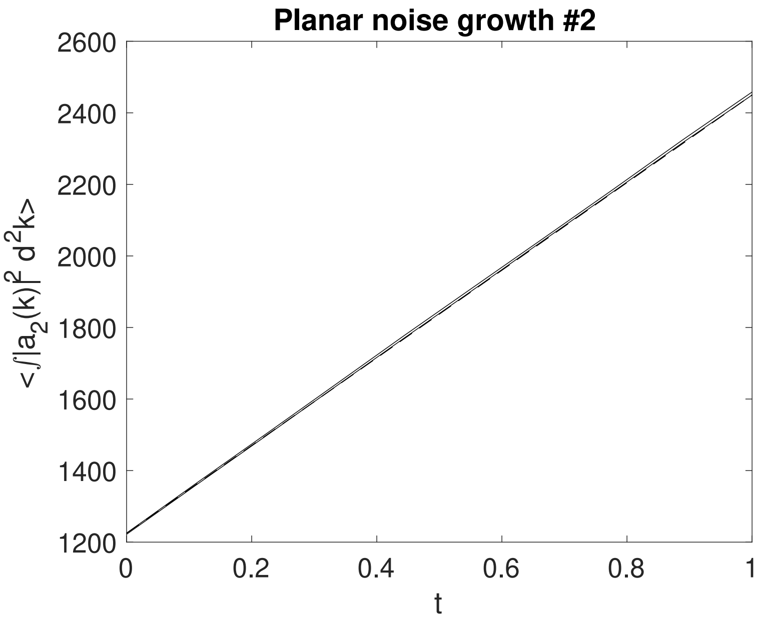
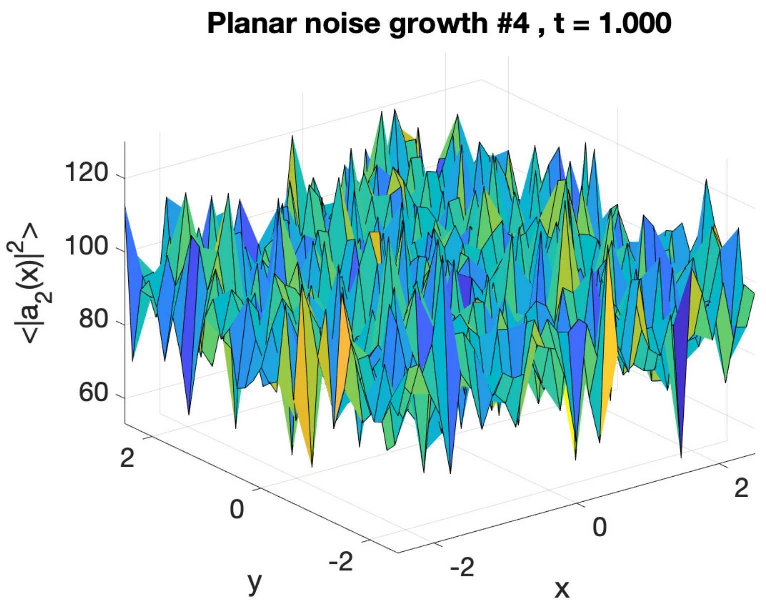


Figure 6.3.: *Top and bottom figure: Time evolution of the integrated modulus square of the first and second field, respectively. The solid lines indicate upper and lower bounds of the stochastic error, which the dashed lines indicate theoretical predictions.*



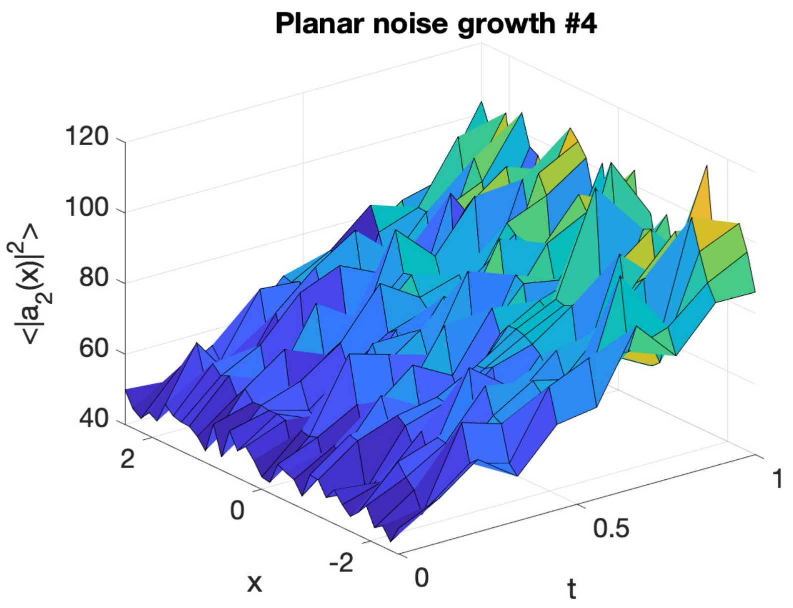


Figure 6.4.: *Top figure: 3D plot of the modulus square of a_2 at $t = 1$ as a function of x and y. Bottom figure: 3D plot of the modulus square of a_2 for $y = 0$ as a function of x and t.*

6.10. Gross-Pitaevskii equation with vortex formation

This solves a (1+2)-dimensional PDE called the Gross-Pitaevskii equation. In addition to the standard GPE terms, it includes the vortex forming term $(\mathbf{x}\times\nabla)\,a$. There is just one ensemble member, to demonstrate how a single trajectory can be imaged. The equation is:

$$\begin{aligned}
\frac{\partial a}{\partial t} &= \left(\frac{1}{2}\nabla^2 a - \left\|\left(\left(V(\mathbf{x}) + 200\,|a|^2\right) + 0.6i\cdot(\mathbf{x}\times\nabla)\right)a\right\|\right) \\
V(\mathbf{x}) &= 0.35\left(x^2+y^2\right) \\
\|b(\mathbf{x})\| &= \frac{b(\mathbf{x})}{\int |b|^2\,d\mathbf{x}}\,. \qquad (6.34)
\end{aligned}$$

Here,$\|\cdot\|$ is the normalized derivative and $\times$ indicates the two-dimensional cross-product. The system is initialized as

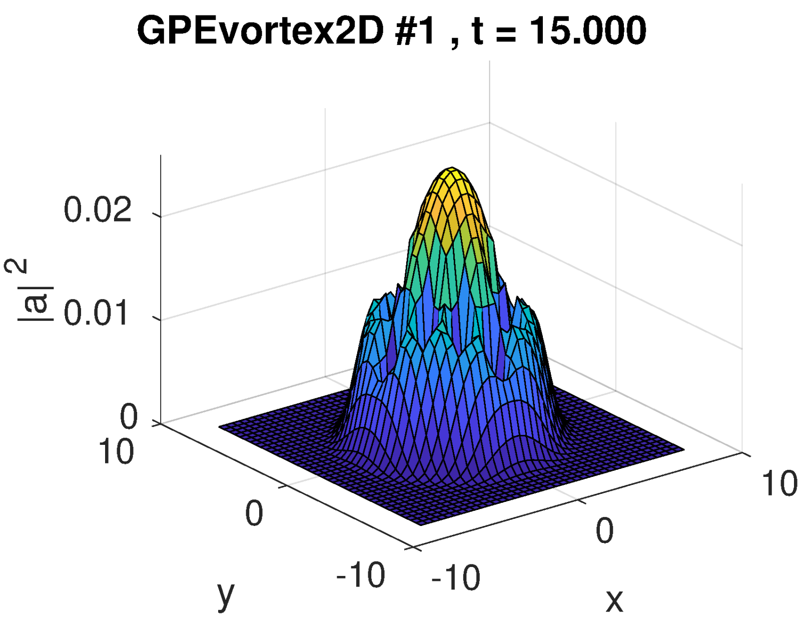
$$a(t=0,\mathbf{x}) = 0.1\cdot\exp\left(-V(\mathbf{x})\right)\,. \qquad (6.35)$$

Notes

- This is a deterministic partial differential equation case
- The 15 intermediate steps used are necessary to reduce integration errors
- The trap potential is an inline function, and is not a parameter
- Normalization is used because otherwise particle number is not conserved
- The output includes transverse images to show how the vortices develop
- Different imagetypes are used to show different 3D features

```
function [e] = GPEvortex2D()
p.name = 'GPEvortex2D';
p.dimensions = 3;
p.fields = 1;
p.points = [50,40,40];
p.ranges = [15,16,16];
p.steps = 15;
g = 200;
om = 0.6;
L = @(a,p) 1i*(p.x.*D1(a,3,p)-p.y.*D1(a,2,p));
V = @(p) 0.35*(p.x.^2+p.y.^2);
p.initial = @(v,p) 0.1*exp(-V(p));
rho = @(a) g*conj(a).*a;
p.deriv = @normda;
p.da1 = @(a,w,p) -a.*(V(p)+rho(a))+om*L(a,p);
p.linear = @(p) 0.5*(p.Dx.^2+p.Dy.^2);
p.observe{1} = @(a,p) a(1,:).*conj(a(1,:));
p.observe{2} = @(a,p) a(1,:).*conj(a(1,:));
p.images = {2,2};
p.imagetype = {1,2};
p.olabels = {'|a|^2','|a|^2'};
e = xspde(p);

function b = normda(a,w,p)
% b = NORMDA(a,z,p) is a normalized derivative
% Takes a derivative and returns a normalized step
b = a+p.da1(a,w,p)*p.dtr;
norm = sqrt(Int(abs(b).^2,p.dx,p));
b = (b./norm-a)/p.dtr;
end
end
```



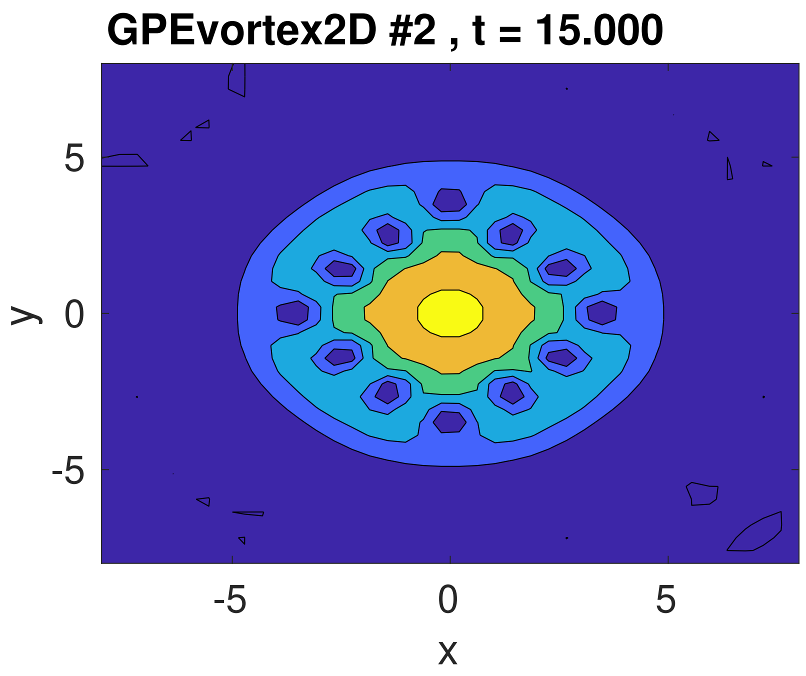


Figure 6.5.: *Top and bottom figure: The computed solution for $|a|^2$ at $t = 15$ as a function of x, y as a 3D plot (top) and as a color map (bottom).*

6.11. Heat equation with non-periodic boundaries

This example solves a (1+1)-dimensional PDE with an initial condition of $\mathbf{a}\left(t=0,x\right)=\mathbf{f}\left(x\right)$ and

$$\frac{\partial\mathbf{a}}{\partial t} = \frac{\partial^2\mathbf{a}}{\partial x^2}\,. \tag{6.36}$$

The solution is subject to either periodic boundary conditions or Dirichlet and/or Neumann with boundary values of zero at $x_{\pm}=[0,\pi]$ so that $a\left(t,x_{\pm}\right)=0$ or $\partial a/\partial x\left(t,x_{\pm}\right)=0$. Each component has different combinations of boundary types. Using spectral methods the solutions here are exact, up to round-off errors of order 10^{-15}, and are also much faster than with finite differences, which is demonstrated in the example.

In all cases the grid range is from $x=0$ to $x=\pi$, and the time duration is from $t=0$ to $t=4$. In the examples, the spectral propagation error is reduced by more than 10^{10} and the time is reduced by a factor of 20 compared to the finite-difference methods. The periodic method has boundaries just outside the grid.

Dirichlet-Dirichlet With $a(0)=a(\pi)=0$, the exact solution has the form:

$$a=\sum_{n=1}^{2} S_n \sin\left(nx\right) e^{-n^2 t}. \tag{6.37}$$

Suppose that

$$a(x,0)=1-2x/\pi+4\sin\left(x\right)+\sin\left(2x\right), \tag{6.38}$$

and

$$\begin{aligned} a_x(-\pi/2) &= 2-4e^{-t}-e^{-4t} \\ a_x(\pi/2) &= +4e^{-t}+e^{-4t} \end{aligned}$$

For this case:

$$a(x,t)=1-2x/\pi+4\sin\left(x\right)e^{-t}+\sin\left(2x\right)e^{-4t}. \tag{6.39}$$

Neumann-Neumann With $\partial_x a(0)=\partial_x a(\pi)=0$, the exact solution has the form:

$$a=\sum_{n=0}^{\infty} C_n \cos\left(nx\right) e^{-n^2 t}. \tag{6.40}$$

Suppose that

$$a(x,0)=5+4\cos\left(x\right)+\cos\left(2x\right), \tag{6.41}$$

For this case:

$$a(x,t)=5+4\cos\left(x\right)e^{-t}+\cos\left(2x\right)e^{-4t}. \tag{6.42}$$

Dirichlet-Neumann Here $a(0) = \partial_x a(\pi) = 0$, the exact solution has the form:

$$a = \sum_{n=1}^{\infty} S_n \sin\left((2n-1)x/2\right) e^{-(2n-1)^2 t/4}. \tag{6.43}$$

Suppose that

$$a(x,0) = 4\sin\left(x/2\right) + \sin\left(3x/2\right), \tag{6.44}$$

For this case:

$$u(x,0) = 4\sin\left(x/2\right) e^{-t/4} + \sin\left(3x/2\right) e^{-9t/4}. \tag{6.45}$$

Neumann-Dirichlet Here $\partial_x a(0) = a(\pi) = 0$, the general solution has the form:

$$a = \sum_{n=1}^{\infty} C_n \cos\left((2n-1)x/2\right) e^{-(2n-1)^2 t/4}. \tag{6.46}$$

Suppose that

$$a(x,0) = 4\cos\left(x/2\right) + \cos\left(3x/2\right). \tag{6.47}$$

For this case:

$$a(x,t) = 4\cos\left(x/2\right) e^{-t/4} + \cos\left(3x/2\right) e^{-9t/4}. \tag{6.48}$$

Periodic Here $a(0) = a(\epsilon\pi)$, where $\epsilon = N/\left(N-1\right)$ accounts for the periodic boundaries being outside the grid range, so the general solution has the form:

$$\begin{aligned} a = &\sum_{n=1}^{\infty} S_n \sin\left(2nx/\epsilon\right) e^{-4n^2 t/\epsilon^2} \\ &+ \sum_{n=0}^{\infty} C_n \cos\left(2nx/\epsilon\right) e^{-4n^2 t/\epsilon^2}. \end{aligned} \tag{6.49}$$

Suppose that

$$a(x,0) = 2 + \cos\left(2x/\epsilon\right) + \sin\left(4x/\epsilon\right). \tag{6.50}$$

For this case:

$$u(x,0) = 2 + 2\cos\left(2x/\epsilon\right) e^{-4t/\epsilon^2} + \sin\left(4x/\epsilon\right) e^{-16t/\epsilon^2}. \tag{6.51}$$

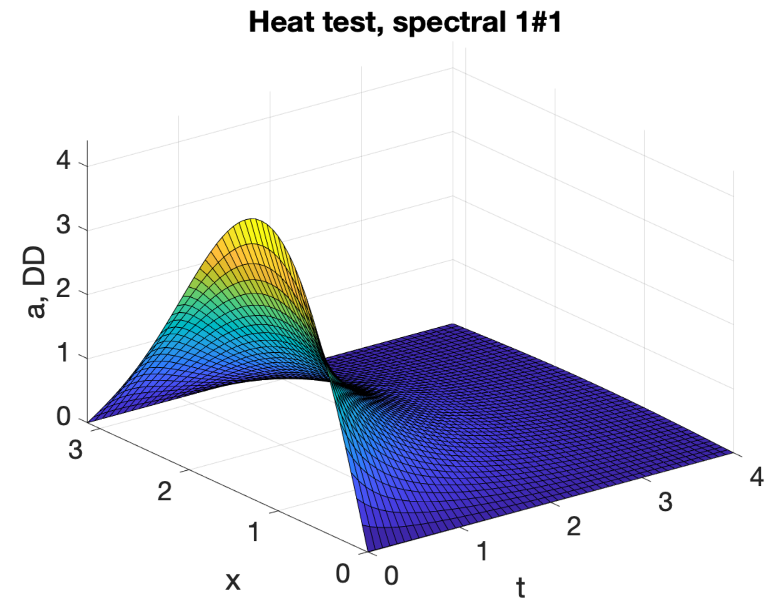
Notes

- This is a deterministic partial differential equation, although noise can be added
- Different boundary conditions apply to each component
- Sequential integration is used, but the initial condition is just recycled.
- In p1, the 80 intermediate steps are necessary to reduce finite-difference errors

```
function [e] = Boundaries()
p.dimensions = 2;
p.points = [51,51];
p.order = 0;
p.verbose = 1;
p.method = @MP;
p.fields = 5;
p.ranges = [4,pi];
p.origins = [0,0];
p.initial = @heat_in;
p.observe = {@(a,p) a(1,:),@(a,p) a(2,:),@(a,p) a(3,:)...
@(a,p) a(4,:),@(a,p) a(5,:)};
p.compare = {@heat_1,@heat_2,@heat_3,@heat_4,@heat_5};
p.diffplot = {1,1,1,1,1};
p.olabels = {'a, DD','a, NN','a, DN','a, ND','a, PP'};
p.name = 'Heat test, spectral';
p.boundaries{2}= [1,1;-1,-1;1,-1;-1,1;0,0];
p1 = p;
p.linear = @(p) p.Dx.^2;
p1.deriv = @(a,w,p) D2(a,2,p);
p1.steps = 40;
p1.transfer = @(~,~,p) heat_in(0,p);
p1.name = 'Heat test, finite diffs';
e = xspde({p,p1});
end

function a = heat_in(~,p)
a(1,:)  = 4*sin(p.x)+sin(2*p.x);
a(2,:)  = 5+4*cos(p.x)+cos(2*p.x);
a(3,:)  = 4*sin(p.x/2)+sin(3*p.x/2);
a(4,:)  = 4*cos(p.x/2)+cos(3*p.x/2);
a(5,:)  = 2+cos(2*p.x/1.02)+sin(4*p.x/1.02);
end

function o = heat_1(p)
o = 4*sin(p.x).*exp(-p.t)+sin(2*p.x).*exp(-4*p.t);
end
function o = heat_2(p)
o = 5+4*cos(p.x).*exp(-p.t)+cos(2*p.x).*exp(-4*p.t);
end
function o = heat_3(p)
o = 4*sin(p.x/2).*exp(-p.t/4)+sin(3*p.x/ 2).*exp(-9*p.t/4);
end
function o = heat_4(p)
o = 4*cos(p.x/2).*exp(-p.t/4)+cos(3*p.x/2).*exp(-9*p.t/4);
end
function o = heat_5(p)
o = 2+cos(2*p.x/1.02).*exp(-4*p.t/1.02^2)+...
sin(4*p.x/1.02).*exp(-16*p.t/1.02^2);
end
```



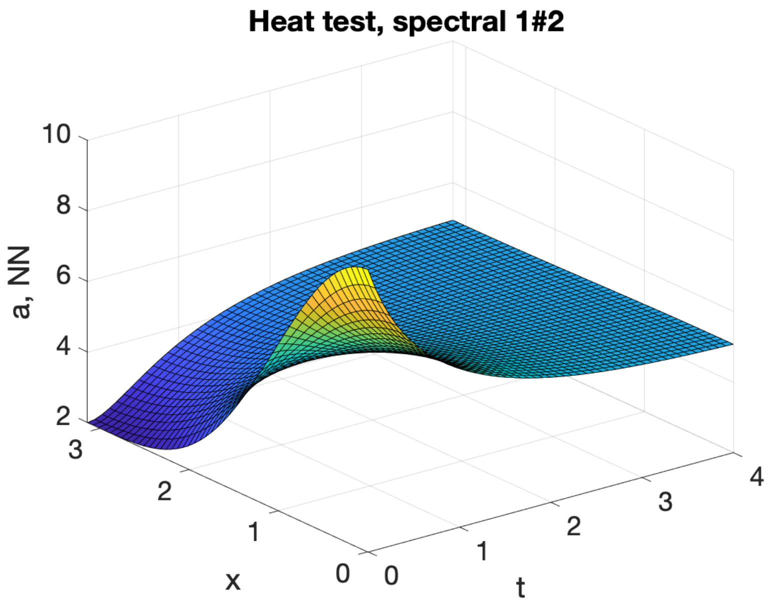


Figure 6.6.: *Top figure: Spectral solution for a as a function of time and position with Dirichlet-Dirichlet boundaries. Bottom figure: Plot of the solution with Neumann-Neumann boundaries.*

6.12. Peregrine solitary wave with arbitrary boundary conditions

Peregrine solitary waves are models for isolated large ocean waves. They are modeled as solutions to a (1+1)-dimensional PDE,

$$\frac{\partial a}{\partial t} = i\cdot\left(a\cdot|a|^2+\frac{1}{2}\frac{\partial^2 a}{\partial x^2}\right). \tag{6.52}$$

The Peregrine solution on an infinite domain is:

$$a = e^{it}\left(\frac{4\left(1+2it\right)}{1+4\left(t^2+x^2\right)}-1\right). \tag{6.53}$$

In the example, this is solved using finite boundary conditions, with initial values and boundary values that correspond to the exact solution.

Notes

- The boundary conditions are specified with p.boundaries{2}, which is the x-dimension.
- Four different boundary conditions are specified for a four component field.
- Spectral methods are used for accuracy
- No noise is included in the example below, but it can be added
- The boundary values are time-dependent, and are specified with p.boundfun
- For improved stability, the integration method is the semi-implicit **MP** method.

Errors

The reported errors in this example are reduced using second order extrapolation, by specifying p.order =2. This gives the following RMS average errors for the output intensity, averaged over all four cases:

- Step=0.000621
- Diff=0.00019

The 'Step' error is from subtracting the most accurate results from lower accuracy results. With extrapolation specified, the most accurate results are the extrapolated results. The less accurate ones are at half the specified time-step. This is averaged over all space-time points, and normalized by the maximum intensity of $|a|^2 = 9$.

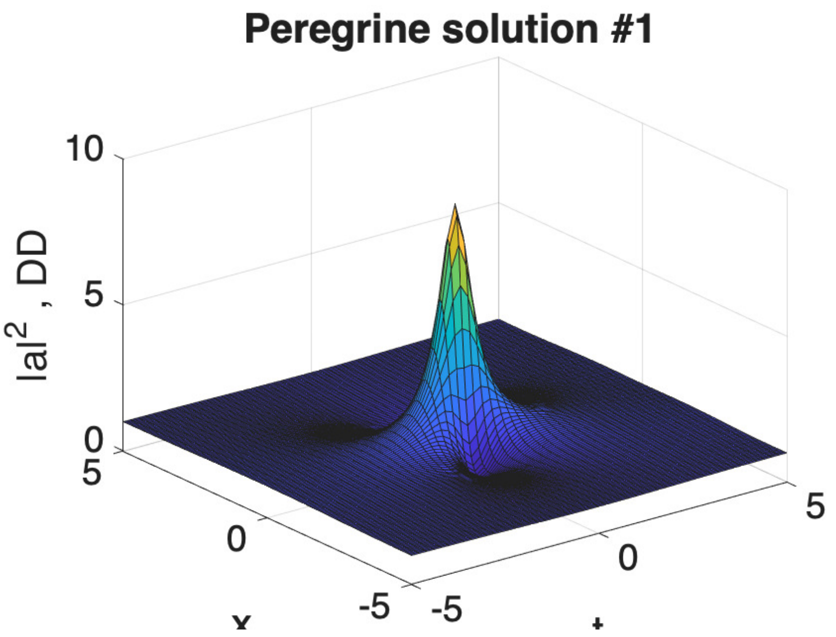
The 'Diff' error is from comparing the most accurate results with the analytic solution. This demonstrates a typical case where the time-step error is an upper bound to the true error. The maximum error occurs at large times, and is greater than the RMS error by about 10×in this case. All four boundary types used give similar results and errors.

```
function e = Peregrine()
% e = Peregrine() tests xSPDE for a nonlinear Schrodinger equn.
% Using NN,DD,DN,ND boundary values with a spectral method
% Uses time dependent boundary values for a peregrine solution
p.dimensions = 2;
p.noises = 1;
p.fields = 4;
p.order = 2;
p.ranges = [10,10];
p.origins = [-5,-5];
p.points = [51,161];
p.method = @MP;
p.olabels = {'|a|^2 , DD','|a|^2 , NN','|a|^2 , DN','|a|^2 , ND'};
p.boundaries{2} = [1,1;-1,-1;1,-1;-1,1];
p.boundfun = @boundval;
sol = @(p) abs(per(p.x,p.t).^2);
p.initial = @(~,p) per(p.x,p.origins(1))+zeros(4,1,1);
p.compare = {@(p) sol(p),@(p) sol(p),@(p) sol(p),@(p) sol(p)};
p.observe = {@(a,p) a(1,:),@(a,p) a(2,:),...
@(a,p) a(3,:),@(a,p) a(4,:)};
p.output = {@(o,p) abs(o{1}).^2,@(o,p) abs(o{2}).^2,...
@(o,p) abs(o{3}).^2,@(o,p) abs(o{4}).^2};
p.name = 'Peregrine solution';
p.steps = 20;
p.deriv = @(a,w,p) 1i*a.*((conj(a).*a));
p.linear = @(p) 0.5*1i*p.Dx.^2;
e = xspde(p);
end

function [p,varargout] = per(x,t)
% Generates peregrine solutions with alpha = 1/2, beta = A0 = 1

p = exp(1i*t).*(4*(1+2*1i*t)./(1+4.*(t.^2+x.^2))-1);
if nargout == 2
dp = -8*x.*exp(1i*t).*(4*(1+2*1i*t)./(1+4.*(t.^2+x.^2)).^2);
varargout{1} = dp;
end

end
function bound = boundval(~,~,~,p)
% Generates nonzero, time dependent boundary values
[p,dp] = per(p.origins(2),p.t);
bound = {p,p;dp,-dp;p,-dp;dp,p};
end
```



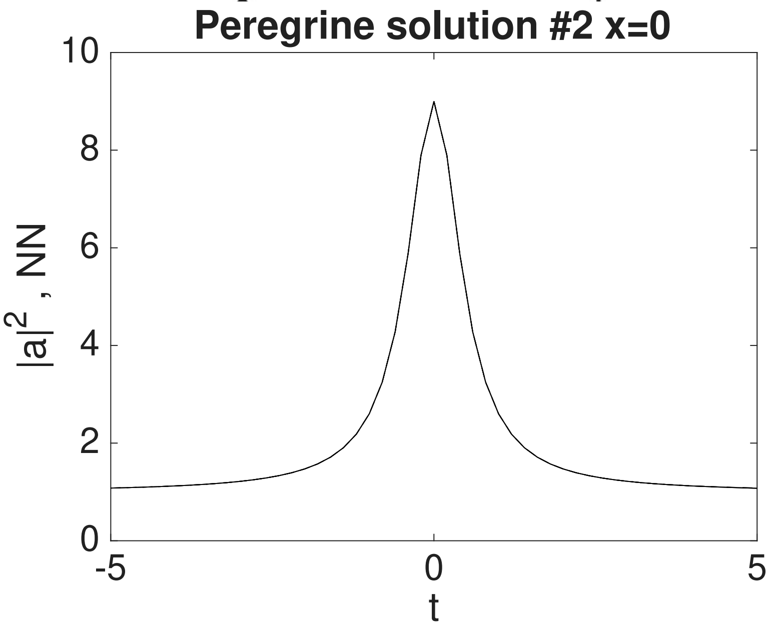


Figure 6.7.: *Top figure: Peregrine solution with Dirichlet-Dirichlet boundaries in space-time.*
Bottom figure: Peregrine solution with Neumann-Neumann boundaries at $x = 0$.

Part IV.

Quantum phase-space

7. Phase-space toolbox

This chapter describes how to use the xSPDE numerical toolbox to solve network and quantum dynamical problems in phase-space. For theoretical background, see Chapter (8). For detailed examples, see Chapter (9).

7.1. Quantum phase-space

Initially developed in the first half of 20th century, phase-space representations of quantum mechanics have been extensively developed, and utilized to simulate large bosonic Hilbert spaces, as well as small ones. Multiple such methods exist, including the classical Wigner [69], Husimi Q and Glauber-Sudarshan [70, 71] P representations, as well as the non-classical positive-P [72] and gauge-P [73] methods. If the reader is unfamiliar with these methods, they are referred to the theory sections in Chapter 8 and the original literature to obtain further explanations of these methods.

These different approaches have areas of applicability that depend on the Hilbert space dimension, as explained below.

7.1.1. Phase-space methods

This is used for phase-space mappings. Details are:

p.phase = 1 - for a normally-ordered P or positive-P representation.

p.phase = 2 - for a symmetrically-ordered Wigner-representation.

p.phase = 3 - for an anti-normally-ordered Q-representation.

7.2. Laser amplification noise

Laser quantum noise is commonly modeled [74–76] using SDEs in a normally ordered quantum phase-space representation. Consider a model for the quantum noise of a single mode laser as it turns on, near threshold:

$$\dot{a} = ga + bw(t) \tag{7.1}$$

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

$$\left\langle w(t)w^*(t')\right\rangle = 2\delta\left(t - t'\right) . \tag{7.2}$$

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

As an example, try the following:

- **Solve for the case of $g = 0.25$, $b = 0.01$**

using xSPDE input is:

```
clear
p.noises = 2;
p.observe = @(a,p) abs(a).^2;
p.olabels = '|a|^2';
p.deriv = @(a,w,p) 0.25*a + 0.01*(w(1)+1i*w(2));
xspde(p);
```

This input script can either be copied into a new script, or simply pasted into the Command Window, noting that one should usually type clear first when starting new interactive simulations. Most lasers have many photons and hence much less noise than this. At this small gain, numerical errors are negligible, so the program reports only small RMS average errors:

- Errors: Step=0.000621 Samp=0 Diff=0 Chisq/k=0

For larger gain, error-bars will display on the graph. These are calculated from the difference between using steps of size dt and steps of size $dt/2$. They only appear if greater than a minimum relative size, typically 1% of the graph size, which can be set by the user. Here the time-step errors are too small to be graphed, there is no sampling error because only one trajectory is requested, and there is neither difference nor χ^2 error, since there are no comparisons specified.

Note that specifying `p.phase = 1` is not required, even though the phase-space is normally-ordered, which is the default phase-space method. This is because there is no vacuum noise in this classical noise case, and only the simple Glauber-Sudarshan P-representation is needed. Non-classical states require a positive P-representation.

7.3. Input parameters

More complex input parameters are stored in a structure which is input to the program. This is a superset of the parameters already defined. In the definitions below, the structure name is omitted, but we use p in the examples.

While the quantum monte-carlo and phase-space toolboxes have some common parameters, they are distinct toolboxes that correspond to different Hilbert space bases. One must choose one or the other by either setting $quantum > 0$ or $phase > 0$. The defaults used are given in the phasepreferences function.

Common parameters with the standard SDE and SPDE simulations include ensembles, noises, inrandoms, points, steps, ranges. All have default values, but other choices can be input to over-ride the default values.

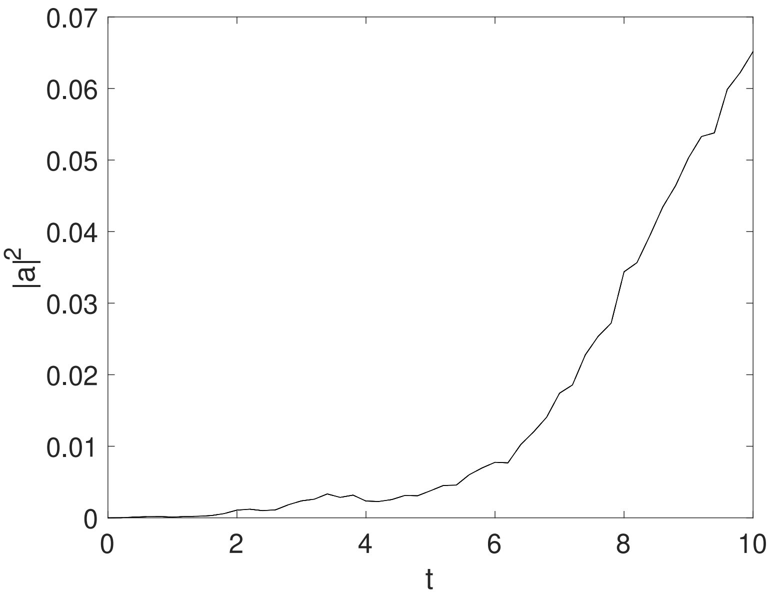


Figure 7.1.: *Simulation of the stochastic equation describing a laser turning on.*

7.3.1. Phase-space parameters

Phase-space simulations can be initiated with a set of parameters that generate a default initial function. This is suitable for network simulations with gaussian inputs of squeezed and thermalized states.

dimension is for the type of calculation. For network transformations, dimension = 0.

sqz is the initial squeezing parameter vector, $\boldsymbol{r}$, per mode.

alpha is the initial coherent amplitude vector per mode.

matrix is a matrix transformation function used to create network transformations.

tr is an amplitude transmission vector used to attenuate the initial state.

thermal is a thermal fraction vector used to initialize a thermalized squeezed state.

initial functions initialise the equations. If omitted, the initial conditions are defined using the parameters above, as definied by the default option, *Initialgaussian.*

7.4. Gaussian boson sampling outputs

If one is interested in computing output probabilities for Gaussian boson sampling (GBS) photonic quantum networks, observe functions are available for comparisons with either experimental data, or exactly known test cases.

As every experimental group treats its data differently, xSPDE4 does not attempt to provide universal code for extracting and binning experimental data. This is provided in a seperate code. Instead, xSPDE4 focuses on simulating such networks in phase-space.

If p.dimension=0, where the mode index can be large, the mode index is treated as a space index rather than a line index. This aids graphical rendering with xGRAPH.

7.4.1. Phase-space support functions

The quantum phase-space suppport functions include a gaussian initialisation, binned photon-counting probability distributions and marginal moments of observables. The current list is given below:

Label	Return type	Description
Initialgaussian	field	Initialises phase-space fields to a Gaussian
X	vector	x-quadrature per channel
X2	vector	x-quadrature squared per channel
Y	vector	y-quadrature per channel
Y2	vector	Y-quadrature squared per channel
Pn	vector	Output photon number, $\hat{n}'_j$, per channel
Nm	vector	Output photon number correlation in sequence $\hat{n}'_1\hat{n}'_2\ldots$
K	vector	Clicks $\hat{\pi}_j(1)$ per channel
Km	vector	Click correlation in sequence $\hat{\pi}_1(1)\hat{\pi}_2(1)\ldots$
Km2	vector	Click correlation over two channels $\hat{\pi}_j(1)\hat{\pi}_k(1)$
Km3	vector	Click correlation over three channels $\hat{\pi}_j(1)\hat{\pi}_k(1)\hat{\pi}_h(1)$
Kmsub	vector	Subset of click correlations per CO channels
K1	vector	Binned click probability - single partition
N1	vector	Binned photon number probability - single partition
Kn	array	Binned click probability - n-fold partition
Nn	array	Binned photon number probability - n-fold partition

7.4.2. Compare functions

The compare functions are used for testing and for experimental data. The comparisons may include error data. This generates comparison plots, as well as error totals that are converted into a χ^2- error estimate when there are statistical variances.

Standard compare functions given below are analogous to their observe counterparts above, with compare functions denoted by the c at the end of each label. The test cases are only applicable when the output is unchanged from the input apart from a transmission factor $p.tr$, and when the input is a thermalised or pure squeezed state. For photon-number resolving (PNR) detector comparisons, uniform squeezing parameters are required for the inputs.

Functions labeled with an asterisk can be used with Gaussian states having a coherent component *p.alpha*:

Label	Return type	Description
Xc	vector	x-quadrature per channel
X2c	vector	x-quadrature squared per channel
Yc	vector	y-quadrature per channel
Y2c	vector	y-quadrature squared per channel
*Pnc	vector	Output photon number, $\hat{n}'_j$, per channel
Nmc	vector	Output photon number correlation in sequence $\hat{n}'_1\hat{n}'_2\dots$
Kc	vector	Clicks $\hat{\pi}_j(1)$ per channel
Kmc	vector	Click correlation in sequence $\hat{\pi}_1(1)\hat{\pi}_2(1)\dots$
Km2c	vector	Click correlation over two channels $\hat{\pi}_j(1)\hat{\pi}_k(1)$
Km3c	vector	Click correlation over three channels $\hat{\pi}_j(1)\hat{\pi}_k(1)\hat{\pi}_h(1)$
Kmsubc	vector	Subset of click correlations per CO channels
K1c	vector	Binned click probability - single partition
N1thc	vector	Thermal input photon number probability - single partition
N1lsc	vector	Pure squeezed input photon number probability - single partition
Knc	array	Binned click probability - n-fold partition
Nnthc	array	Thermal input photon number probability - n-fold partition

7.5. Sampling methods in phase-space

Generating initial samples of a phase-space distribution is the first step of any phase-space simulation, be that dynamical or input-output, such as for GBS. In most cases, analytical forms of the initial samples exist, which are straightforard to implement numerically.

The section outlines the sampling methods used to simulate GBS input and output distributions in phase-space, and perform comparisons with either experimental data, or exact tests.

7.5.1. Input-output samples

Simulating dynamic and static quantum systems in phase-space follows the same, general procedure, at least initially. In dynamical applications, initial stochastic samples generated from some phase-space distribution. These can be from a default gaussian description of coherent, squeezed or thermal states, given by the *alpha*, *sqz*, *thermal* and *tr* parameters, or else from an *initial* function, which takes precedence over the gaussian parmeters.

To simulate quantum networks in phase-space, one must first generate initial stochastic samples. This is achieved using the σ-ordering scheme (see Subsection 8.1.5 for theory) as stochastic samples for any Gaussian input state in any representation are generated following:

$$\begin{aligned}\alpha_j &= \frac{1}{2}\left(\Delta_{\sigma x_j} w_j + i\Delta_{\sigma y_j} w_{j+M}\right)\\ \beta_j &= \frac{1}{2}\left(\Delta_{\sigma x_j} w_j - i\Delta_{\sigma y_j} w_{j+M}\right),\end{aligned} \tag{7.3}$$

where $\langle w_j w_k\rangle = \delta_{jk}$ are real Gaussian noises and

$$\begin{aligned}\Delta^2_{\sigma x_j} &= 2(n_j + \sigma + \tilde{m}_j)\\ \Delta^2_{\sigma y_j} &= 2(n_j + \sigma - \tilde{m}_j),\end{aligned} \tag{7.4}$$

are thermal squeezed state quadrature variances which are altered from the pure squeezed state definitions, (8.50) and (8.51).

For normally ordering, the input amplitudes $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are converted to outputs as

$$\begin{aligned}\boldsymbol{\alpha}' &= \boldsymbol{T}\boldsymbol{\alpha}\\ \boldsymbol{\beta}' &= \boldsymbol{T}^*\boldsymbol{\beta},\end{aligned} \tag{7.5}$$

which follows from Eq.(8.39). However for non-normally ordered methods, additional vacuum noise arising from the reservoir modes must be included.

This is achieved using a hermitian decoherence matrix

$$\boldsymbol{D} = \boldsymbol{I} - \boldsymbol{T}^\dagger\boldsymbol{T}, \tag{7.6}$$

with decomposition $\boldsymbol{D} = \boldsymbol{U}\boldsymbol{\lambda}^2\boldsymbol{U}^\dagger$ where $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{\lambda}\boldsymbol{U}^\dagger$ is the matrix square root and $\boldsymbol{\lambda}$ is a diagonal, positive matrix. The output amplitudes when $\sigma > 0$ are then obtained as

$$\boldsymbol{\alpha}' = \boldsymbol{T}\boldsymbol{\alpha} + \sqrt{\frac{\sigma}{2}}\boldsymbol{B}(\boldsymbol{u} + i\boldsymbol{v}), \tag{7.7}$$

where $\boldsymbol{\beta}' = \boldsymbol{\alpha}'^*$ as these are a classical phase-space.

7.5.2. Grouped correlations computation: Threshold detectors

Grouped correlations, or grouped count probabilities (GCPs), are binned photon counting probability distributions that can be simulated in xSPDE4 for photon-counting setups using both photon-number resolving (PNR) or threshold detection. These are defined analytically in Section 8.10, where all the nessecary background theory is presented in Section 8.6.

These observables are readily simulated in phase-space using the positive-P representation. For threshold detectors, one replaces normally ordered projection operator Eq.(8.61) with the positive-P observable

$$\pi_i(c_i) =: e^{-n'_i}\left(e^{n'_i}-1\right)^{c_i}, \tag{7.8}$$

where $n'_i = \alpha'_i\beta'_i$ is the output photon number.

The summation over exponentially many patterns implemented by GCPs (see Eq.(8.77) is simulated using a multidimensional inverse discrete Fourier transform

$$\begin{aligned}\tilde{\mathcal{G}}^{(n)}_{\boldsymbol{S}}(\boldsymbol{k}) &= \left\langle \prod_{j=1}^{d}\bigotimes_{i\in S_j}\left(\pi_i(0)+\pi_i(1)e^{-ik_j\theta_j}\right)\right\rangle_P ,\\ \mathcal{G}^{(n)}_{\boldsymbol{S}}(\boldsymbol{m}) &= \frac{1}{\prod_j (M_j+1)}\sum_{\boldsymbol{k}}\tilde{\mathcal{G}}^{(n)}_{\boldsymbol{S}}(\boldsymbol{k})e^{i\sum k_j\theta_j m_j},\end{aligned} \tag{7.9}$$

where $\theta_j = 2\pi/(M_j+1)$ and $k_j = 0,\dots,M_j$.

The Fourier transform removes all patterns which don't contain $\boldsymbol{m}$ counts, in doing this the Fourier transform simulates all possible correlations generated in a network. This reduces an otherwise computationally complex task into a highly efficient and scalable one, allowing comparisons to be performed on experimental correlations of any order.

7.5.3. Grouped correlations computation: PNR detectors

To simulate GCPs of PNR detectors, we are interested in the total number of output photons contained in each subset S_j defined in operator form

$$\hat{n}'_{S_j} = \sum_{i\in S_j}\hat{n}'_i. \tag{7.10}$$

As was the case for threshold detectors, we replace this total number operator with the phase-space observable

$$n'_{S_j} = \sum_{i\in S_j} n'_i, \tag{7.11}$$

such that the actual GCP computation is performed as

$$\mathcal{G}^{(n)}_{\boldsymbol{S}}(\boldsymbol{m}) = \iint P(\boldsymbol{\alpha},\boldsymbol{\beta})\left[\prod_{j=1}^{d}\left[\frac{1}{m_j!}:(n'_{S_j})^{m_j}e^{-n'_{S_j}}:\right]\right]\mathrm{d}^2\boldsymbol{\alpha}\mathrm{d}^2\boldsymbol{\beta}. \tag{7.12}$$

We note that this method allows one to avoid the multi-dimensional Fourier transform needed for the threshold detector distribution. The Fourier transform has the effect of introducing Fourier overheads which can slow down simulations of multi-dimensional distributions, especially for dimensions $d \geq 4$.

8. Quantum phase-space theory

This chapter describes the background of quantum phase-space theory, including dynamical problems obtained by transforming the master equation into a second-order partial differential equation called the Fokker-Planck equation (FPE), as well as input-output problems such as GBS.

8.1. Phase-space representations

Phase-space representations are an alternative description of quantum mechanics where one maps operators and fields of various orderings to classical probability distributions on a phase-space. This alternative description was first introduced by Wigner [69], and has since burgeoned into a vast field with applications to quantum optics, atom-optics, quantum information, and many more.

In this section, we introduce the phase-space representations implemented in xSPDE4, although one is not restricted to these representations. The outputs in phase-space are continuous real or complex variables whose stochastic moments are equal to quantum expectation values, including probability distributions obtained from experimental data such as photon counting experiments.

This assumes that the parameters are precisely known, and do not have noise or fluctuations. Even then, some differences from sampling errors due to finite numbers of experimental and theoretical counts.

For clarity, throughout this chapter, hats like $\hat{a}$ are used to indicate operators that do not commute with each other, as opposed to stochastic variables like α that do commute. For any given operator ordering, it is always possible to find a probability distribution such that the expectation of an operator product equals the stochastic variable correlations [77].

8.1.1. Glauber-Sudarshan P-representation

The M-mode Glauber-Sudarshan P-representation expands the density matrix as a sum of diagonal coherent state projectors

$$\hat{\rho} = \int P(\boldsymbol{\alpha}) \left|\boldsymbol{\alpha}\right\rangle \left\langle\boldsymbol{\alpha}\right| \mathrm{d}^{2M}\boldsymbol{\alpha}, \tag{8.1}$$

where $\left|\boldsymbol{\alpha}\right\rangle = \bigotimes_{j=1}^{M} \left|\alpha_j\right\rangle$ is a multimode coherent state eigenvector with corresponding eigenvalues $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_M]$. Coherent states [70] are the right eigenstate of the annihilation operator, $\hat{a}\left|\alpha\right\rangle = \alpha\left|\alpha\right\rangle$, and are the most commonly used basis state to define phase-space representations, although other bases are possible, such as quadratures.

For classical states such as coherent and thermal states, the P-distribution $P(\boldsymbol{\alpha})$ satisfies the mathematical requirements of a probability distribution: Real, positive, non-singular and normalizable with

$$\int P(\boldsymbol{\alpha})\mathrm{d}^{2M}\boldsymbol{\alpha} = 1. \tag{8.2}$$

The P-representation breaks down for certain quantum states, giving non-positive and singular distributions. This is due to the lack of off-diagonal coherent state amplitudes required to represent nonclassical superpositions.

8.1.2. Positive P-representation

Part of a family of generalized P-representations developed to extend the Glauber-Sudarshan P-representation to quantum states [72], the normally ordered positive P-representation always generates a non-singular and positive distribution for any quantum state. The trade-off is that it is non-unique, which can lead to growing sampling errors for nonlinear Hamiltonians.

In the positive P-representation, the density matrix is defined as an expansion over a multidimensional subspace of the complex plane:

$$\hat{\rho} = \iint P(\boldsymbol{\alpha},\boldsymbol{\beta})\hat{\Lambda}(\boldsymbol{\alpha},\boldsymbol{\beta})\mathrm{d}^{2M}\boldsymbol{\alpha}\mathrm{d}^{2M}\boldsymbol{\beta}, \tag{8.3}$$

where $P(\boldsymbol{\alpha},\boldsymbol{\beta})$ is the positive-P distribution over coherent state amplitudes $\boldsymbol{\alpha},\boldsymbol{\beta}$ satisfying the normalization condition

$$\iint P(\boldsymbol{\alpha},\boldsymbol{\beta})\mathrm{d}^{2M}\boldsymbol{\alpha}\mathrm{d}^{2M}\boldsymbol{\beta} = 1. \tag{8.4}$$

The projector

$$\hat{\Lambda}(\boldsymbol{\alpha},\boldsymbol{\beta}) = \frac{|\boldsymbol{\alpha}\rangle\langle\boldsymbol{\beta}^*|}{\langle\boldsymbol{\beta}^*|\boldsymbol{\alpha}\rangle} = e^{-\boldsymbol{\alpha}\cdot\boldsymbol{\beta}+\frac{1}{2}|\boldsymbol{\alpha}|^2+\frac{1}{2}|\boldsymbol{\beta}|^2}|\boldsymbol{\alpha}\rangle\langle\boldsymbol{\beta}^*|, \tag{8.5}$$

is responsible for the doubled classical phase-space dimension. This allows off-diagonal amplitudes $\boldsymbol{\beta} \neq \boldsymbol{\alpha}^*$ to exist, such that $\boldsymbol{\alpha},\boldsymbol{\beta}$ are now independent, generating two pairs of complex amplitudes.

One can restrict the distribution to a classical phase-space with $\boldsymbol{\beta} = \boldsymbol{\alpha}^*$, in which case the diagonal P-representation is obtained as a special case of the positive P-representation via the substitution $P(\boldsymbol{\alpha},\boldsymbol{\beta}) = P(\boldsymbol{\alpha})\delta(\boldsymbol{\alpha}^* - \boldsymbol{\beta})$. As stated above, for non-classical states such as squeezed or Fock states, this will lead to the well known singular behavior of the distribution.

Moments of the positive-P distribution are equivalent to normally ordered operator moments

$$\begin{aligned}\left\langle \hat{a}^{\dagger}_{j_1}, \ldots, \hat{a}_{j_n} \right\rangle &= \langle \beta_{j_1}, \ldots, \alpha_{j_n} \rangle_P \\ &= \iint P(\boldsymbol{\alpha}, \boldsymbol{\beta}) [\beta_{j_1}, \ldots, \alpha_{j_n}] \mathrm{d}^{2M}\boldsymbol{\alpha} \mathrm{d}^{2M}\boldsymbol{\beta}, \end{aligned} \tag{8.6}$$

where $\langle \ldots \rangle$ denotes a quantum expectation value and $\langle \ldots \rangle_P = \langle \ldots \rangle_{P,\infty}$ is the positive-P average in the limit of an infinite ensemble of stochastic trajectories.

8.1.3. Wigner representation

Although the diagonal P-representation is unsuitable for simulating non-classical states, not all classical phase-space distributions suffer from the same limitations. The symmetrically ordered Wigner representation and anti-normally ordered Q-function both produce positive, well defined distributions for Gaussian quantum states such as squeezed states. Unfortunately, for non-Gaussian non-classical states such as Fock and Schrödinger cats states, the resulting Wigner distribution is negative, and hence not a probability distribution.

The negativity of the Wigner distribution for some states is why its commonly referred to as a quasi-probability, and for an M-mode system is defined as the Fourier transform of the symmetrically ordered characteristic function such that

$$W(\boldsymbol{\alpha}) = \frac{1}{\pi^{2M}} \int \mathrm{d}^2\boldsymbol{z} \mathrm{Tr}\left(\hat{\rho} e^{i\boldsymbol{z}(\hat{a}-\boldsymbol{\alpha}) + i\boldsymbol{z}^*(\hat{a}^\dagger - \boldsymbol{\alpha}^*)} \right), \tag{8.7}$$

where $\mathrm{Tr}\,(\ldots)$ is the matrix trace and $\boldsymbol{z}$ is a complex vector. For hermitian operators such as the density operator, the Wigner distribution always exists as a real-valued function on phase-space. This isn't the case for non-hermitian operators, in which case Wigner distribution becomes complex.

The Wigner representation is directly applicable to symmetrically ordered operator products. Symmetric ordering, denoted $\{\ldots\}_{sym}$, is the average over all possible combinations of creation and annihilation operators, for example:

$$\{\hat{a}^\dagger \hat{a}\}_{sym} = \frac{1}{2}(\hat{a}\hat{a}^\dagger + \hat{a}^\dagger \hat{a}) \tag{8.8}$$

$$\{\hat{a}^\dagger \hat{a}^2\}_{sym} = \frac{1}{3}(\hat{a}^2\hat{a}^\dagger + \hat{a}\hat{a}^\dagger\hat{a} + \hat{a}^\dagger\hat{a}^2). \tag{8.9}$$

Therefore, moments of the Wigner distribution correspond to symmetrically ordered operator moments

$$\begin{aligned}\left\langle \{\hat{a}^{\dagger}_{j_1}, \ldots, \hat{a}_{j_n}\}_{sym} \right\rangle &= \langle \alpha^*_{j_1}, \ldots, \alpha_{j_n} \rangle_W \\ &= \int W(\boldsymbol{\alpha}) [\alpha^*_{j_1}, \ldots, \alpha_{j_n}] \mathrm{d}^2\boldsymbol{\alpha}, \end{aligned} \tag{8.10}$$

where $\langle \dots \rangle_W = \langle \dots \rangle_{W,\infty}$ is the Wigner ensemble average.

For applications to photon-counting experiments, the Wigner representation is suitable for simulating quadrature operators as measured by homodyne detectors, which are symmetrically ordered. The ordering requirement makes applications to normally-ordered detectors both cumbersome, as one must reorder all operators to normal order, for example:

$$\left\langle \{\hat{a}^\dagger a\}_{sym} \right\rangle = |\alpha|^2 + \frac{1}{2}, \tag{8.11}$$

as well as inaccurate. The additional term arises from the reordering, as the Wigner representation adds half a quantum of vacuum noise per mode, causing a rapid increase sampling errors. Hence, the Wigner representation is unsuitable for simulations of normally-ordered photon counting probabilities.

8.1.4. Q-function

The standard form of the anti-normally ordered, e.g. $\hat{a}\hat{a}^\dagger$, M-mode Q-function is

$$Q(\boldsymbol{\alpha}) = \frac{1}{\pi^M} \left\langle \boldsymbol{\alpha} | \hat{\rho} | \boldsymbol{\alpha} \right\rangle, \tag{8.12}$$

and, like the Wigner function, can be expressed as the Fourier transform of the anti-normally ordered characteristic function.

Unlike the Wigner distribution, the Q-function distribution is always positive for any classical or non-classical state but is only defined for anti-normally ordered operator products with moments being obtained as

$$\begin{aligned} \left\langle \hat{a}_{j_1}, \dots, \hat{a}^\dagger_{j_n} \right\rangle &= \left\langle \alpha_{j_1}, \dots, \alpha^*_{j_n} \right\rangle_Q \\ &= \int Q(\boldsymbol{\alpha}) [\alpha_{j_1}, \dots, \alpha^*_{j_n}] \mathrm{d}^{2M}\boldsymbol{\alpha}, \end{aligned} \tag{8.13}$$

where $\langle \dots \rangle_Q = \langle \dots \rangle_{Q,\infty}$ denotes a Q-distribution ensemble average.

Like the Wigner function, this ordering requirment means that for applications to normally-ordered photon counting experiments, or any normally-ordered measurement, operators must be reordered. This is readily illustrated using the standard bosonic commutation relations Eqs.(8.41), where anti-normal ordered expectation value $\left\langle a\hat{a}^\dagger \right\rangle$ is reordered to

$$\left\langle a\hat{a}^\dagger \right\rangle = |\alpha|^2 + 1. \tag{8.14}$$

Therefore, the Q-function adds an entire quantum of vacuum noise per mode, generating the largest increase in sampling errors of any phase-space representation when used to simulate a normally ordered measurement. This accumulation of vacuum noise for multimode linear photonic networks rapidly causes Q-function simulations to become inaccurate.

8.1.5. σ-ordering

The amount of vacuum noise added by each representation can be used to define the operator ordering parameter σ, where $\sigma = 0$ corresponds to normal ordering, $\sigma = 1/2$ symmetric ordering and $\sigma = 1$ anti-normal ordering.

From the ability to define a common ordering scheme arises the ability to define a σ-ordered phase-space distribution. Since the Wigner and Q-function distributions can be defined as convolutions of the positive P-representation, a σ-ordered representation is defined as:

$$P_\sigma(\boldsymbol{\alpha}) = \frac{1}{(\pi\sigma)^M} \int P(\boldsymbol{\alpha}_0, \boldsymbol{\beta}_0) e^{-(\boldsymbol{\alpha}-\boldsymbol{\alpha}_0)(\boldsymbol{\alpha}^*-\boldsymbol{\beta}_0)/\sigma} \mathrm{d}^{2M}\boldsymbol{\alpha}\mathrm{d}^{2M}\boldsymbol{\beta}. \tag{8.15}$$

Here, $P_\sigma(\boldsymbol{\alpha})$ is a σ-ordered distribution, $P(\boldsymbol{\alpha}_0, \boldsymbol{\beta}_0)$ is the positive-P distribution and $\boldsymbol{\alpha}_0, \boldsymbol{\beta}_0$ are used to denote the normal-ordered non-classical phase-space variables, whilst $\boldsymbol{\alpha}, \boldsymbol{\alpha}^*$ denote a classical phase-space which is valid for $\sigma = 1/2, 1$.

Operator moments for any ordering can now be obtained via

$$\begin{aligned}\left\langle \left\{ \hat{a}^\dagger_{j_1}, \ldots, \hat{a}_{j_n} \right\}_\sigma \right\rangle &= \left\langle \alpha^*_{j_1}, \ldots, \alpha_{j_n} \right\rangle_\sigma \\ &= \int P_\sigma(\boldsymbol{\alpha}) [\alpha^*_{j_1}, \ldots, \alpha_{j_n}] \mathrm{d}^{2M}\boldsymbol{\alpha},\end{aligned} \tag{8.16}$$

where, as above, $\langle \ldots \rangle_\sigma = \langle \ldots \rangle_{\sigma,\infty}$ is a σ-ordered ensemble average.

8.2. Dynamics in phase-space

The density matrix master equation is basic to quantum theory, particularly for open quantum systems, the theory of which will be treated in more detail in Chapter 11. The master equations of most interest in this chapter have the standard Lindblad form:

$$\begin{aligned}\dot{\hat{\rho}} &= \mathcal{L}_J \hat{\rho} \\ &= -i\left[\hat{H}, \hat{\rho}\right] + \sum_{j=1}^{J} \gamma_j \left(2\hat{L}_j \rho \hat{L}_j^\dagger - \hat{L}_j^\dagger \hat{L}_j \rho - \rho \hat{L}_j^\dagger \hat{L}_j \right).\end{aligned} \tag{8.17}$$

Here, $\mathcal{L}_J$ is the total super-operator for J terms, $\hat{H}$ is the reversible system Hamiltonian, $\hat{L}_j$ are J operators that couple the system to the dissipative reservoir, and γ_j is the decay rate. The dissipative operators can be further classified by type n and mode index k, including vector indices if needed.

Solving the master equation using orthogonal basis methods is impractical, as the master equation has a memory requirement that scales as $e^{2\lambda M}$ for M modes, where $\lambda = \ln(N_{max})$, and N_{max} is the dimension of the Hilbert space of a single mode. As $M \to \infty$, the Hilbert space dimension grows exponentially, limiting orthogonal basis methods to small mode numbers as the memory and CPU time grows rapidly. Therefore, large quantum systems like linear photonic networks or quantum fields are inaccessible, apart from using various approximations like mean-field or tensor network methods. Although even tensor networks eventually succumb to computational limitations.

One possible way to treat such large quantum systems is via quantum phase-space expansions. These methods convert the master equation into a stochastic differential equation, which are often more scalable than other methods [78, 79]. There are trade-offs, and this often may require further approximations.

The advantage arises from the phase-space distribution being sampled using random sampling, where each sample in phase-space requires a polynomial amount of storage, typically growing linearly with the number of modes. From this sampling procedure, it is also usually relatively straightforward to estimate sampling errors.

This approach started when Schrödinger [80] pointed out that quantum oscillators can have classical equations. This was extended to other systems [69, 70, 81], especially including lasers and quantum optics [2, 74–76].

8.2.1. Operator mappings

To perform dynamical simulations of a quantum system requires first mapping the master equation into a second-order partial differential equation called a Fokker-Planck equation (FPE) (see Chapter 2 for a theoretical review of FPEs). To do this, one requires a mapping between operator products, such as $\hat{a}_j^\dagger \hat{\rho}$, and partial derivatives of phase-space distributions, i.e. $\frac{\partial}{\partial \alpha_j} W(\boldsymbol{\alpha})$.

Using the σ-ordered notation, unified operator identities can be defined to perform this differential equation mapping as:

$$\begin{aligned}
\hat{a}_j^\dagger \hat{\rho} &\rightarrow \left[\beta_j + (\sigma - 1)\frac{\partial}{\partial \alpha_j}\right] P_\sigma \\
\hat{a}_j \hat{\rho} &\rightarrow \left[\alpha_j + \sigma \frac{\partial}{\partial \beta_j}\right] P_\sigma \\
\hat{\rho} \hat{a}_j &\rightarrow \left[\alpha_j + (\sigma - 1)\frac{\partial}{\partial \beta_j}\right] P_\sigma \\
\hat{\rho} \hat{a}_j^\dagger &\rightarrow \left[\beta_j + \sigma \frac{\partial}{\partial \alpha_j}\right] P_\sigma \, .
\end{aligned} \tag{8.18}$$

If the resulting differential equation obtained from this mapping has a second-order positive-definite form it is an FPE, which is equivalent to an SDE (see Chapter 2 for theoretical background on SDEs), or an SPDE for quantum fields [82]. The noise can be additive or multiplicative, depending on the problem. Although we have defined a unified mapping for any ordered phase-space representation, not all methods give stable FPE equations [73], such as the positive P-representation, which can suffer from boundary-term corrections [73, 83]. The Wigner representation meanwhile requires a truncation of larger than second-order derivatives if the Hamiltonian is nonlinear [84], hence simulations only approximate the system dynamics, although this can be accurate in some cases [84]. The FPE obtained from the Q-function on the other hand is no longer positive-definite, which is a requirement for FPEs.

The total noise includes internal quantum noise generated from the Hamiltonian term $\hat{H}_{sys}$, as well as reservoir noise terms generated from the coupling to the reservoir operators, which is proportional to the damping rate Γ_j. There is a similar behavior in classical systems, except that these correspond to a high-temperature limit, and in most cases only have external reservoir noise from thermal fluctuations.

8.3. Damped harmonic oscillator

As an example, take the driven quantum harmonic oscillator. This has the Hamiltonian

$$\hat{H}/\hbar = i\mathcal{E}\left(\hat{a}^\dagger - \hat{a}\right) + \omega_0 \hat{a}^\dagger \hat{a}, \tag{8.19}$$

where $\mathcal{E}$ is the driving amplitude, and ω_0 the harmonic oscillator frequency. If damping is added, it obeys the master equation

$$\begin{aligned}\frac{d\hat{\rho}}{dt} &= -i[i\left(\mathcal{E}\hat{a}^\dagger - \mathcal{E}^*\hat{a}\right) + \omega_0\hat{a}^\dagger\hat{a}, \rho] + \gamma\left(1+\bar{n}\right)\left(2\hat{a}\rho\hat{a}^\dagger - \hat{a}^\dagger\hat{a}\rho - \rho\hat{a}^\dagger\hat{a}\right) \\ &\quad + \gamma\bar{n}(2\hat{a}^\dagger\rho\hat{a} - \hat{a}\hat{a}^\dagger\rho - \rho\hat{a}\hat{a}^\dagger),\end{aligned} \tag{8.20}$$

where $\bar{n}$ is the temperature reservoir occupation (see Chapter 11).

This leads to a random walk in a complex space [1, 2]:

$$\begin{aligned}\frac{d\alpha}{dt} &= \mathcal{E} - \left(\gamma + i\omega_0\right)\alpha + \sqrt{2\gamma\left(\sigma + \bar{n}\right)}\zeta(t) \\ \frac{d\beta}{dt} &= \mathcal{E}^* - \left(\gamma - i\omega_0\right)\beta + \sqrt{2\gamma\left(\sigma + \bar{n}\right)}\zeta^*(t),\end{aligned} \tag{8.21}$$

where the noise is complex and $\zeta(t) = \left(w_1(t) + iw_2(t)\right)/\sqrt{2}$. The correlations are

$$\begin{aligned}\left\langle \zeta(t)\left(\zeta(t')\right)^*\right\rangle &= \delta\left(t-t'\right) \\ \left\langle \zeta(\omega)\left(\zeta\left(\omega'\right)\right)^*\right\rangle &= \delta\left(\omega-\omega'\right).\end{aligned} \tag{8.22}$$

8.3.1. Wigner dynamics

In the undriven, zero temperature Wigner case with γ =1, $\sigma = 1/2$, and in a rotating frame so that $\omega_0 = 0$, using the mappings Eq.(8.18), the probability follows the Fokker-Planck equation:

$$\frac{\partial P_{1/2}}{\partial t} = \left[\frac{\partial}{\partial\alpha_x}\alpha_x + \frac{\partial}{\partial\alpha_y}\alpha_y + \frac{1}{4}\left(\frac{\partial^2}{\partial\alpha_x^2} + \frac{\partial^2}{\partial\alpha_y^2}\right)\right]P_{1/2}\,, \tag{8.23}$$

which is an example of Eq.((2.27)). Ignoring terms that vanish or can be obtained from symmetry, the first corresponding moment equations in each of the real and imaginary directions are

$$\begin{aligned}\frac{\partial}{\partial t}\langle\alpha_x\rangle &= \left\langle -\alpha_x\frac{\partial}{\partial\alpha_x}\alpha_x\right\rangle = -\left\langle\alpha_x\right\rangle\\ \frac{\partial}{\partial t}\langle\alpha_x\alpha_y\rangle &= \left\langle -\left(\alpha_x\frac{\partial}{\partial\alpha_x}+\alpha_y\frac{\partial}{\partial\alpha_y}\right)\alpha_x\alpha_y\right\rangle = -\left\langle\alpha_x\alpha_y\right\rangle\\ \frac{\partial}{\partial t}\langle\alpha_x^2\rangle &= \left\langle\left(-\alpha_x\frac{\partial}{\partial\alpha_x}+\frac{1}{4}\frac{\partial^2}{\partial\alpha_x^2}\right)\alpha_x^2\right\rangle = \frac{1}{2}-2\left\langle\alpha_x^2\right\rangle. \end{aligned} \tag{8.24}$$

The steady-state is therefore a Gaussian distribution with $\langle\alpha_{x,y}\rangle = 0$, $\langle\alpha_x\alpha_y\rangle = 0$ and $\left\langle\alpha_{x,y}^2\right\rangle = 1/4$. One can use an initial condition of $\alpha = (v_1+iv_2)/2$, with $\left\langle v_i^2\right\rangle = 1/2$, in order to replicate the steady state, which is a Gaussian with$\langle\alpha_x\rangle = \langle\alpha_y\rangle = 0$ and$\left\langle\alpha_x^2\right\rangle = \left\langle\alpha_y^2\right\rangle = 1/4$.

8.3.2. Internal spectrum

Neglecting any boundary terms, the equation in frequency space is:

$$-i\omega\tilde{\alpha}(\omega) = -\tilde{\alpha}(\omega)+\tilde{\zeta}(\omega). \tag{8.25}$$

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

$$\tilde{\alpha}(\omega) = \frac{\tilde{\zeta}(\omega)}{1-i\omega}. \tag{8.26}$$

The expectation value of the noise spectrum, $\left\langle|\tilde{\alpha}(\omega)|^2\right\rangle$ in the long time limit, is:

$$\begin{aligned}\left\langle|\tilde{\alpha}(\omega)|^2\right\rangle &= \frac{1}{2\pi\left(1+\omega^2\right)}\int\int e^{-i\omega(t-t')}\left\langle\zeta(t)\zeta^*(t')\right\rangle dtdt'.\\ &= \frac{T}{2\pi\left(1+\omega^2\right)}.\end{aligned} \tag{8.27}$$

This equation can also be used for some classical problems, which correspond to the high-temperature limit of $\bar{n}\gg 1$.

8.4. Stochastic gauge expansion

In this approach, the density matrix is expanded as a weighted integral over coherent state projection operators:

$$\rho(t) = \int d\boldsymbol{\phi}P\left(t,\boldsymbol{\phi}\right)\Lambda\left(\boldsymbol{\phi}\right). \tag{8.28}$$

Here, in the stochastic gauge method [73], $\boldsymbol{\phi} \equiv [\Omega, \boldsymbol{\alpha}, \boldsymbol{\beta}]$, where $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are each M-dimensional complex numbers, and Ω is a real or complex weight. The operator basis Λ is defined using un-normalized coherent states $\|\boldsymbol{\alpha}\rangle = \exp\left(\boldsymbol{\alpha} \cdot \hat{\boldsymbol{a}}^{\dagger}\right)|0\rangle$, so that:

$$\Lambda(\boldsymbol{\phi}) = \Omega \|\boldsymbol{\alpha}\rangle \langle \boldsymbol{\beta}^{*}\| e^{-\boldsymbol{\alpha}\cdot\boldsymbol{\beta}}. \tag{8.29}$$

There are standard identities available, namely:

$$\begin{aligned}
\hat{a}_j\Lambda &= \alpha_j\Lambda \\
\hat{a}_j^{\dagger}\Lambda &= [\partial/\partial\alpha_j + \beta_j]\Lambda \\
\Lambda\hat{a}_j^{\dagger} &= \beta_j\Lambda \\
\Lambda\hat{a}_j &= [\partial/\partial\beta_j + \alpha_j]\Lambda \\
0 &= [\Omega\partial/\partial\Omega - 1]\Lambda \\
0 &= \partial^2/\partial\Omega^2\Lambda
\end{aligned} \tag{8.30}$$

The hermiticity of ρ means that every $\boldsymbol{\phi}$ has a conjugate $\boldsymbol{\phi}^{*}$ of equal weight, so the integral is sampled in pairs $\boldsymbol{\phi}_s$ and $\boldsymbol{\phi}_s^{*}$, corresponding to a sum over $\mathcal{S}$ samples of the real part of Λ:

$$\rho_c(t) = \lim_{\mathcal{S}\to\infty} \frac{1}{\mathcal{S}} \sum_s \Re\Lambda(\boldsymbol{\phi}_s(t)). \tag{8.31}$$

Operator averages are obtained through defining a *weighted* average as the infinite ensemble limit of a sum of trajectories:

$$\langle f(\boldsymbol{\phi})\rangle \equiv \lim_{\mathcal{S}\to\infty} \langle f(\boldsymbol{\phi})\rangle_{\mathcal{S}}. \tag{8.32}$$

Here, for hermitian operators,

$$\langle f(\boldsymbol{\phi})\rangle_{\mathcal{S}} \equiv \frac{1}{\mathcal{S}} \sum_s \Re\left[\Omega_s f(\boldsymbol{\phi}_s)\right], \tag{8.33}$$

with the approximation of taking only a finite number of samples $\mathcal{S}$. For example, the quantum average particle number $\langle \hat{n}_j\rangle_Q$ is obtained on taking a weighted average of $n_{js} \equiv \alpha_{js}\beta_{js}$:

$$\langle \hat{n}_j\rangle_Q = \langle n_j\rangle. \tag{8.34}$$

Individual trajectory photon numbers $\Re(\Omega_s n_{js})$ can be negative, although their large-$\mathcal{S}$ average is non-negative. These trajectories correspond to Schrodinger cat superpositions, causing mixtures of positive and negative 'effective' photon numbers. Such behavior is impossible in the diagonal Glauber-Sudarshan representation, where for a probabilistic distribution, only classical photon statistics occur [85, 86].

8.5. Input-output spectra

The spectrum of an internal field variable is not the one that is usually measured. An important application of stochastic equations is therefore in calculating output, measured spectra of lasers, quantum optics, opto-mechanics and quantum circuits [76, 87]. These have the feature that the measured output spectrum may also include noise from reflected fields at the input/output ports. If the quantum noise term in the Heisenberg equations for a cavity operator $\hat{a}_c$ is given by: $\dot{\hat{a}}_c \sim .. + \sqrt{2\gamma}\hat{a}_{in}(t)$, then the corresponding operator input-output relations are $\hat{a}_{out}(t) + \hat{a}_{in}(t) = \sqrt{2\gamma}\hat{a}_c$.

In quantum phase-space for the case of the harmonic oscillator or similar systems, $\alpha_{in} = \sqrt{\sigma + \bar{n}}\zeta$ is the noise term in the Langevin equation. The output fields α_{out} that are measured are given by:

$$\alpha_{out} = \sqrt{2\gamma}\alpha - \alpha_{in}. \tag{8.35}$$

Hence one must include in the spectrum both the internal mode variables and the noise terms themselves. Solving for the spectra, one obtains auxiliary fields with

$$\begin{aligned}\tilde{\alpha}_{in}(\omega) &= \sqrt{\sigma + \bar{n}}\tilde{\zeta}(\omega) \\ \tilde{\alpha}_{out}(\omega) &= \sqrt{2\gamma}\tilde{a}(\omega) - \sqrt{\sigma + \bar{n}}\tilde{\zeta}(\omega).\end{aligned} \tag{8.36}$$

In summary, it is the output fields that are amplified and measured. Hence one must be able to compute the spectra of the output fields for experimental comparisons. These have the additional feature that they include the reservoir noise $\tilde{\zeta}(\omega)$, evaluated at the same time as the field is evaluated, since the reservoir noise is the input here. In xSPDE these are called *auxfields*.

8.5.1. Steady-state result

Consider the example of the damped quantum harmonic oscillator in the Wigner representation case with $\gamma = 1$, $\sigma = 1/2$ and $\bar{n} = 0$. Over long time-scales, so that one is in the steady state, the solution for $\tilde{a}_{out}$ is that:

$$\begin{aligned}\tilde{\alpha}_{out}(\omega) &= \sqrt{2}\left[\frac{1}{1-i\omega} - \frac{1}{2}\right]\tilde{\zeta}(\omega) \\ &= \frac{1}{\sqrt{2}}\left[\frac{1+i\omega}{1-i\omega}\right]\tilde{\zeta}(\omega).\end{aligned} \tag{8.37}$$

This gives the following expectation values:

$$\begin{aligned}\left\langle\tilde{\alpha}_{out}(\omega)\left(\tilde{\alpha}_{out}(\omega)\left(\omega'\right)\right)^*\right\rangle &= \frac{1}{2}\delta\left(\omega-\omega'\right) \\ \left\langle\tilde{\alpha}_{in}(\omega)\left(\tilde{\alpha}_{in}(\omega)\left(\omega'\right)\right)^*\right\rangle &= \frac{1}{2}\delta\left(\omega-\omega'\right).\end{aligned} \tag{8.38}$$

These are the expectation values of the zero temperature quantum fluctuations in the input and output channels. This means that the harmonic oscillator in its ground state is in equilibrium with an external vacuum field reservoir, also in its ground state. However, the internal spectral correlations of the harmonic oscillator are modified by the coupling.

While this is a simple result, exactly the same general type of behavior occurs in more sophisticated cases. These may include many coupled modes with nonlinearities. Additional or auxiliary fields that depend both on noise terms and internal stochastic variables are required. The soluble case given above is a useful test case, and it is treated numerically later in the manual.

8.6. Linear photonic network theory

In some quantum systems, output observables are obtained not via dynamical processes, but after a simple linear transformation of a multi-mode density operator. In the case of linear photonic quantum computing networks such as GBS, the mode transformation is traditionally generated by a network of beam-splitters, phase shifters, and mirrors [88, 89], although other set-ups include fibre delay lines [90, 91].

These networks act as M-mode interferometers which interfere input photon, generating large amounts of entanglement due to the exponential number of interference pathways available to photons. In the ideal lossless regime, the network itself is defined by an $M \times M$ Haar random unitary matrix $\boldsymbol{U}$, such that output modes are linear combinations of each input mode:

$$\hat{a}_i^{(\text{out})} = \sum_{j=1}^{M} U_{ij} \hat{a}_j^{(\text{in})}, \tag{8.39}$$

where $\hat{a}_i^{(\text{in})}$ and $\hat{a}_j^{(\text{out})}$ are the input and output annihilation operators for modes i, j respectively.

Practically, photon loss in the network is commonplace, thus causing the matrix to be non-unitary. Therefore, lossy networks are denoted by the transmission matrix $\boldsymbol{T}$. These give a different transformation law, where:

$$\hat{a}_i^{(\text{out})} = \sum_{j=1}^{M} T_{ij} \hat{a}_j^{(\text{in})} + \sum_{j=1}^{M} B_{ij} \hat{b}_j^{(\text{in})}, \tag{8.40}$$

Here, the M operators $\hat{b}_i^{(\text{in})}$are noise operators which are necessary to conserve the operator commutation relations

$$\begin{aligned} [\hat{a}_i, \hat{a}_j] &= 0 \\ \left[\hat{a}_i, \hat{a}_j^\dagger\right] &= \delta_{ij}. \end{aligned} \tag{8.41}$$

The noise operators are independent, commuting operators, who comprise inputs from the reservoirs that cause losses, where the reservoirs are all in a vacuum state.

The inclusion of the loss matrix conserves the unitarity of the network. Substituting Eq.(8.40) into Eq.(8.41) and and taking expectation values for a vacuum state input gives

$$\begin{aligned}\delta_{ij} &= \left\langle \left[\hat{a}_i^{(\text{out})}, \hat{a}_j^{\dagger(\text{out})}\right] \right\rangle \\ &= \sum_k \left(T_{ik}T_{jk}^* + B_{ik}B_{jk}^*\right). \end{aligned} \tag{8.42}$$

Next, we can define a new $M \times M$ matrix

$$\boldsymbol{D} = \boldsymbol{B}\boldsymbol{B}^\dagger = \boldsymbol{I} - \boldsymbol{T}\boldsymbol{T}^\dagger. \tag{8.43}$$

This is hermitian, since $\boldsymbol{D}^\dagger = \boldsymbol{D}$, and so has a diagonal representation as $D = \tilde{U}\lambda^2\tilde{U}^\dagger$, for some unitary matrix $\tilde{U}$. We assume that the transmission matrix $\boldsymbol{T}$ is lossy, so that $\boldsymbol{D}$ is positive definite and λ is real, representing absorption rather than gain.

8.7. Quantum input states

The density operator $\hat{\rho}^{(\text{in})}$ transformed by linear network has N input modes, where one can have $N = M$ or $N \subset M$, in which case the remaining $M - N$ modes are vacuum inputs at unused ports. If each input mode is independent, $\hat{\rho}^{(\text{in})}$ is a product of input states.

Currently, xSPDE can only generate input squeezed states and thermal states as outlined below. Other inputs are possible, since the positive P-representation and Q-representation are complete, positive representations, and can be added through user customization.

For pure squeezed vacuum states, the input is defined as

$$\hat{\rho}^{(\text{in})} = \prod_{j=1}^{M} |r_j\rangle \langle r_j|, \tag{8.44}$$

where $\boldsymbol{r} = [r_1, \dots, r_M]$ is a vector of squeezing parameters r_j and

$$\begin{aligned}|r_j\rangle &= \hat{S}(r_j)|0\rangle \\ &= \exp\left(r_j \frac{(\hat{a}_j^{\dagger(\text{in})})^2}{2} - r_j \frac{(\hat{a}_j^{(\text{in})})^2}{2}\right)|0\rangle, \end{aligned} \tag{8.45}$$

is the squeezed vacuum state with squeezing operator $\hat{S}(r_j)$, which satisfies the unitarity condition $\hat{S}\hat{S}^\dagger = \hat{S}^\dagger\hat{S} = 1$. Here, we have assumed the squeezed state phase is zero.

8.7.1. Pure squeezed states

Pure squeezed states are the default type of squeezed state generated in xSPDE. Given each input mode is independent, one can use the well known single-mode squeezed state theory [92, 93], to derive the basic properties of pure squeezed states.

Using the relations

$$\begin{aligned}\hat{S}^{\dagger}(r_j)\hat{a}_j^{(\text{in})}\hat{S}(r_j) &= \hat{a}_j^{(\text{in})}\cosh(r_j) - \hat{a}_j^{\dagger(\text{in})}\sinh(r_j)\\ \hat{S}^{\dagger}(r_j)\hat{a}_j^{\dagger(\text{in})}\hat{S}(r_j) &= \hat{a}_j^{\dagger(\text{in})}\cosh(r_j) - \hat{a}_j^{(\text{in})}\sinh(r_j),\end{aligned} \tag{8.46}$$

the mean input photon number per mode is defined as

$$\begin{aligned}\bar{n}_j &= \left\langle \hat{a}_j^{\dagger(\text{in})}\hat{a}_j^{(\text{in})}\right\rangle\\ &= \langle 0|\,\hat{S}^{\dagger}(r_j)\hat{a}_j^{\dagger(\text{in})}\hat{S}(r_j)\hat{S}^{\dagger}(r_j)\hat{a}_j^{(\text{in})}\hat{S}(r_j)\,|0\rangle\\ &= \sinh^2(r_j),\end{aligned} \tag{8.47}$$

while the mean input coherence per mode is

$$\begin{aligned}m_j &= \left\langle \left(\hat{a}_j^{(\text{in})}\right)^2\right\rangle\\ &= \langle 0|\,\hat{S}^{\dagger}(r_j)\hat{a}_j^{(\text{in})}\hat{S}(r_j)\hat{S}^{\dagger}(r_j)\hat{a}_j^{(\text{in})}\hat{S}(r_j)\,|0\rangle\\ &= \sinh(r_j)\cosh(r_j).\end{aligned} \tag{8.48}$$

For pure squeezed states, the coherence and photon number are related via $m_j^2 - \bar{n}_j = \bar{n}_j^2$.

Squeezed states are minimum uncertainty states and are therefore defined entirely by their quadrature variances. From the quadrature operators

$$\begin{aligned}\hat{x}_j^{(\text{in})} &= \hat{a}_j^{(\text{in})} + \hat{a}_j^{\dagger(\text{in})}\\ \hat{y}_j^{(\text{in})} &= -i\left(\hat{a}_j^{(\text{in})} - \hat{a}_j^{\dagger(\text{in})}\right),\end{aligned} \tag{8.49}$$

which obey the commutation relation $\left[\hat{x}_j^{(\text{in})}, \hat{y}_k^{(\text{in})}\right] = 2i\delta_{jk}$, the normally ordered $x_j^{(\text{in})}$-quadrature variance is defined as

$$\begin{aligned}\left\langle :(\Delta\hat{x}_j^{(\text{in})})^2:\right\rangle &= \left\langle (\hat{x}_j^{(\text{in})})^2\right\rangle\\ &= 2(\bar{n}_j + m_j)\\ &= e^{2r_j} - 1,\end{aligned} \tag{8.50}$$

while the normally ordered $y_j^{(\text{in})}$-quadrature variance is

$$\begin{aligned}\left\langle : (\Delta \hat{y}_j^{(\mathrm{in})})^2 : \right\rangle &= \left\langle (\hat{y}_j^{(\mathrm{in})})^2 \right\rangle \\ &= 2(\bar{n}_j - m_j) \\ &= e^{-2r_j} - 1. \end{aligned} \tag{8.51}$$

8.7.2. Thermal squeezed states

As stated above, xSPDE4 can currently simulate pure and thermalized squeezed states, as well as classical thermal state inputs into a photonic network. This is achieved using a model for thermal squeezed states which alters the multi-mode input coherence as $\tilde{m}_j = (1-\epsilon)m_j$, with ϵ being the thermalization component which is input to xSPDE as *p.thermal*, whilst keeping the input photon number unchanged. This allows one to interpolate between pure thermal, $\epsilon = 1$, and pure squeezed, $\epsilon = 0$, states.

Thermal states are classical states with fluctuations larger than the vacuum limit such that their quadrature variances are $\left\langle : (\Delta \hat{x}_j^{(\mathrm{in})})^2 : \right\rangle = \left\langle : (\Delta \hat{y}_j^{(\mathrm{in})})^2 : \right\rangle > 1$. In terms of Fock states, the thermal state density operator for the j-th mode is the single-mode state

$$\hat{\rho}_j^{(\mathrm{in})} = \frac{1}{1+\bar{n}_j} \sum_{n_j=0}^{\infty} \left(\frac{\bar{n}_j}{1+\bar{n}_j} \right)^{n_j} |n_j\rangle \langle n_j| , \tag{8.52}$$

which gives the well known single-mode photon number distribution

$$P(n_j) = \frac{\bar{n}_j^{n_j}}{(\bar{n}_j+1)^{n_j+1}}. \tag{8.53}$$

Thermal states can be used to generate thermal squeezed states with initial occupation n_j^{th}, which gives [94]:

$$\begin{aligned} \bar{n}_j &= n_j^{\mathrm{th}} + \left(2n_j^{\mathrm{th}}+1\right)\sinh^2(r_j) \\ \tilde{m}_j &= \left(2n_j^{\mathrm{th}}+1\right)\sinh(r_j)\cosh(r_j). \end{aligned} \tag{8.54}$$

In the thermalized case, the relationship between coherence and photon number is modified, since to eliminate r_j one must use the relationship that

$$\begin{aligned} \frac{\tilde{m}_j^2}{\left(2n_j^{\mathrm{th}}+1\right)^2} &= \sinh^2(r_j)\left(1+\sinh^2(r_j)\right) \\ &= \frac{\bar{n}_j - n_j^{\mathrm{th}}}{\left(2n_j^{\mathrm{th}}+1\right)} \left(1 + \frac{\bar{n}_j - n_j^{\mathrm{th}}}{\left(2n_j^{\mathrm{th}}+1\right)} \right). \end{aligned} \tag{8.55}$$

Therefore:

$$\begin{aligned}\tilde{m}_j^2 &= \left(\bar{n}_j - n_j^{\text{th}}\right)\left(1 + \bar{n}_j + n_j^{\text{th}}\right) \\ &= \bar{n}_j + \bar{n}_j^2 - \left((n_j^{\text{th}})^2 + n_j^{\text{th}}\right).\end{aligned} \tag{8.56}$$

Using the above theory, thermalized squeezed states can also be used as a test for numerical simulations of photon counting observables, as one can define the threshold detector projection operators in terms of the photon number and coherence as explained below.

8.8. Photon counting

Although linear photonic networks are conceptually very simple, when employed as quantum computers they generate samples from an output distribution which corresponds to the #P-hard matrix permanent, Hafnian or Torontonian functions. Which type of matrix function is evalutated depends on the input states to the network, with Fock states corresponding to the permanent and squeezed states corresponding to either the Hafnian or Torontonian functions, where the difference between these distributions comes from the type of detector used.

Currently in xSPDE4, only Gaussian states are generated natively, hence we restrict our discussion here to Gaussian state photonic quantum computing networks such as GBS. When photon-number resolving (PNR) detectors are used, photon count patterns, which are our samples of the output distribution, correspond ot the Hafnian function, while the Torontonian requires threshold detectors that "click" for a photon detection event.

From standard photon counting theory, the projection operator for observing $c_j = 0, 1, 2, \dots, c_j^{(\max)}$ counts is denoted by [93]

$$\hat{p}_j(c_j) = \frac{1}{c_j!} : (\hat{n}'_j)^{c_j} e^{-\hat{n}'_j} :, \tag{8.57}$$

where $: \cdots :$ denotes normal ordering and $\hat{n}'_j = a_j^{\dagger(\text{out})} a_j^{(\text{out})}$ is the output photon number and $c_j^{(\max)}$ is the maximum observable count.

For PNR detectors, which can discriminate between photon numbers, each detector is defined by the above projector, with $c_j^{(\max)}$ varying depending on experimental implementation. Output photon count patterns are denoted by the count vector $\boldsymbol{c} = [c_1, c_2 \dots, c_M]$, and the projection operator for a specific output pattern given as

$$\hat{P}(\boldsymbol{c}) = \bigotimes_{j=i}^{M} \hat{p}_j(c_j). \tag{8.58}$$

The expectation value of this pattern projector corresponds to the Hafnian [95]

$$\left\langle \hat{P}(\boldsymbol{c})\right\rangle = \frac{1}{\sqrt{\det(\boldsymbol{Q})}}\frac{\left|\text{Haf}(\boldsymbol{B}_S)\right|^2}{\prod_{j=1} c_j!}, \tag{8.59}$$

which is $\#P$-hard to compute at large M. Here, $\boldsymbol{B}_S$ is the sub-matrix of $\boldsymbol{B} = \boldsymbol{U}\left(\bigoplus_{j=1}^{M}\tanh(r_j)\right)\boldsymbol{U}^T$ formed from modes with detected counts and $\boldsymbol{Q}$ is a $2M \times 2M$ covariance matrix.

Threshold detectors saturate for more than one count at a detector. Therefore, outputs are binary with $c_j = c_j^{(\text{max})} = 1$ denoting a detection event, or click, even if multiple photons hit the same detector, and $c_j = 0$ is no detection event. From Eq.(8.57), the click projection operator is obtained by summing over all $c_j > 0$ counts such that

$$\begin{aligned}\hat{\pi}(1) &=: \sum_{c_j>0}\frac{(\hat{n}'_j)^{c_j}}{c_j!}e^{-\hat{n}'_j} : \\ &= 1 - e^{-\hat{n}'_j},\end{aligned} \tag{8.60}$$

which gives the standard threshold detector projection operator

$$\hat{\pi}_j(c_j) =: e^{-\hat{n}'_j}\left(e^{\hat{n}'_j}-1\right)^{c_j} : . \tag{8.61}$$

The projection operator for a count pattern output is then similarly defined as

$$\hat{\Pi}(\boldsymbol{c}) = \bigotimes_{j=i}^{M}\hat{\pi}_j(c_j), \tag{8.62}$$

where the expectation value corresponds to the Torontonian function [96]

$$\left\langle \hat{\Pi}(\boldsymbol{c})\right\rangle = \frac{\text{Tor}\left(\boldsymbol{O}_S\right)}{\sqrt{\det\left(\boldsymbol{\Sigma}\right)}}, \tag{8.63}$$

where $\boldsymbol{O}_S$ is the sub-matrix of $\boldsymbol{O} = \boldsymbol{I} - \boldsymbol{\Sigma}^{-1}$ with covariance matrix $\boldsymbol{\Sigma}$.

8.8.1. Exact output examples: Threshold detectors

For photon counting probabilities obtained through threshold detection, a variety of probabilities can be computed exactly.

Initially, we are interested in computing the probability of detecting no photons at the threshold detector output, i.e. $\langle \hat{\pi}\left(0\right)\rangle$. From Marian [94], the $n = 0$ single-mode photon number probability is known exactly from the full distribution of a thermalized squeezed state, $P_{\text{sqth}}(n)$, such that

$$P_{\text{sqth}}(0) = \langle \hat{\pi}\left(0\right)\rangle = \frac{1}{n^{\text{th}}+1}\left(1 + \frac{2n^{\text{th}}+1}{\left(n^{\text{th}}+1\right)^2}\sinh^2\left(r\right)\right)^{-(1/2)}$$

Substituting the thermal squeezed state modified photon number Eq.(8.54) the vacuum state detection event probability can be derived as

$$\begin{aligned}\langle\hat{\pi}\left(0\right)\rangle &= \left(\bar{n}-n_{th}+\left(n_{th}+1\right)^{2}\right)^{-(1/2)}\\ &= \left(1+\bar{n}+n_{th}+n_{th}^{2}\right)^{-(1/2)}\\ &= \left(\left(1+\bar{n}\right)^{2}-\tilde{m}^{2}\right)^{-(1/2)}.\end{aligned}$$

Hence, the click and no-click probabilities for thermalized squezed states can be computed exactly in the limit of an identity transmission matrix or in the case of a fully thermal state ($\epsilon = 1$) input with Haar random unitary matrix as:

$$\begin{aligned}\langle\hat{\pi}\left(0\right)\rangle &= \frac{1}{\sqrt{\left(1+\bar{n}\right)^{2}-\tilde{m}^{2}}}\\ \langle\hat{\pi}\left(1\right)\rangle &= 1-\langle\hat{\pi}\left(0\right)\rangle = 1-\frac{1}{\sqrt{\left(1+\bar{n}\right)^{2}-\tilde{m}^{2}}}.\end{aligned} \tag{8.64}$$

8.8.2. Exact output examples: PNR detectors

For PNR detectors, the multi-mode photon counting distributions are known exactly for both pure squeezed states and thermal states. These exact distributions are only valid if one assumes each input state has equal squeezing parameters $r = r_1 = \cdots = r_N$, which in turn causes the input mean photon number to be equal for each mode $\bar{n} = \bar{n}_1 = \cdots = \bar{n}_N$.

For thermal states, the single-mode photon counting distribution Eq.(8.53) is a geometric distribution, which becomes clear by defining the success probability as

$$p = \frac{1}{1+\bar{n}}, \tag{8.65}$$

such that

$$P(n) = p(1-p)^{n}. \tag{8.66}$$

This single-mode theory can be extended for multiple thermal state inputs as the sum of a geometrically distributed random variable X with success probability p is the random variable

$$Y = \sum_{j}^{M} X_j \tag{8.67}$$

which is negative binomially distributed with M and p. The total photon number distribution is then defined as

$$P(Y=m) = \begin{pmatrix} m+M-1 \\ m \end{pmatrix} p^{M}(1-p)^{m}, \tag{8.68}$$

where $m = 0, 1, 2, \ldots$ is the total, or binned, photon number. At large M, the distribution is a Gaussian with mean and variance

$$\begin{aligned}
\mu_m &= \bar{n}M, \\
\sigma_m^2 &= \frac{\mu_m}{p} = \bar{n}M\left(\bar{n}+1\right).
\end{aligned} \tag{8.69}$$

For pure squeezed states transformed by a lossless unitary matrix, the probability of observing $m = 0, 1, 2, \ldots$ total photon counts from M modes is [97,98]:

$$\begin{aligned}
P(2m) &= \binom{\frac{M}{2}+m-1}{m}\text{sech}^M(r)\tanh^{2m}(r) \\
P(2m+1) &= 0.
\end{aligned} \tag{8.70}$$

This distribution is well known due to the distinct oscillations between even and odd photon count bins. Such oscillations arises due to the generation of squeezed photons in highly correlated pairs in a parametric down-conversion process. Therefore, only even numbers of photons are ever generated, as in clear from the squeezing operator in Eq.(8.45).

In the limit $M \to \infty$, the total photon counting distribution $P(m)$ reduces to a Poisson distribution for the even counts [97]

$$\begin{aligned}
P(2m) &= \frac{1}{m!}e^{-Mn/2}\left(\frac{Mn}{2}\right)^m \\
P(2m+1) &= 0,
\end{aligned}$$

although the full distribution is considered super-Poissonian, i.e. the variance is now larger than the mean. The lossless distribution can also be written in terms of success probabilities as

$$\begin{aligned}
P(2m) &= \binom{\frac{M}{2}+m-1}{m}p^{M/2}(1-p)^m \\
P(2m+1) &= 0,
\end{aligned} \tag{8.71}$$

where p is defined in Eq.(8.65).

For lossless linear networks, the total photon counting distribution is also known exactly and is defined as [90]

$$P(2m) = t^{4m}\binom{\frac{M}{2}+m-1}{m}p^{M/2}(1-p)^m {}_2F_1\left(m+\frac{1}{2},\frac{M}{2}+m;\frac{1}{2};(1-t^2)^2(1-p)\right)$$

$$P(2m+1) = (1-t^2)\left(\frac{m+1}{2}\right)t^{m-1}\binom{\frac{M}{2}+\frac{m+1}{4}-1}{\frac{m+1}{4}}p^{M/2}(1-p)^{(m+1)/4}$$

$$\times\, {}_2F_1\left(\frac{m+3}{4},\frac{1}{4}(2M+m+1);\frac{3}{2};(1-t^2)^2(1-p)\right), \tag{8.72}$$

where t is a uniform amplitude loss coffieicent, which is applied as $t\boldsymbol{U}$ and is related to intensity loss via $t=\sqrt{\eta}$, and ${}_2F_1(a,b;c;z)$ is the Gauss hypergeometric function. The lossy distribution converges to the lossless distribution Eq.(8.71) when $t=1$ as ${}_2F_1(a,b;c;0)=1$.

When identical thermal or pure squeezed states are input into a optical linear, one can use the above exact distributions to compare phase-space simulated grouped count probabilities, which are explained below.

8.9. Intensity correlations

We now explain two types of measurable correlations: Glauber intensity correlations [70] and grouped correlations [99, 100], also referred to as a grouped count probabilities (GCPs).

Intensity correlation simulations can only be performed on photon number operator observables. Therefore, although they are valid for determining photon number probabilities in click experiments, they correspond directly to PNR detector outputs. Meanwhile GCPs are valid for both threshold and PNR detectors.

Glauber's n-th order intensity correlation is defined as [70]

$$G^{(n)}(c_j) = \left\langle :(\hat{n}'_j)^{c_j}\dots(\hat{n}'_M)^{c_M}:\right\rangle, \tag{8.73}$$

where $n=\sum c_j$ is the correlation order. Multi-mode Glauber correlations determine the probability of detecting n photons at M modes.

The normal ordering requirement causes all creation operators to the right and all annihilation operators to the left. For example, the second-order correlation

$$G^{(2)} = \left\langle a_1^{\dagger(\mathrm{out})}a_2^{\dagger(\mathrm{out})}a_2^{(\mathrm{out})}a_1^{(\mathrm{out})}\right\rangle, \tag{8.74}$$

corresponds to detecting one photon at $M=1$, and one at $M=2$.

Upon reordering, one obtains

$$G^{(2)} = \left\langle a_1^{\dagger(\mathrm{out})}a_2^{(\mathrm{out})}\right\rangle\left\langle a_2^{\dagger(\mathrm{out})}a_1^{(\mathrm{out})}\right\rangle + \left\langle a_1^{\dagger(\mathrm{out})}a_1^{(\mathrm{out})}\right\rangle\left\langle a_2^{\dagger(\mathrm{out})}a_2^{(\mathrm{out})}\right\rangle. \tag{8.75}$$

The first term describes non-local correlations, which is the interference of photons between modes (or practically, detectors), while the second term describes the photon intensity at each mode (or detector), which are termed local correlations.

If the mean number of photons is small, such that a detector will only ever observe one photon, the intensity correlation becomes a coincidence count

$$P_N = \left\langle \prod_j \hat{n}'_j \right\rangle , \tag{8.76}$$

as we assume photons do not interfere at detectors, removing non-local correlations.

8.10. Grouped correlations

Grouped count probabilities (GCPs) are another observable correlation implemented in the Quantum phase-space toolbox in xSPDE4.

For threshold detectors, GCPs are defined as [99]

$$\mathcal{G}^{(n)}_{\boldsymbol{S}}(\boldsymbol{m}) = \left\langle \prod_{j=1}^{d} \left[\sum_{\sum c_i = m_j} \hat{\Pi}_{S_j}(\boldsymbol{c}) \right] \right\rangle , \tag{8.77}$$

while for PNR detectors, GCPs are similiarly defined as [100]

$$\mathcal{G}^{(n)}_{\boldsymbol{S}}(\boldsymbol{m}) = \left\langle \prod_{j=1}^{d} \left[\sum_{\sum c_i = m_j} \hat{P}_{S_j}(\boldsymbol{c}) \right] \right\rangle . \tag{8.78}$$

For both detector type, $\boldsymbol{m} = (m_1, \dots, m_d)$ is the observed d-dimensional grouped count and $\boldsymbol{S} = (S_1, S_2, \dots)$ is a vector of disjoint subsets of $\boldsymbol{M} = (M_1, M_2, \dots)$ modes. Each grouped count is obtained by summing over binary patterns $m_j = \sum_i^M c_i$. Therefore, grouped counts contain k bins, with each bin corresponding to the total number of clicks in each pattern. In one-dimension, GCPs are the probability of observing m counts in any pattern with $n = M$ and $S = \{1, \dots, M\}$. This observable is called total counts.

Descriptions on how these GCPs are simulated in phase-space are given in Section 7.5.

8.10.1. Multi-dimensional binning

For larger dimensions, each grouped count sums over detector outputs for a subset of modes only such that $m_j = \sum_i^{M/d} c_i$. The modes in each subset are denoted in the vector $\boldsymbol{S}$. For example, in two-dimensions one has subsets $\boldsymbol{S} = (S_1, S_2)$ which contain modes

$$\begin{aligned} S_1 &= \left\{ 1, \dots, \frac{M}{2} \right\} \\ S_2 &= \left\{ \frac{M+2}{2}, \dots, M \right\} . \end{aligned} \tag{8.79}$$

The output GCP is then a joint probability of observing $m_1 = \sum_{i=1}^{M/2} c_i$ and $m_2 = \sum_{i=M/2+1}^{M} c_i$ grouped counts with $k = (M/2+1)^2$ total bins.

The implied segregation of output modes in the two-dimensional example above is that S_1 will always contain the first M/d modes, S_2 the next $M/d+1 \to 2M/d$ modes, and so on for larger dimensions. However, there is no practical restriction on the output modes each subset can contain.

Therefore, by randomly permuting each binary pattern we can change the output modes that are contained in each subset giving

$$\frac{\binom{M}{M/d}}{d} = \frac{M!}{d(M/d)!(M-M/d)!}, \tag{8.80}$$

possible ways of generating $m_1, \dots, m_d$ grouped counts without repeating a specific permutation.

For example, when $M = 4$ and $d = 2$, including the standard division, there are 3 different orderings of outputs modes with subsets

$$\begin{aligned}
\boldsymbol{S} &= (S_1, S_2) = (\{1,2\},\{3,4\}), \\
\boldsymbol{S} &= (S_1, S_2) = (\{1,3\},\{2,4\}), \\
\boldsymbol{S} &= (S_1, S_2) = (\{1,4\},\{2,3\}).
\end{aligned} \tag{8.81}$$

Each permutation generates a different correlation, where we assume the commutation of GCP probabilities with subsets $(\{1,3\},\{2,4\}) = (\{2,4\},\{1,3\})$.

This permutation only changes the multidimensional GCP simulations, as in the total count case all modes are contained in the same subset $S = \{1, \dots, M\}$. This is also the case when simulating marginal probabilities, which are obtained by setting $n < M$ such that $M - n$ inputs are ignored.

9. Phase-space examples

These examples show two ways to use xSPDE in quantum phase-space. One way is to define all the noises and ordering methods in the input files. Another way is to use the p.phase parameter, in which case the default initialization will set up a Gaussian state network input according to the parameters described in this chapter. Either method works equivalently, depending on the preference of the user.

Note that choosing *p.dimension=0* corresponds to a network initialization, followed by a quantum measurement. For this case the output axes which correspond to different counting dimensions are treated as space dimensions. This allows better graphics for the outputs of large networks, where it is not practical to treat each different type of output as a separate line on a single graph.

9.1. Saturated laser noise

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

$$\dot{a} = \left(1 - |a|^2\right) a + bw(t) \tag{9.1}$$

To learn how to use the function inputs, try the following:

- **Solve for the saturated laser case**

You should get the output graph in Fig (9.1).

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

9.2. Nonlinear quantum simulation

This example involves a full nonlinear quantum phase-space simulation using the positive-P representation described in Sec (8), in which the two variables are only conjugate in the mean. This allows quantum superpositions of coherent states to be represented, or in fact any state, including squeezed or entangled states in more general cases.

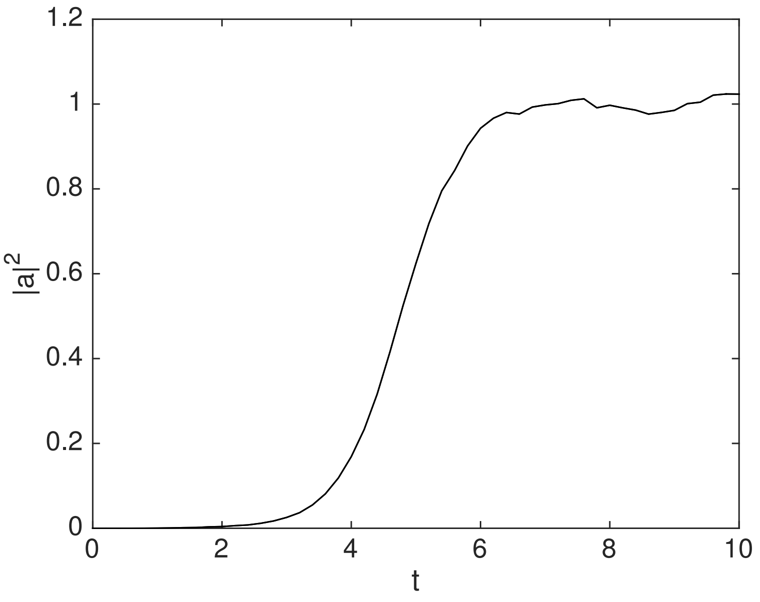


Figure 9.1.: *Simulation of the stochastic equation describing a laser turning on.*

A simple example is the nonlinear driven quantum subharmonic generator - for example, an opto-mechanical, superconducting or nonlinear optical medium in a driven cavity [101–104]. This is derived from the Hamiltonian for a resonant, coupled two-mode nonlinear interferometer, with $\hat{a}_2$ driven externally at twice the frequency of $\hat{a}_1$:

$$\hat{H} = i\hbar\left[\frac{\kappa}{2}\hat{a}_2\hat{a}_1^{\dagger 2} + \mathcal{E}_2\hat{a}_2^\dagger - h.c.\right] \tag{9.2}$$

After including losses in both modes in the positive P-representation, assuming zero temperature reservoirs, and adiabatically eliminating α_2 with $\gamma_2 \gg \gamma_1$, one has the following Ito equation:

$$\begin{aligned}\frac{d\alpha_1}{dt} &= -\gamma_1\alpha_1 + \alpha_1^\dagger\frac{\kappa\epsilon_2}{\gamma_2}\left[1 - \frac{\kappa}{2\epsilon_2}\alpha_1^2\right] + \sqrt{\frac{\kappa\epsilon_2}{\gamma_2} - \frac{\kappa^2}{2\gamma_2}\alpha_1^2}w_1(t)\\ \frac{d\alpha_1^\dagger}{dt} &= -\gamma_1\alpha_1^\dagger + \alpha_1\frac{\kappa\epsilon_2}{\gamma_2}\left[1 - \frac{\kappa}{2\epsilon_2}\alpha_1^{\dagger 2}\right] + \sqrt{\frac{\kappa\epsilon_2}{\gamma_2} - \frac{\kappa^2}{2\gamma_2}\alpha_1^{\dagger 2}}w_1(t)\end{aligned} \tag{9.3}$$

Rescaling the fields so that $\alpha_1 = a_1\sqrt{n_c}$, $\alpha_1^\dagger = a_2\sqrt{n_c}$, where $n_c = \frac{2\epsilon_2}{\kappa}$, then rescaling time by letting $\tau = \frac{\kappa\epsilon_2}{\gamma_2}t$, defining $c = \frac{\gamma_1\gamma_2}{\kappa\epsilon_2}$, and using Eq (2.11) to transform from an

Ito to a Stratonovich equation gives:

$$\begin{aligned}
\frac{da_1}{d\tau} &= -(c-\frac{1}{2n_c})a_1 + a_2\left[1-a_1^2\right] + \frac{1}{\sqrt{n_c}}\sqrt{1-a_1^2}w_1(\tau) \\
\frac{da_2}{d\tau} &= -(c-\frac{1}{2n_c})a_2 + a_1\left[1-a_2^2\right] + \frac{1}{\sqrt{n_c}}\sqrt{1-a_2^2}w_2(\tau)\ ,
\end{aligned} \tag{9.4}$$

where w_1, w_2 are delta-correlated real Gaussian noises.

There is a bistable region, which leads to a discrete time symmetry breaking. The solution in the steady-state is

$$P = \left(1-a_1^2\right)^{cn_c-1}\left(1-a_2^2\right)^{cn_c-1} e^{2n_c a_1 a_2} \tag{9.5}$$

The integration manifold is the region of real a_1, a_2, such that $a_1^2 \le 1$, $a_2^2 \le 1$. There are two physically possible metastable values of the amplitudes. The physically observed quantity is the amplitude and number:

$$\begin{aligned}
\langle \hat{a} \rangle &= \langle a_1 + a_2 \rangle \sqrt{\frac{n_c}{2}} \\
\langle \hat{n} \rangle &= n_c \langle a_1 a_2 \rangle\ .
\end{aligned} \tag{9.6}$$

Parameters that show bistable behavior on reasonable time-scales of $T = 100$ are $c = 0.6$, $n_c = 4$. To learn more, try the following:

- **Simulate the nonlinear oscillator by creating a file, say,** $NonlinearQ.m$
- **Can you observe quantum tunneling in the bistable regime?**
- **Do you see transient Schrödinger 'cat states' with a negative** $n = \alpha_1\alpha_2$ **value?**

A negative value of $\alpha_1\alpha_2$ is evidence for a quantum superposition! For experimental comparisons, one would measure correlation functions and spectra. These calculations require long time scales, `p.ranges`, to observe tunneling, and of order 100 time steps per plotted time point, `p.steps`, to maintain good accuracy in the quantum simulations.

For lower damping and large nonlinearity, other methods should be used, as the stochastic equations can become unstable in this limit.

The model is a simplified version of more recent quantum technologies used to investigate Schrödinger cat formation in superconducting quantum circuits [105], and the CIM machine used to solve NP-hard optimization problems with photonic circuits [106–108], although there are greater complexities in both these cases.

Similar methods can also be used to investigate quantum and chemical non-equilibrium phase transitions [109], tunneling in open systems [110], quantum entanglement [111], Einstein-Podolsky-Rosen paradoxes [112,113], Bell violations [114,115], and many other problems treated in the literature [2,76].

9.3. Quantum linear oscillator

This solves an SDE for a damped quantum harmonic oscillator in the (truncated) Wigner phase-space calculus. It is initialized as a vacuum state, corresponding to a complex Gaussian initial condition having $\left\langle |a\,(0)|^2 \right\rangle = 1$. It is subject to vacuum noise, here realized by the auxiliary field a_{in}. An output field is given through the input-output relations and is realized by the auxiliary field a_{out}.

$$\begin{aligned}
\frac{\partial a}{\partial t} &= -a + \sqrt{2} a_{in}. \\
a_{in} &= \frac{1}{2}\left(w_1(t) + i w_2(t)\,\right) \\
a_{out} &= \sqrt{2} a - a_{in}
\end{aligned} \tag{9.7}$$

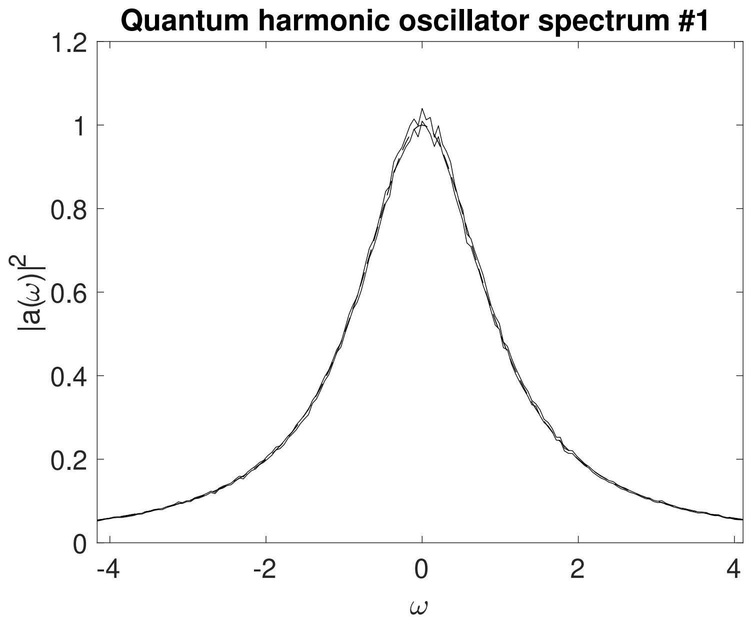
The computed spectral variances are compared with exact solutions and graphed, where:

$$\begin{aligned}
\frac{2\pi}{T}\left\langle |a\,(\omega)|^2 \right\rangle &= \frac{1}{(1+\omega^2)}. \\
\left\langle |a_{in}\,(\omega)|^2 \right\rangle &= \frac{1}{2} \\
\left\langle |a_{out}\,(\omega)|^2 \right\rangle &= \frac{1}{2}.
\end{aligned} \tag{9.8}$$

Notes

- Demonstrates how to include defined fields
- There are 4 steps per point, to give better accuracy due to finite steps
- The observe functions are all transformed, and include defined fields.

```
function e = Quantum()
p.name = 'Quantum harmonic oscillator spectrum';
p.points = 160;
p.steps = 4;
p.ranges = 120;
p.fields = 1;
p.auxfields = 2;
p.noises = 2;
p.ensembles = [400,1,12];
p.initial = @(w,~) (w(1,:)+1i*w(2,:))/(2);
p.a1 = @(w) (w(1,:)+1i*w(2,:))/2;
p.deriv = @(a,w,~) -a(1,:)+sqrt(2)*p.a1(w);
p.define = @(a,w,p) [p.a1(w);sqrt(2)*a(1,:)-p.a1(w)];
T = @(p) p.ranges(1);
p.observe{1} = @(a,x,p) (2.*pi/T(p))*a(1,:).*conj(a(1,:));
p.observe{2} = @(a,x,p) (2.*pi/T(p))*x(1,:).*conj(x(1,:));
p.observe{3} = @(a,x,p) (2.*pi/T(p))*x(2,:).*conj(x(2,:));
p.transforms = {1,1,1};
p.olabels{1} = '|a(\omega)|^2';
p.olabels{2} = '|a_{in}(\omega)|^2';
p.olabels{3} = '|a_{out}(\omega)|^2';
p.compare{1} = @(p) 1./(1+p.w.^2);
p.compare{2} = @(p) 0.5;
p.compare{3} = @(p) 0.5;
e = xspde(p);
end
```



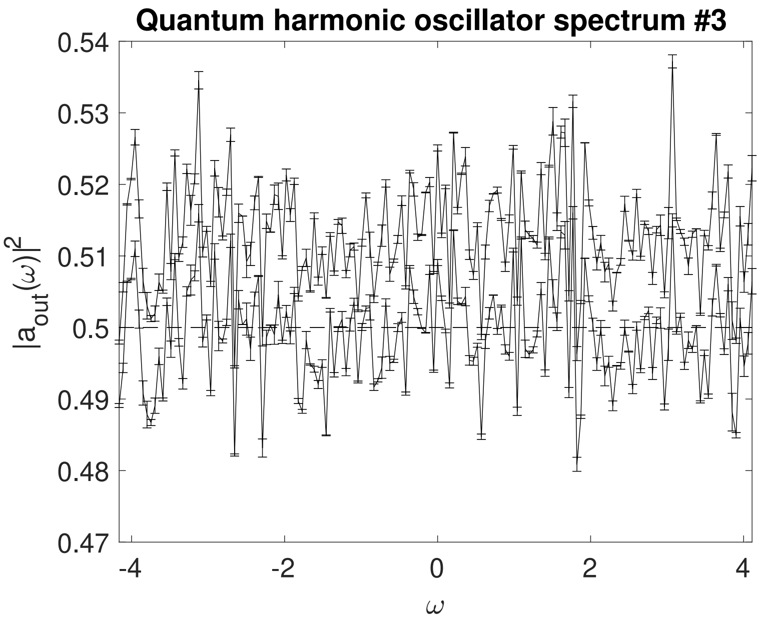


Figure 9.2.: *Top figure: Spectral density of the quantum state. Bottom figure: Spectral density of the output field. The solid lines indicate upper and lower sampling error bounds* ($\pm\sigma$)*, from sampling the stochastic equations. The dashed lines are exact results, the error-bars indicate step-size errors. Error bars are less than the minimum size for display in the top figure.*

9.4. Quantum network

This solves for a quantum network in the positive-P phase-space calculus. It is initialized with a thermalized multi-mode squeezed state together with a coherent component, giving Gaussian initial conditions. An output field is obtained after an identity network transformation for testing purposes.

The output is the mean per-channel photon count, compared to an exact prediction.

```
function e1 = phase_alphaGBS( )
p.dimensions = 0;
p.phase = 1; %+P phase-space
p.modes = 50; %matrix size m
p.name = sprintf('+P coherent, M=%d',p.modes);
p.tr = .5*ones(1,p.modes); %transmission
I = ones(1,p.modes/5); %identity vector
p.sqz = [I/2,I,1.5*I,2*I,0*I]; %nonuniform squeezing
p.alpha = [I/4,2*I,4*I,I/2,I]; %nonuniform coherence
p.thermal = 0.5*ones(1,p.modes); %thermal decoherence
p.ensembles = [1000,10,1]; %ensmbles for averaging
p.observe = @pn;
p.compare = @nc;
p.glabels = {{' ','Mode j'}};
p.olabels = {'<n>'};
p.diffplot = {1};
e1 = xspde(p);
end
```

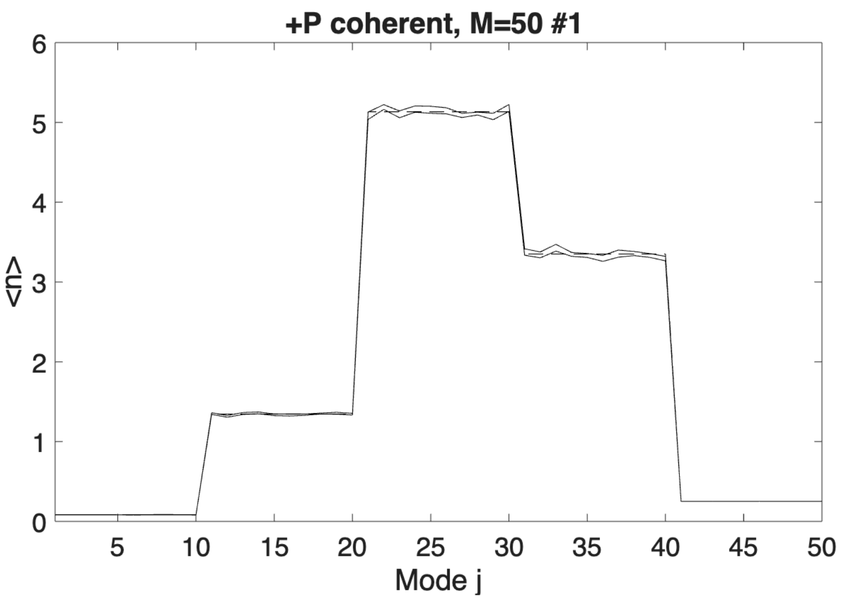


Figure 9.3.: *Simulation of the Gaussian boson sampling channel count for a thermalized, squeezed input with a coherent part, compared to an exact solution.*

Part V.

Quantum Monte-Carlo

10. Quantum stochastic toolbox

This chapter describes how to use the xSPDE numerical toolbox to solve quantum dynamical problems in Hilbert space, including unitary evolution and reservoirs. For theoretical background, see Chapter (11). For extended examples, see (12).

10.1. Wave-functions and density matrices

The basic master equation treated here has the Markovian form:

$$\dot{\rho} = -i\left[\hat{H}, \rho\right] + \sum_j \gamma_j \left(2\hat{L}_j \rho \hat{L}_j^\dagger - \hat{L}_j^\dagger \hat{L}_j \rho - \rho \hat{L}_j^\dagger \hat{L}_j\right), \tag{10.1}$$

where: $j = [j_1, j_2(, j_3)]$. Here j_1 is for the type of damping operator, j_2 is a mode index, and j_3 is an optional second mode index.

The quantum toolbox in xSPDE has three methods for representing open quantum systems, which allow the treatment of Hilbert spaces of increasing dimensionality:

1. Stochastic wave-functions with functional operators: p.quantum = 1, p.sparse = 0.

2. Density matrices with sparse operators: p.quantum = 2, p.sparse = 1.

3. Stochastic wave-functions with sparse operators: p.quantum = 1, p.sparse = 1.

There is a speed/memory tradeoff here. The higher numbered methods are typically faster, but use more memory. In the first two cases, one can use either a method using functions for operators, which is the default option, or else a sparse matrix method, which requires the operators to be stored in memory. The wave-function equations describe decoherence through stochastic methods, so each of these two approaches can treat coupling to reservoirs, up to the limits of time and memory constraints.

Less memory is used if the operator action on the wave-function is calculated when needed. This is a function call strategy. It is currently available for stochastic wave-function calculations only. It is slower than using sparse matrices for very small casee, but it is more scalable to large sizes. This approach is not used for density matrices.

While sparse methods are useful for storing operators, these require memory, which must be allocated when the matrices are generated. As a result, the total memory required scales with M^2, apart from the reduction due to sparcity. This can be minimized by only generated the operators that are needed, rather than all possible ones.

When using sparse methods, the full M-mode index $\boldsymbol{n}$ is packed into the first single index n. This is automatic for density matrices, but it is optional for stochastic wavefunction calculations, which can use either sparse or full vectors. For functional methods, the wavefunction is an array of rank $M+1$ for an M mode bosonic case.

10.2. Bose-Hubbard Hamiltonian

A support function is provided to generate the Bose-Hubbard Hamiltonian $\hat{H}$, which in units where $\hbar = 1$, is

$$\hat{H} = \frac{1}{2}\sum_{n,m=1}^{M} \Omega_{m,n}\hat{a}_m^{\dagger}\hat{a}_n + \sum_{n=1}^{M}\kappa_n\left(\hat{a}_n^{\dagger}\hat{a}_n\right)^2$$

The xSPDE input parameters are Ω_{mn}, and κ_n. Scalar nonlinear parameters κ are extended to vectors of length $M = p.modes$. Scalar couplings Ω are extended to vectors of length $M-1$, where

$$\Omega_{mm+1} = \Omega_{m+1m} = \Omega_m$$

so that vector couplings are assumed to hold for linear nearest neighbors. If the last coupling term Ω_M is present, then the model becomes a circular coupled model, with:

$$\Omega_{M,1} = \Omega_{1,M} = \Omega_M$$

The Hamiltonian term $\hat{H}\left|\psi\right\rangle$ is calculated from $BoseHubbardH(\Omega,\kappa,\psi,p)$ in the non-sparse case. For sparse matrices, one use $MkBoseHubbardH(\Omega,\kappa,p)$ tp create the Hamiltonian matrix.

THis can be readily combined with and loss mechanism using resevoir methods.

10.3. Sparse matrix methods

The different approaches have areas of applicability that depend on the Hilbert space dimension. Suppose we use a stochastic method to solve a Lindblad master equation for linear decay with initial condition $\psi_j = \delta_{(N+1)j}$ and $L = a$, $\hat{H} = \hat{a}^{\dagger}\hat{a}$, for $N = 6$, $\gamma = 0.25$. The script below uses a sparse operator method, and compares the solution with an exact result

$$\left\langle\hat{n}\right\rangle = \left\langle\hat{n}\left(0\right)\right\rangle e^{-2\gamma t}.$$

The alternative functional operator method inputs are explained in Section 12.1.

```
clear;
p.name = 'SSE linear decay, N=6';
p.ranges = 2;
p.nmax = 7;
p.sparse = 1;
p.quantum = 1;
p.a = Mkbose(p);
p.ensembles = [100,1,10];
p.gamma{1} = @(p) 0.25;
p.compare{1} = @(p) 6*exp(-0.5*p.t);
p.L{1} = @(~,p) p.a{1};
p.H = @(p) p.a{1}'*p.a{1};
p.diffplot = {1,1};
p.initial = @(~,p) Mknumber(6,p);
p.expect{1} = @(p) p.a{1}'*p.a{1};
p.olabels = {'\langle N \rangle'};
xspde(p);
```

With the sparse method, the function 'Mkbose' is used to create the operator matrix cell array 'p.a', before it is used. These are only generated as needed. For large numbers of modes they can use a large amount of storage, even though they are sparse matrices. The use of p.L{1} indicates the first decay type is a linear loss, but there could be other dissipative processes as well.

The use of p.quantum=1 shows that it is a stochastic wave-function problem, while p.sparse=1 indicates the use of sparse matrices. Here, $p.a\{1\}$ is the matrix version of the operator $\hat{a}_1$, and $p.a\{1\}'$ is the matrix version of the operator $\hat{a}_1^\dagger$. The number operator $\hat{n}_1$ is $p.a\{1\}' * p.a\{1\}$. To use the master equation method, set p.quantum =2 and remove the p.ensembles input.

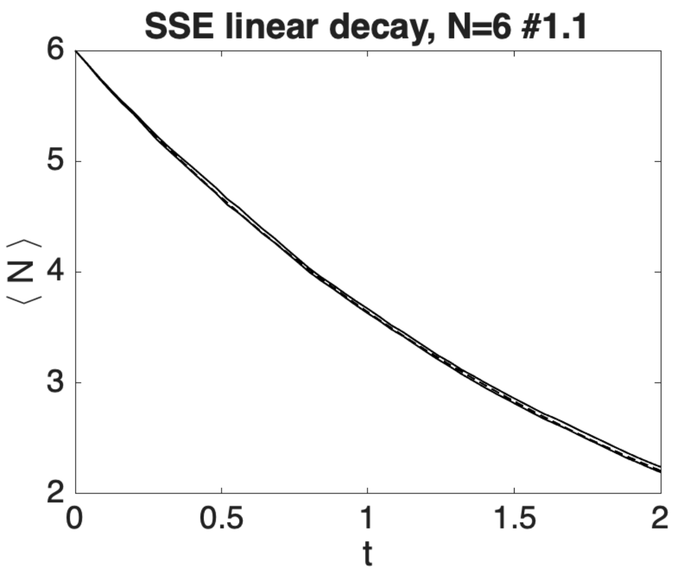


Figure 10.1.: *Example: Linear decay, including a comparison with the exact result, using sparse methods.*

10.3.1. Method versus memory requirements

Algorithms for either wave-functions or density matrices, are selected by choosing two parameters, quantum and sparse. If sparse is omitted, the default is $sparse = 0$.

p.quantum = 1, p.sparse = 0 - this is used to treat a full wave-function, $psi(\boldsymbol{n}, e)$, Here, $\boldsymbol{n} = n_1, \dots n_m$ is a wave-function index index, while e is an ensemble index for random ensembles, if used. Operators are treated as functions, so there are no operator matrices stored. This minimizes the overall memory requirement.

p.quantum = 1, p.sparse = 1 - this is used to treat a packed wave-function, $psi(n, e)$, Here, n is a wave-function index index, which is a packed version of the vector index $\boldsymbol{n}$, while e is an ensemble index for random ensembles. Operators are treated as sparse matrices, so these must be stored. This increases the overall memory usage but is somewhat faster.

p.quantum = 2, p.sparse = 1 - this is used to treat a packed density matrix, $rho(n, \ell)$, Here, n and ℓ are density matrix indices, which are packed version of the vector index $\boldsymbol{n}$. Operators are treated as sparse matrices. Due to the storage requirements of a density matrix, this uses the most memory, and is the fastest. There is no vector ensemble here.

10.4. Input parameters

Input parameters are stored in a structure which is input to the xSPDE program. This is a superset of the parameters already defined. In the definitions below, the structure name is omitted. but we normally use p in the examples. For example, to specify a quantum wave-function method, one would use $quantum = 1$, as explained already. The input parameters can be chosen not just in terms of the problem itself, but also to suit the computational hardware that is available.

Not that while the *quantum* toolbox and *phase* toolbox share common parameters listed below, but they are distinct toolboxes, and one must choose to use either one or the other by setting $quantum > 0$ or $phase > 0$.

10.4.1. Common parameters

modes gives the number of modes, hence $modes = 3$ defines a 3 mode quantum system,. This can be given implicitly through nmax.

ensembles(1) gives a vector of trajectories, $e = 1, \ldots$ensembles(1). This is fast, but increases memory use. It is not used for density matrices.

ensembles(2) gives the number of series repeats for stochastic ensembles. It is always available, but slower.

ensembles(3) gives the number of parallel repeats for stochastic ensembles. It is useful for multicore processors with fast memory.

jump selects either a stochastic differential equation (jump = 0), the default, or a stochastic jump equation (jump = 1).

noises noise dimensions, set automatically for the built-in quantum methods.

points The number of integration points in time for data outputs. The default setting is 51.

steps The integration steps used per time-step, used to reduce time-step errors. The default is 1.

ranges The total integration range in time. The default setting is10.

initial The initial state is given by a function *initial*. This returns a column vector of size $fields \times 1$ or $fields \times ensembles(1)$, for wave-functions, or else of size $fields \times fields$ for density matrix calculations. The default is the state with the first level occupied.

inrandoms are initial random number dimensions. They specify the first argument of the function $initial(v, p)$ as a real Gaussian noise vector v with unit variance and length inrandoms. These are used for an initially decoherent, randomized wave-function.

The internal variable fields is used to specify the dimension of the integrated variables, and is automatically set.

10.4.2. Quantum parameters

quantum is the type of problem: $quantum = 1$ for a wave-function, $quantum = 2$ for a density matrix.

sparse indicates sparseness: if $sparse = 1$, sparse matrices are used to store operators. The default is $sparse = 0$.

nmax is the Hilbert dimension per mode. If this is a vector, the dimension can be varied.

Mk... is a make function to generate wavefunctions or sparse operators where required, eg, **Mkbose**.

operator: If sparse = 0, an operator is a function with inputs of the mode index (or indices), and the wave-function *psi*. Operators acting on multiple modes may have two or more indices. The function O_k returns a wave-function $\hat{O}_k |\psi\rangle$.

sparse_operator: When sparse = 1, operators are sparse matrices. This is faster, but uses more memory.

Hamiltonian: the function $H(psi, p)$ returns a wave-function $\hat{H} |\psi\rangle$, if sparse = 0. Otherwise, if sparse = 1, it is an operator function $H(p)$ that returns a sparse matrix.

The following defaults are used to simplify input:

- If modes is not specified, it is equal to the length of nmax.
- If modes and nmax are not specified the default is a single qubit: modes=1 , nmax=2.
- If the nmax vector is shorter than modes, the last value of nmax is repeated as necessary.

10.5. Dissipative parameters

In order to explain the terminology for dissipative input, the following list is useful. There are some differences that depend on whether one uses sparse matrices or functional operators. In the list below, n is the channel index for the dissipative operators. One channel index can generate any number of mode operators of the same type.

L{n}: this is a cell array of dissipative functions. The first argument is the mode index, k, or a vector of two indices, $[k_1, k_2]$, and the last argument p is the parameter structure.

@(k,p) is used for sparse operators, and returns a sparse matrix $\hat{L}_k$.

@(k,psi) is used for operator functions, returning $\hat{L}_k |\psi\rangle$.

Conjugate operators: for functions, a conjugate is returned if the index or indices is negative.

gamma{n}(p): This is a cell array of functions for every type of damping process. Cell array indices are used to distinguish different dissipative processes. These return a vector or matrix of damping rates for each type of Lindblad operator. The resulting vector indices give the mode (or modes, if the operator acts on more than one mode).

alpha{n}(k): This is a cell array of noise amplitude vectors or matrices for each type of damping process with real noises. If **alpha** is zero, which is the default, a complex noise is used.

measure: This gives the number of measured channel operators.

Note that:

- Operators may have one or two mode indices.
- Functional operators are slower than sparse operators for small Hilbert spaces.
- Only the required index combinations are accessed by the Lindblad functions, reducing storage.

10.6. Support functions for operators

These are linear functions that act on the quantum wave-function. New ones can readily be added. They reduce memory requirements, which is an advantage for large Hilbert spaces, where storing even sparse operators can require large quantities of memory.

The xSPDE code includes internal functions for bosonic and spin operators. The predefined operators also return auxiliary quantities used in dissipative equations if required, as they have variable input and output lists.

10.6.1. Bosonic operators

Label	Inputs	Output(s)
A	(m, psi)	$\hat{a}_m |\psi\rangle$
A2	(m, psi)	$\hat{a}^2_m |\psi\rangle$
N	$([m_1\,(, m_2)], psi)$	$\hat{a}^\dagger_{m_1}\hat{a}_{m_2} |\psi\rangle$
BoseHubbardH	(O, K, p, psi)	Bose-Hubbard Hamiltonian
Mkbose	$([list,]p)$	Makes sparse bosonic operator matrices
Mknumber	(nv, p)	Makes number states with occupations nv
MkBoseHubbard	(O, K, p)	Makes Bose-Hubbard Hamiltonian

Operators have scalar or vector indices. For a complete description, see (15.4.2).

10.6.2. Qubit and Pauli spin operators

The following set of operators are used for spin chain evolution.

Label	Inputs	Output(s)
sx	(m, psi)	$\hat{\sigma}^x_m \|\psi\rangle$
sy	(m, psi)	$\hat{\sigma}^y_m \|\psi\rangle$
sz	(m, psi)	$\hat{\sigma}^z_m \|\psi\rangle$
sx2	$([m_1, m_2], psi)$	$\hat{\sigma}^x_{m_1}\hat{\sigma}^x_{m_2} \|\psi\rangle$
sy2	$([m_1, m_2], psi)$	$\hat{\sigma}^y_{m_1}\hat{\sigma}^y_{m_2} \|\psi\rangle$
sz2	$([m_1, m_2], psi)$	$\hat{\sigma}^z_{m_1}\hat{\sigma}^z_{m_2} \|\psi\rangle$

10.6.3. Qubit gate operators

The following operators can be used to implement quantum logic gates, in addition to the standard Pauli operators. These assume qubit or two-state qubit logic in each mode.

Label	Inputs	Output(s)
cx	$([m_1, m_2], psi)$	Controlled Not
ha	(m, psi)	Hadamard
p8	(m, psi)	$\pi/8$
ph	(m, psi)	Phase

10.7. Sparse operators

The xSPDE code includes internal functions to generate operators. These are either sparse or full. Sparse operators are generated if needed requiring a mk function call to create the required index combinations, before they are used.

10.7.1. Sparse bosonic operators: Mkbose

These are a cell array of annihilation operators, generated using Mkbose.

Label	Indices	Meaning
a	$\{m\}$	$\hat{a}_m$
a'	$\{m\}$	$\hat{a}^\dagger_m$

p.a = Mkbose((list,) p) Returns a cell array of annihilation operators defined either at all modes, if there is no list, or at the listed mode locations. Here list is a vector of integers, p is the parameter structure.

10.8. Observe, expect, output and compare

There are four types of possible outputs. The observe and compare functions are computed during the time-evolution, so that the entire wave-function doesn't need to be stored in time, reducing the storage needs. Additional functional transformations for either can be used as well, called output functions. Finally, a compare function allows comparison plots.

observe is a cell array of any stochastic function. xSPDE expects a (named or anonymous) function that takes two parameters, namely the wave-function *psi* or density matrix ρ, and the input structure p. The function return a real or complex matrix of dimension $(\ell, ensembles(1))$, where ℓ indexes a vector observable. xSPDE then averages over the second index, to calculate the observable. This allows an average of any type.

expect is a cell array of operators defining a quantum expectation value. For full matrices, expect is a (named or anonymous) function that takes two inputs, the wave-function *psi* and the structure p. For sparse matrices, the expect function returns a matrix. xSPDE internally averages over both the quantum and stochastic degrees of freedom to calculate the observable.
To plot the mean number in mode $m = 1$, using:
a) the number operator $\hat{n} = \hat{a}^{\dagger}\hat{a}$ with sparse operators:

```
p.expect{1} = @(p) p.a{1}'*p.a{1};
```

b) the number operator $\hat{n} = \hat{a}^{\dagger}\hat{a}$, with function calls:

```
p.expect{1} = @(psi,p) n(1,psi);
```

output All observe and expect results are stored. Transformations of both can be introduced. These may include multiple averages and/or different times. These are called the output functions. Default outputs pass through observe and expect results with no change. Defined outputs, like p.output{1}, replace the defaults, or add new outputs. Graphed data uses the outputs, which include sampling errors if ensembles are used, and step-size errors if checks is turned on.

compare Comparison functions can be used to obtain comparison graphs and differences.

The output numbering that is used is the **same** for all four types of function. This can lead to overwriting, with the precedence that output>expect>observe. To prevent overwriting, use different cell-indices. Compare functions are plotted independently, as an extra line on an existing graph, so they don't overwrite, and can be compared with any of output, expect, observe.

10.9. SSE derivative

The SSE derivative terms are calculated from $SSE(a, w, p)$ for solving Eq 11.14. The equation can be solved by any *method* for a Stratonovich SDE. Projective normalization of wavefunction equations is automatic for the standard methods of xSPDE. Mathematical details are given in Section (11.3).

10.10. Solving with the MCWF method

At each time step in the numerical simulation with the MCWF method, the jump probability ΔP is first calculated. This is carried out by computing the jump probability per unit time of each jump operator L_m in the master equation, which is given by

$$\Delta P_m = 2\gamma_m \langle \psi(t)| L_m^\dagger L_m |\psi(t)\rangle \,. \tag{10.2}$$

The calculated jump rate ΔP_m is then compared with a uniform, randomly generated number r_m between zero and $1/\Delta t$. If $\Delta P_m > r_m$, the state then undergoes the given jump. As $\Delta t \to 0$, a jump in any given step become increasingly rare.

After this, the state vector evolves according to the non-hermitian Hamiltonian H_{eff} in Eq. (11.27) as follows:

$$\frac{d}{dt}|\psi(t)\rangle = -iH_{eff}|\psi(t)\rangle \tag{10.3}$$

This differential equation is solved by a midpoint or Runge-Kutta algorithm, or others available. These steps are repeated till the final time step, and they constitute a single trajectory. Many trajectories are taken to compute the expectation values for the observables of interest.

For error-checking, fine step results are checked against a coarse step with a noise given by $r_c = \min(r_1, r_2)$, so that the coarse jump occurs if a jump takes place in either fine step. This allows errors due to step-size to be accurately estimated by comparing the fine and coarse step-size results, just as with continuous noise.

The MCWF algorithm described above is carried out simply by setting $p.jump = 1$. No further inputs from the user are required. All other parameters are input in exactly the same way as in the SSE numerical simulation. Projective normalization of wavefunction equations is automatic for the standard methods of xSPDE. Mathematical details are given in Section (11.4).

11. Quantum Monte-Carlo theory

This chapter describes the quantum theory used in xSPDE, to explain the background to the open system methods available.

11.1. Master equations

The master equation [2, 75] is a standard tool for solving Markovian open quantum system dynamics:

$$\begin{aligned}\dot{\rho} &= \mathcal{L}_J \rho && (11.1)\\ &= -i\left[\hat{H}, \rho\right] + \sum_{j=1}^{J} \gamma_j \left(2\hat{L}_j \rho \hat{L}_j^\dagger - \hat{L}_j^\dagger \hat{L}_j \rho - \rho \hat{L}_j^\dagger \hat{L}_j\right) && (11.2)\end{aligned}$$

Here, $\mathcal{L}_J$ is the total super-operator for J terms,H is the reversible system Hamiltonian, L_j are J operators that couple the system to the dissipative reservoir, and γ_j is the decay rate. The dissipative operators can be further classified by type n and mode index k, including vector indices if needed.

Provide that ρ is normalized, the expectation values of observables $\hat{O}$ are given by:

$$\left\langle \hat{O} \right\rangle = Tr\left(\rho \hat{O}\right) \tag{11.3}$$

The definitions used here mean that for the case of linear damping with $\hat{L} = \hat{a}$, the rate γ is the amplitude decay rate. This abstract notation does not include the effects of finite temperatures, which are explained below.

The rate is written explicitly here. This is useful for xSPDE inputs, which use standard dimensionless operators. Alternative approaches include combining the rate with the operator [116], implying $\gamma = 1$. Others use a rate constant $\kappa = 2\gamma$, i.e., the number decay rate. Some combine this with the operator, defining $\hat{c}_j = \sqrt{2\gamma_j}\hat{L}_j$, giving a fourth operator convention.

Including finite temperature reservoir occupation numbers $\bar{n}_j$ explicitly, the quantum master equation with damping rates Γ_j is

$$\begin{aligned}\frac{\partial \hat{\rho}}{\partial t} &= -i\left[\hat{H}, \hat{\rho}\right] + \sum_j \Gamma_j \left(\bar{n}_j + 1\right)(2\hat{A}_j \hat{\rho} \hat{A}_j^\dagger - \hat{A}_j^\dagger \hat{A}_j \hat{\rho} - \hat{\rho} \hat{A}_j^\dagger \hat{A}_j)\\ &\quad + \sum_j \Gamma_j \bar{n}_j (2\hat{A}_j^\dagger \hat{\rho} \hat{A}_j - \hat{A}_j \hat{A}_j^\dagger \hat{\rho} - \hat{\rho} \hat{A}_j \hat{A}_j^\dagger)\ , && (11.4)\end{aligned}$$

Damping operator ($\hat{A}_j$)	Γ_j	Physical interpretation
$\hat{a}_j$	γ_j	Linear amplitude loss (units s^{-1})
$\hat{a}_j^\dagger$	g_j	Linear amplitude gain (units s^{-1})
$\hat{a}_j^\dagger \hat{a}_j$	γ_j^p	Phase decay rate gain (units s^{-1})
$\hat{a}_j^2$	$\kappa_j/2$	nonlinear amplitude loss (units s^{-1}).

Table 11.1.: Typical types of quantum decoherence term. Note that when the damping operator is a number operator, it conserves particle number but causes phase coherence decay.

where Γ_j is a zero temperature damping rate for reservoir couplings to the operator $\hat{A}_j$, $\bar{n}_j$ is the finite temperature reservoir occupation. In the numerical toolbox, the finite-temperature reservoirs are included explicitly as a separate Lindblad term.

11.1.1. Bosonic Hilbert spaces

The operators available in xSPDE include multimode bosonic operators $\hat{a}_j$. For these, typical damping operators are:

11.1.2. Qubit Hilbert spaces

Additionally, xSPDE includes finite Hilbert spaces, focusing on $SU(2)$ or qubit cases. Operators available are: $\hat{\sigma}_m^x$,$\hat{\sigma}_m^y$,$\hat{\sigma}_m^z$, together with quantum logic gates: hadamard, controlled-not, phase and $\pi/8$. These can be combined to give a complete set of logic gates, allowing a simulation of quantum computers.

Using the master-equation toolbox, one can also include decoherence and loss. As usual, this is limited by exponential growth in the Hilbert space dimension, but the stochastic Schrödinger equation and related methods improve memory efficiency compared to the full density matrix.

11.2. Stochastic Schrödinger equation (SSE)

While master equations can be solved directly, they grow in size quadratically with Hilbert space dimension. An alternative to reduce memory size is to use quantum Monte Carlo methods, which although still restricted to small mode numbers, can reduce memory requirements substantially.

A stochastic Schrödinger equation (SSE) is an equation with noise terms used to solve a dissipative master equation by random sampling, and was originally developed for applications in quantum foundations [116–118]. It has the advantage over a master equation that for large numbers of modes it uses less storage. This requires $e^{\lambda M}$complex numbers for the storage of an $M-$mode quantum system, compared to $\sim e^{2\lambda M}$ for the master equation, where $\lambda = \log_e(N)$ for an N-level local Hilbert space. This is exponentially large, but the memory required is less than with a density matrix equation, doubling the number of modes that are accessible. The drawback is that many parallel trajectories must be averaged in order to give a low final sampling error.

An SSE can also be regarded in certain cases as providing a direct simulation of the measurement process, which means that the information recorded in a simulation is similar to the information measured in a quantum experiment. This requires a suitable choice of the method, sometimes called an "unraveling" and the stochastic integration algorithm. Different unravellings mean different measurements and different convergence rates.

There are many versions of the SSE, which use different normalization, different random noises or different types of stochastic calculus. Noises can be real or complex, and either continuous or with discrete jumps. These are different "unravellings", and correspond physically to distinct measurement devices and outcomes. They also have different sampling errors.

Compared to phase-space expansions, the SSE method has the problem that storage requirements are exponential in the system size, although it may have lower sampling errors for high nonlinearities. This limits mode numbers to $10-50$, depending on the size of the Hilbert space per mode and the computational resources. The approach is most useful for small mode numbers, especially for large nonlinearities.

11.2.1. Normalized Ito SSE

A widely used form of the continuous noise, normalized SSE is as follows [116], in the Ito calculus:

$$\begin{aligned} d\left|\Psi\right\rangle &= \left\{-i\hat{H}+\sum_j \gamma_j\left(2\left\langle \hat{L}_j^\dagger\right\rangle_\Psi \hat{L}_j - \left\langle \hat{L}_j^\dagger\right\rangle_\Psi \left\langle \hat{L}_j\right\rangle_\Psi - \hat{L}_j^\dagger \hat{L}_j\right)\right\}\left|\Psi\right\rangle dt \\ &+ \sum_j \sqrt{2\gamma_j}\Delta\hat{L}_j\left|\Psi\right\rangle d\xi_j . \end{aligned} \tag{11.5}$$

where $\Delta\hat{L}_j = \hat{L}_j - \left\langle \hat{L}_j\right\rangle_\Psi$, and

$$\begin{aligned} \left\langle d\xi_j^* d\xi_k\right\rangle &= \delta_{jk}dt. \\ \left\langle d\xi_j d\xi_k\right\rangle &= 0. \end{aligned} \tag{11.6}$$

A more general form of the normalized SSE [119] in the Ito calculus is:

$$\begin{aligned} d\left|\Psi\right\rangle &= \left\{-i\hat{H}+\sum_j \gamma_j\left(2\left\langle \hat{L}_j^\dagger\right\rangle_\Psi \hat{L}_j - \left\langle \hat{L}_j^\dagger\right\rangle_\Psi \left\langle \hat{L}_j\right\rangle_\Psi - \hat{L}_j^\dagger \hat{L}_j\right)\right\}\left|\Psi\right\rangle dt \\ &+ \sum_{jn} \sqrt{2\gamma_j}\Delta\hat{L}_j\left|\Psi\right\rangle \alpha_{jn} d\zeta_{jn} . \end{aligned} \tag{11.7}$$

where we require that $\sum_n |\alpha_{jn}|^2 = 1$, and

$$\left\langle d\zeta_{jn}(t)\, d\zeta_{km}\left(t'\right)\right\rangle = \delta_{jk}\delta_{nm}dt. \tag{11.8}$$

One can also include a unitary transformation, which we set to a delta function for simplicity. When there is one noise per decay channel, then $\alpha_j(t) = e^{i\phi_j}$, where ϕ_j is arbitrary. If there are two noises, then one can choose $\alpha_{j1}(t) = 1/\sqrt{2}$, and $\alpha_{j2}(t) = i/\sqrt{2}$, giving a complex noise SDE, with $\xi_j = (\zeta_1 + i\zeta_2)/\sqrt{2}$, as above.

This can be written for $|\Psi\rangle \to \Psi_\nu$, and $d\zeta_{jn} \to dw_\sigma$, as:

$$d\psi_\mu = A_\mu dt + B_{\mu\sigma}dw_\sigma$$

11.3. Stratonovich SSE

To obtain standard calculus for an SSE, one must transform to the Stratonovich equation. This form of stochastic calculus allows integration algorithms that often give lower errors [28]. There are also higher order methods for Ito equations, but these have greatly increased complexity. Here we derive the Stratonovich correction [1, 43], which is obtained with $\hat{L}_j|\Psi\rangle \to L_{j\mu\nu}\Psi_\nu$ so that $\left\langle \hat{L}_j^\dagger \right\rangle_\Psi = \sum_{\sigma\rho}\Psi_\sigma^* L_{j\rho\sigma}^*\Psi_\rho$.

For complex noise as in Eq (11.6), the Stratonovich drift is given by:

$$A_\mu = A_\mu^{(I)} - \frac{1}{2}\sum_{j\nu} B_{\nu j}^*\partial_\nu^* B_{\mu j}. \tag{11.9}$$

On taking matrix elements in an orthogonal basis, and defining:

$$\Delta L_{j\mu\nu} = L_{j\mu\nu} - \delta_{\mu\nu}\sum_{\sigma\rho}\Psi_\rho^* L_{j\rho\sigma}\Psi_\sigma, \tag{11.10}$$

one has:

$$\begin{aligned} B_{\mu j} &= \sqrt{2\gamma_j}\Delta L_{j\mu\beta}\Psi_\beta \\ B_{\mu j}^* &= \sqrt{2\gamma_j}\Psi_\beta^*\Delta L_{j\mu\beta}^*. \end{aligned} \tag{11.11}$$

On differentiating one therefore obtains:

$$\begin{aligned} \partial_\nu^* B_{\mu j} &= \sqrt{2\gamma_j}\sum_\beta\left[-\Psi_\beta\partial_\nu^*\left(\delta_{\mu\beta}\sum_{\sigma\rho}\Psi_\rho^* L_{j\rho\sigma}\Psi_\sigma\right)\right] \\ &= \sqrt{2\gamma_j}\left[-\Psi_\mu\sum_\sigma L_{j\nu\sigma}\Psi_\sigma\right]. \end{aligned} \tag{11.12}$$

The Stratonovich correction is given by:

$$\begin{aligned}
-\frac{1}{2}\sum_{\nu,j}B^*_{\nu j}\partial^*_\nu B_{\mu j} &= -\sum_j \gamma_j \sum_{\nu,\alpha}\Psi^*_\alpha \Delta L^*_{j\nu\alpha}\left[-\Psi_\mu \sum_\sigma L_{j\nu\sigma}\Psi_\sigma\right] \\
&= \sum_j \gamma_j \left[\sum_{\nu\sigma\alpha}\Psi^*_\alpha\left(L^*_{j\nu\alpha} - \delta_{\nu\alpha}\left\langle \hat{L}^\dagger_j\right\rangle_\Psi\right) L_{j\nu\sigma}\Psi_\sigma\right]\Psi_\mu \\
&= \sum_j \gamma_j \left[\left\langle \hat{L}^\dagger_j \hat{L}_j\right\rangle_\Psi - \left\langle \hat{L}^\dagger_j\right\rangle_\Psi \left\langle \hat{L}_j\right\rangle_\Psi\right]\Psi_\mu . \qquad (11.13)
\end{aligned}$$

In summary, the complex Ito SSE can be transformed to a nonlinear Stratonovich stochastic differential equation which locally preserves normalization for zero step-size [120]. This is called the quantum state diffusion model:

$$\begin{aligned}
\frac{d\left|\Psi\right\rangle}{dt} &= \left(-i\hat{H} + \sum_j \gamma_j \left(2\Delta\hat{L}_j\left\langle \hat{L}^\dagger_j\right\rangle_\Psi - \Delta\left[\hat{L}^\dagger_j\hat{L}_j\right]\right)\right)\left|\Psi\right\rangle \\
&\quad + \sum_j \sqrt{2\gamma_j}\xi_j \Delta\hat{L}_j\left|\Psi\right\rangle \qquad (11.14)
\end{aligned}$$

where:

$$\left\langle \xi_k\left(t\right)\xi^*_j\left(t'\right)\right\rangle = \delta_{kj}\delta\left(t-t'\right). \qquad (11.15)$$

Here, $\Delta\hat{L}_j \equiv \hat{L}_j - \left\langle \hat{L}_j\right\rangle_\Psi$ and the equation uses Stratonovich calculus. This preserves the norm of the wave-function. Suppose the Stratonovich form has a dissipative term $\Delta\hat{\mathcal{L}}_s$, where

$$\frac{d\left|\Psi\right\rangle}{dt} = \left\{\Delta\hat{\mathcal{L}}_s - i\hat{H}\right\}\left|\Psi\right\rangle \qquad (11.16)$$

Since it is a Stratonovich equation, one can use ordinary calculus rules. Only dissipative terms can change the norm, and:

$$\begin{aligned}
\frac{d}{dt}\left\langle \Psi \middle| \Psi\right\rangle &= \left\langle \Delta\left(\hat{\mathcal{L}}_s + \hat{\mathcal{L}}^\dagger_s\right)\right\rangle_\psi \\
&= \left\langle \hat{\mathcal{L}}_s + \hat{\mathcal{L}}^\dagger_s\right\rangle_\psi - \left\langle \hat{\mathcal{L}}_s + \hat{\mathcal{L}}^\dagger_s\right\rangle_\psi = 0 \qquad (11.17)
\end{aligned}$$

For Ito equations, the trajectories have a norm error that grows with time. While there are projective methods to prevent this, the result has higher step-size errors [36].To obtain observables, one must use the "double" expectation indicating a quantum and stochastic mean, where the wave-functions $\left|\Psi\right\rangle$ are normalized, and have all the same weight:

$$\left\langle O\right\rangle \equiv \left\langle \left\langle \Psi\right| O\left|\Psi\right\rangle\right\rangle_\xi . \qquad (11.18)$$

Integrating this equation is best carried out with a projection at each time-step to prevent the normalization changing, as derived elsewhere [36]. This is implemented automatically within xSPDE.

11.3.1. Real noise Stratonovich equation

For the real noise case, the correction term is:

$$A_\mu = A_\mu^{(I)} - \frac{1}{2}\sum_{\sigma\nu} \left(B_{\nu\sigma}\partial_\nu + B^*_{\nu\sigma}\partial^*_\nu\right) B_{\mu\sigma}, \tag{11.19}$$

where $B_{\mu j} = \sqrt{2\gamma_j}\Delta L_{j\mu\beta}\Psi_\beta$. Taking $n = 1$ and $j = \sigma$, the conjugate correction is given above and is independent of α_j. The first term is obtained from differentiation of the noise matrix:

$$B_{\mu j} = \alpha_j\sqrt{2\gamma_j}\Delta L_{j\mu\beta}\Psi_\beta. \tag{11.20}$$

hence one obtains that:

$$\partial_\nu B_{\mu j} = \sqrt{2\gamma_j}\alpha_j \left[\Delta L_{j\mu\nu} - \sum_\beta \delta_{\mu\beta}\left[\Psi^*_\rho L_{j\rho\nu}\right]\Psi_\beta\right]. \tag{11.21}$$

The additional correction is as follows:

$$\begin{aligned}
-\frac{1}{2}\sum_{\nu j} B_{\nu j}\partial_\nu B_{\mu j} &= -\sum_{\nu j}\gamma_j\alpha_j^2\left[\Delta L_{j\mu\nu} - \Psi^*_\rho L_{j\rho\nu}\Psi_\mu\right]\Delta L_{j\nu\sigma}\Psi_\sigma \\
&= -\sum_{\nu j}\gamma_j\alpha_j^2\left(\Delta L_{j\mu\nu}\Delta L_{j\nu\sigma}\Psi_\sigma - \left[\Psi^*_\rho L_{j\rho\nu}\Delta L_{j\nu\sigma}\Psi_\sigma\right]\Psi_\mu\right).
\end{aligned} \tag{11.22}$$

Written in operator/wave-function terminology, the real correction $|\delta A^r\rangle$is

$$\begin{aligned}
|\delta A^r\rangle &= -\sum_{\nu j}\gamma_j\alpha_j^2\left(\left[\Delta\hat{L}_j\Delta\hat{L}_j\right] - \left\langle \hat{L}_j\Delta\hat{L}_j\right\rangle_\Psi\right)|\Psi\rangle \\
&= -\sum_{\nu j}\gamma_j\alpha_j^2\left(\left[\hat{L}_j^2 - 2\hat{L}_j\left\langle\hat{L}_j\right\rangle_\Psi + \left\langle\hat{L}_j\right\rangle_\Psi^2\right] - \left\langle\hat{L}_j^2\right\rangle_\Psi + \left\langle\hat{L}_j\right\rangle_\Psi^2\right)|\Psi\rangle \\
&= \sum_{\nu j}\gamma_j\left(2\Delta\hat{L}_j\left\langle\alpha_j^2\hat{L}_j\right\rangle_\Psi - \Delta\left[\alpha_j^2\hat{L}_j^2\right]\right)|\Psi\rangle.
\end{aligned} \tag{11.23}$$

Combining both terms, and defining $\hat{X}_j = \hat{L}_j^\dagger + \alpha_j^2\hat{L}_j$, one obtains a result known in the literature [121] for the case $\alpha = 1$;

$$\begin{aligned}
\frac{d|\Psi\rangle}{dt} &= \left\{-i\hat{H} + \sum_j\gamma_j\left(2\left\langle\hat{X}_j\right\rangle_\Psi\Delta\hat{L}_j - \Delta\left[\hat{X}_j\hat{L}_j\right]\right)\right\}|\Psi\rangle \\
&\quad + \sum_j\sqrt{2\gamma_j}\alpha_j\zeta_j(t)\Delta\hat{L}_j|\Psi\rangle
\end{aligned} \tag{11.24}$$

As with the complex noise case, this is explicitly norm-preserving since the dissipative terms have zero quantum mean values for every noise realization. This generic result reduces to the complex case if one sets$\hat{X}_j = \hat{L}_j^\dagger$ and $\alpha_j\zeta_j \to \xi_j$.

11.4. Monte Carlo wave-function method

The Monte Carlo or quantum jump method is another approach to solve a master equation. The master equation treated here has the form given in Eq (11.1).

11.4.1. Integer noise

A jump SSE is obtained by using an Ito stochastic differential equation with real noise, in the form:

$$\begin{aligned} d\left|\phi\right\rangle &= \left\{-i\hat{H} - \sum_j \gamma_j \left[\hat{L}_j^\dagger \hat{L}_j - \left\langle \hat{L}_j^\dagger \hat{L}_j \right\rangle\right]\right\} \left|\phi\right\rangle dt \\ &+ \sum_j \left(\hat{L}_j / \sqrt{\left\langle \hat{L}_j^\dagger \hat{L}_j \right\rangle} - 1\right) \left|\phi\right\rangle dN_j, \end{aligned} \tag{11.25}$$

where the real integer noise $dN = [0, 1]$ has correlations of:

$$\left\langle dN_j\left(t\right)\right\rangle = 2\gamma_j \left\langle \hat{L}_j^\dagger \hat{L}_j \right\rangle dt. \tag{11.26}$$

In any interval dt, dN is unity with probability $p = 2\gamma_j \left\langle \hat{L}_j^\dagger \hat{L}_j \right\rangle dt$, and zero otherwise.

To generate integer noise, one first obtains a random real number r where $0 < r < 1/dt$. From this, one can choose $dN = 1$ if $r < 2\gamma_j \left\langle \hat{L}_j^\dagger \hat{L}_j \right\rangle$.

11.4.2. MCWF method

In the MCWF method, state vectors evolve according to an effective Hamiltonian,

$$H_e = H - i \sum_m \gamma_m L_m^\dagger L_m \,, \tag{11.27}$$

punctuated by quantum jumps

$$|\psi\rangle \rightarrow L_m |\psi\rangle \,, \tag{11.28}$$

where L_m is one of the possible operators in the master equation. At each step in time, the system will either evolve according to the non-Hermitian Hamiltonian Eq. (11.27) or undergo a jump operation, depending on the jump probability ΔP.

A sequence of quantum jumps or photo-counts giving total counts $\boldsymbol{c} = c_1, \ldots c_M$ is obtained. For times when there is no jump,

$$\dot{\psi}_c = -i\hat{H}_e \psi_c. \tag{11.29}$$

Jumps occur at random times given by choosing random numbers r_m such that $0 < r_m < 1/\Delta t$, where r_m determines the jump probability for the $m - th$ process.

The jump changes counts so that $c_j \to c_j + 1$. Afterwards, one resets ρ_c after an infinitesimal time ϵ so that

$$|\psi\left(t_c+\epsilon\right)\rangle = \frac{|\psi_j\rangle}{\sqrt{\langle\psi_j|\ \psi_j\rangle}}. \tag{11.30}$$

The MCWF algorithm is presented in the numerical section.

11.4.3. Monte-Carlo master equations

Monte Carlo master equation theory [38,39,122–125] implements the Copenhagen model for measurement as a sequential wave-function projections. It treats dissipative evolution whose average behavior is given by a master equation, where if there is one decay channel $\hat{L}_j$ per mode M:

$$\frac{d\rho}{dt} = -i\left[\hat{H},\rho\right] + \sum_{j=1}^{M}\gamma_j\left(2\hat{L}_j\rho\hat{L}_j^\dagger - \left[\hat{L}_j^\dagger\hat{L}_j,\rho\right]_+\right). \tag{11.31}$$

An equivalent sequence of quantum jumps or photo-counts giving total counts $\boldsymbol{c} = c_1,\ldots c_M$ is described by a conditional density matrix equation, which is a nonlinear Ito discrete SDE in the form:

$$d\rho = -i\left[\hat{H}_e\rho - \rho\hat{H}_e^\dagger\right]dt + \sum_j\left(\frac{\hat{L}_j\rho\hat{L}_j^\dagger}{\left\langle\hat{L}_j^\dagger\hat{L}_j\right\rangle} - \rho\right)dN_j$$
$$, \tag{11.32}$$

where the effective Hamiltonian $\hat{H}_e$ is non-hermitian:

$$\hat{H}_e = H - i\sum_{j=1}^{M}\gamma_j\left[\hat{L}_j^\dagger\hat{L}_j - \left\langle\hat{L}_j^\dagger\hat{L}_j\right\rangle\right]$$

and the real integer noise $dN_j = [0,1]$ has correlations of:

$$\langle dN_j\left(t\right)\rangle = 2\gamma_j\left\langle\hat{L}_j^\dagger\hat{L}_j\right\rangle dt. \tag{11.33}$$

In any interval dt, dN is unity with probability $p = 2\gamma_j\left\langle\hat{L}_j^\dagger\hat{L}_j\right\rangle dt$, and zero otherwise. This is not a standard Lindblad form due to the nonlinear terms, but it conserves probabilities, and has an average behavior that corresponds to the full master equation.

Jumps occur at times given as above by choosing random numbers r_j in $[0,1/dt]$ such that $dN = 1$ if

$$r_j < 2\gamma_j\left\langle\hat{L}_j^\dagger\hat{L}_j\right\rangle. \tag{11.34}$$

The Ito density matrix equation can be integrated by integrating the deterministic part over a small time interval, then deciding whether or not to jump. A jump changes detector counts so $c_j \to c_j+1$. One must correspondingly project ρ after an infinitesimal time to give the new density matrix, given by the discontinuous jump $d\rho_N$, where

$$d\rho_N = \sum_j \left(\frac{\hat{L}_j \rho \hat{L}_j^\dagger}{\left\langle \hat{L}_j^\dagger \hat{L}_j \right\rangle} - \rho \right) dN_j. \tag{11.35}$$

11.5. Examples

We now consider examples of linear and nonlinear dissipative operators.

11.5.1. Linear master equation

The standard case of linear losses in quantum optics, gives:

$$L = a \tag{11.36}$$

The corresponding master equation is;

$$\dot{\rho} = 2a\rho a^\dagger - a^\dagger a\rho - \rho a^\dagger a. \tag{11.37}$$

This leads to a linear decay in amplitude and occupation number:

$$\begin{aligned} \langle \dot{n} \rangle &= Tr\left[\left(2a\rho a^\dagger a^\dagger a - n^2\rho - \rho n^2 \right) \right] \\ &= 2Tr\left[\rho a^{\dagger 2} a^2 - n^2 \rho \right] \\ &= 2Tr\left[\rho\left(n^2 - n\right) - n^2\rho \right] \\ &= -2\langle n \rangle . \end{aligned} \tag{11.38}$$

The effect of the operator on the state expansion is

$$\begin{aligned} |\phi\rangle &= \sum_n \phi_n |n\rangle \\ a|n\rangle &= \sqrt{n}\,|n-1\rangle \\ a^\dagger |n\rangle &= \sqrt{n+1}\,|n+1\rangle \\ a^\dagger a|n\rangle &= n|n\rangle . \end{aligned} \tag{11.39}$$

Therefore for a number state expansion of the density operator:

$$\begin{aligned} \langle a \rangle &= \sum_{nm} \phi_j^* \langle m| \phi_n a |n\rangle \\ &= \sum_{nm} \phi_j^* \langle m| \phi_n \sqrt{n}\,|n-1\rangle \\ &= \sum_{n=0}^{\infty} \phi_n^* \phi_{n+1} \sqrt{n+1}. \end{aligned} \tag{11.40}$$

also, for the conjugate,

$$\left\langle a^{\dagger}\right\rangle = \sum_{n=1}^{\infty}\phi_n^*\phi_{n-1}\sqrt{n}. \tag{11.41}$$

11.5.2. Linear stochastic equation

This is the simplest case:

$$\begin{aligned}\frac{d\left|\phi\right\rangle}{dt} &= \sum_n\left(-a^{\dagger}a + a\xi\right)\phi_n\left|n\right\rangle \\ &= \sum_n\left(-n\left|n\right\rangle + \sqrt{n}\left|n-1\right\rangle\xi\right)\phi_n. \end{aligned}\tag{11.42}$$

Taking matrix elements, one obtains:

$$\frac{d\phi_j}{dt} = \sqrt{m+1}\phi_{m+1}\xi - m\phi_j$$

11.5.3. Normalized, nonlinear stochastic equation

$$\begin{aligned}\frac{d\left|\phi\right\rangle}{dt} &= \sum_n\left(\left[\left\langle a^{\dagger}a\right\rangle - a^{\dagger}a\right] + \left[a - \left\langle a\right\rangle\right]\left[\xi + 2\left\langle a^{\dagger}\right\rangle\right]\right)\phi_n\left|n\right\rangle \\ &= \sum_n\left(\left[\left\langle n\right\rangle - n\right]\left|n\right\rangle + \left[\xi + 2\left\langle a^{\dagger}\right\rangle\right]\left[\sqrt{n}\left|n-1\right\rangle - \left\langle a\right\rangle\left|n\right\rangle\right]\right)\phi_n\end{aligned}$$

Taking matrix elements,

$$\frac{d\phi_j}{dt} = \left[\xi + 2\left\langle a^{\dagger}\right\rangle\right]\left[\sqrt{m+1}\phi_{m+1} - \left\langle a\right\rangle\phi_j\right] + \left[\left\langle n\right\rangle - m\right]\phi_j. \tag{11.43}$$

11.5.4. Nonlinear absorber

The next case of nonlinear two-photon losses in quantum optics, gives:

$$L = a^2$$

where we recall that:

$$\begin{aligned}\left|\phi\right\rangle &= \sum_n\phi_n\left|n\right\rangle \\ a^2\left|n\right\rangle &= \sqrt{n\left(n-1\right)}\left|n-2\right\rangle \\ a^{\dagger 2}\left|n\right\rangle &= \sqrt{\left(n+1\right)\left(n+2\right)}\left|n+2\right\rangle \\ a^{\dagger 2}a^2\left|n\right\rangle &= n\left(n-1\right)\left|n\right\rangle\end{aligned}$$

11.5.5. Master equation

The quantum expectations in a pure state are given by:

$$\begin{aligned}\left\langle a^2\right\rangle &= \sum_{nm} \phi_j^* \langle m| \phi_n a^2 |n\rangle \\ &= \sum_{nm} \phi_j^* \langle m| \phi_n \sqrt{n(n-1)}\, |n-2\rangle \\ &= \sum_{n=0}^{\infty} \phi_n^* \phi_{n+2} \sqrt{(n+2)(n+1)}\end{aligned}$$

The diagonal master equation in a number state basis is therefore:

$$\dot{\rho}_n = -2n(n-1)\rho_n + 2(n+1)(n+2)\rho_{n+2}.$$

This equation is generated automatically using the master equation quantum method in the numerical toolbox.

12. Quantum examples

12.1. Linear decay, complex SSE

This solves a standard Lindblad master equation for linear decay with initial condition $\psi_j = \delta_{Nj}$ and $L = \sqrt{\gamma}a$, $\hat{H} = \hat{a}^\dagger \hat{a}$; for $N = 6$, $\gamma = 0.25$:

$$\dot{\rho} = -i[\hat{H}, \rho] + 2L\rho L^\dagger - L^\dagger L\rho - \rho L^\dagger L$$

Function operator method

```
function [e] = SSElin
%Uses an SSE to solve for a linear decay
p.name = 'SSE linear decay, N=6 initial photons';
p.N = 6;
p.nmax = p.N+1;
p.ranges = 2;
p.quantum = 1;
p.ensembles = [100, 10];
p.gamma{1} = @(p) 0.25;
p.H = @(psi,p) N(1,psi);
p.compare = @(p) p.N*exp(-0.5*p.t);
p.L{1} = @A;
p.diffplot = 1;
p.initial = @(w,p) Mknumber(6,p);
p.expect = @(psi,p) N(1,psi);
p.olabels = {'\langle N \rangle'};
e = xspde(p);
end
```

With the function method, the function 'Mkbose' is not required. Instead, the effect of the operators is obtained through a function call to the handle '@a' . For large numbers of modes this method uses a reduced amount of memory as there is no stored matrix involved in this case.

One cannot simply write p.H=@n here, because the Hamiltonian is a function of the wave-function *psi* and the parameters p, while the number operator is a function of the mode number and the wave-function. For Lindblad operators, these arguments are inserted automatically.

The flag p.diffplot =1 is used by the graphics code to create a plot of the difference between the comparison solution and the simulation.

Note that one can determine the relative size of the sampling errors and step-size errors from the difference plot, although these are also printed out.

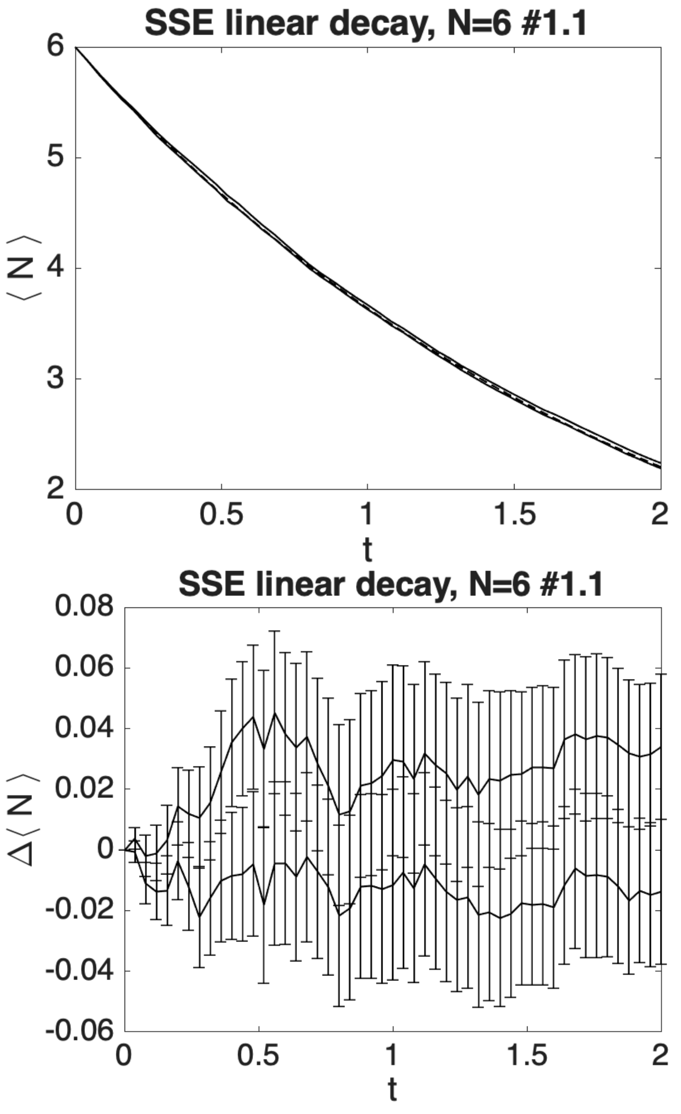


Figure 12.1.: *Example: Linear decay, including a comparison with the exact result, below. The graph shows the sampling error-bars as two parallel lines. The discretization error-bars are less than the minimum, and are not shown.*

12.2. Time-dependent decay, real SSE

This solves a Lindblad master equation for linear time-dependent decay with two modes. and real noises, corresponding to homodyne detection. The initial condition is $\psi_j = \delta_{Nj}$ and $L_1 = a$, for $\boldsymbol{N} = [3,6]$.

The decay rates are:

$$\gamma_1 = [0.5, 1] * t$$

As above, the sparse and functional methods give identical results, but the sparse method is faster. For comparison purposes, the following results are expected:

$$\boldsymbol{N} = [3e^{-t^2/2}, 6e^{-t^2}]$$

Sparse operator method

```
function e = SSElin2spr
%Uses a sparse SSE to solve for a linear two-mode decay
p.name = 'SSE sparse real, N = 3,6';
p.N = [3,6];
p.Om = 1;
p.noises = 4;
p.ranges = 2;
p.nmax = p.N+1;
p.a = Mkbose(p);
p.quantum = 1;
p.sparse = 1;
p.ensembles = [100,1,10];
p.theta{1} = [1,1];
p.gamma{1} = @(p) [0.5,1]*p.t;
p.L{1} = @(m,p) p.a{m};
p.H = @(p) p.Om*(p.a{1}'*p.a{1}+p.a{2}'*p.a{2});
p.initial = @(~,p) Mknumber([3,6],p);
p.expect{1} = @(p) p.a{1}'*p.a{1};
p.expect{2} = @(p) p.a{2}'*p.a{2};
p.compare{1} = @(p) p.N(1)*exp(-p.t.^2/2);
p.compare{2} = @(p) p.N(2)*exp(-p.t.^2);
p.diffplot = {1,1};
p.olabels = {' <n_1 > ','< n_2 > '};
e = xspde(p);
end
```

The use of p.quantum=1 shows that it is a stochastic wave-function problem, while p.sparse=1 indicates sparse matrices, and p.theta = {[1,1]} specifies that all channels have real noises. To use the master equation method, set p.quantum =2 and remove the p.ensembles and p.theta inputs.

Function operator method

```
function e = SSElin2r
%Uses a non-sparse SSE to solve for a linear two-mode decay
p.name = 'SSE, N = 3,6';
p.N = [3,6];
p.Om = 1;
p.ranges = 2;
p.nmax = p.N+1;
p.quantum = 1;
p.ensembles = [100, 10];
p.gamma{1} = @(p)[0.5,1]*p.t;
p.theta{1} = [1,1];
p.L{1} = @A;
p.H = @(psi,p) p.Om*(N(1,psi)+N(2,psi));
p.initial = @(~,p) Mknumber(p.N,p);
p.expect{1} = @(psi,p) N(1,psi);
p.expect{2} = @(psi,p) N(2,psi);
p.compare{1} = @(p) p.N(1)*exp(-p.t.^2/2);
p.compare{2} = @(p) p.N(2)*exp(-p.t.^2);
p.olabels = {'n_1','n_2'};
e = xspde(p);
end
```

With the function method, the function 'Mkbose' is not required. Instead, the effect of the operators is obtained through a function call to the handle '@A'. For large numbers of modes this method uses a reduced amount of memory as there is no stored matrix involved.

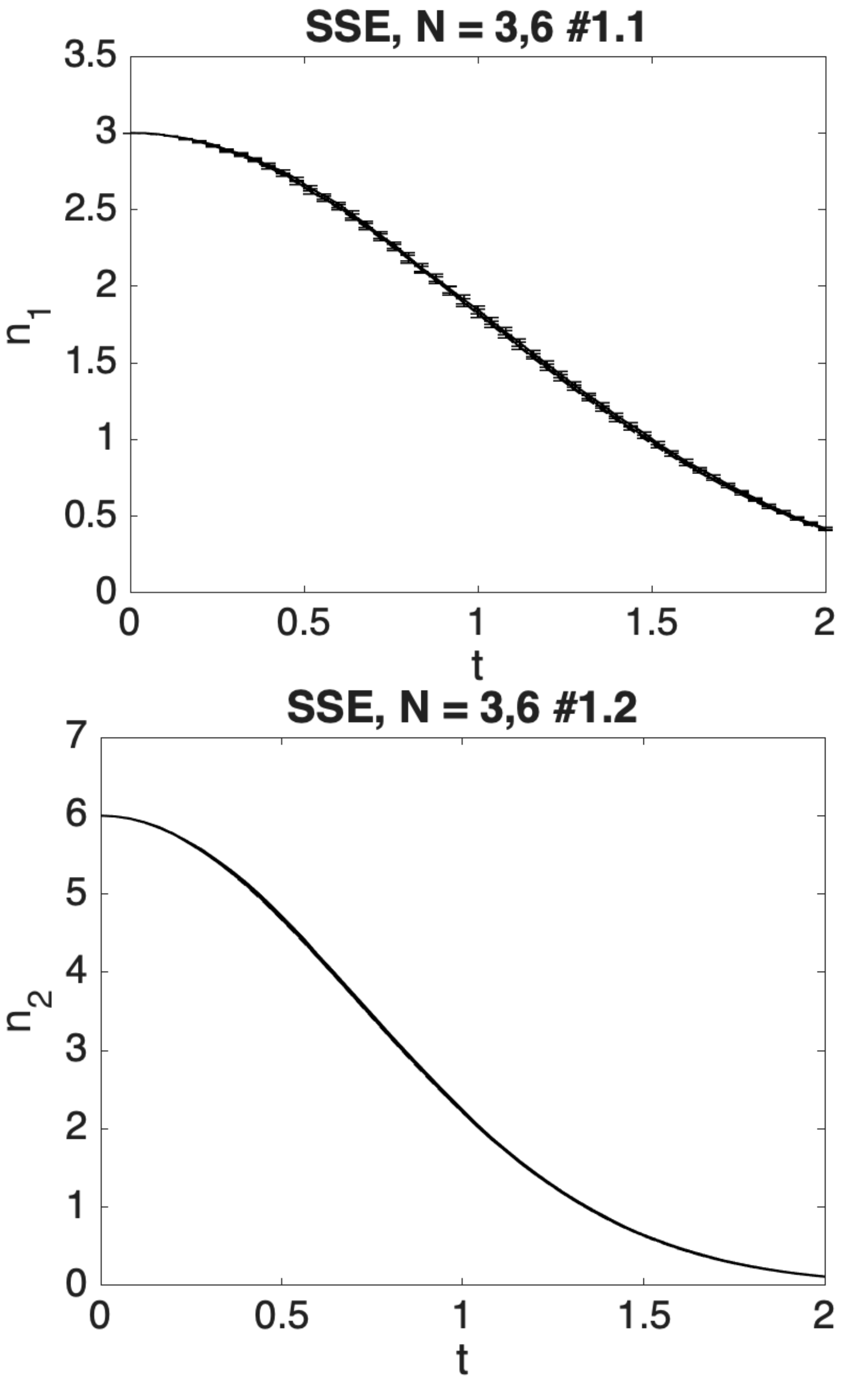


Figure 12.2.: *Example: SSE linear decay, with a time-dependent decay rate.. Top graph has $N = 3$, lower graph has $N = 6$.*

12.3. Nonlinear decay, real SSE

This solves a Lindblad master equation for nonlinear decay with two modes and two decay channels. The initial condition is $\psi_j = \delta_{Nj}$ and $L_1 = a$, $L_2 = a^2$, for $N = [3,6]$.

The decay rates are:

$$\gamma_1 = [0.01, 0.01]$$
$$\gamma_2 = [.5, .25],$$

The function uses the midpoint algorithm with the SSE derivative, and has real noise terms. The sparse and functional methods give identical results, but the sparse method is faster.

Sparse operator method

```
function e = SSEnonlin2spr
%Uses a real sparse SSE to solve for nonlinear two-mode decay
p.name = 'Real sparse SSE, M=2, N=3,6';
p.nmax = [4,7];
p.steps = 8;
p.a = Mkbose(p);
p.a2 = Mkbose(1:2,2,p);
p.ensembles = [10,10,10];
p.quantum = 1;
p.sparse = 1;
p.gamma = {@(p)[0.01,0.01],@(p)[.5,.1]};
p.theta = {[1,1],[1,1]};
p.L = {@(m,p) p.a{m},@(m,p) p.a2{m}};
p.initial = @(~,p) Mknumber([3,6],p);
p.expect{1} = @(p) p.a{1}'*p.a{1};
p.expect{2} = @(p) p.a{2}'*p.a{2};
p.olabels = {'n_1','n_2'};
e = xspde(p);
end
```

With the sparse method, the function 'Mkbose' is used twice to create the operator matrix cell array 'p.a', and 'p.a2' before they are used.

The use of p.quantum=1 shows that it is a stochastic wavefunction problem, p.sparse=1 indicates sparse matrices, and p.theta = {[1,1],[1,1]} specifies that all channels have real noises.

Function operator method

```
function e = SSEnonlin2r
%Uses an SSE to solve for a linear two-mode decay
p.name = 'Real nonlinear SSE, 2-modes, N = 3,6';
p.nmax = [4,7];
p.steps = 8;
p.ensembles = [10,10,10];
p.quantum = 1;
p.gamma = {@(p)[0.01,0.01],@(p) [.5,.1]};
p.L = {@A,@A2};
p.theta = {[1,1],[1,1]};
p.H = @(psi,p) (N(1,psi)+N(2,psi));
p.initial = @(~,p) Mknumber([3,6],p);
p.expect{1} = @(psi,p) N(1,psi);
p.expect{2} = @(psi,p) N(2,psi);
p.olabels = {'n_1','n_2'};
e = xspde(p);
end
```

With the function method, the function 'Mkbose' is not required. Instead, the effect of the operators is obtained through a function call to the handles '@a' and '@a2' . For large numbers of modes this method uses a reduced amount of memory as there is no stored matrix involved in this case.

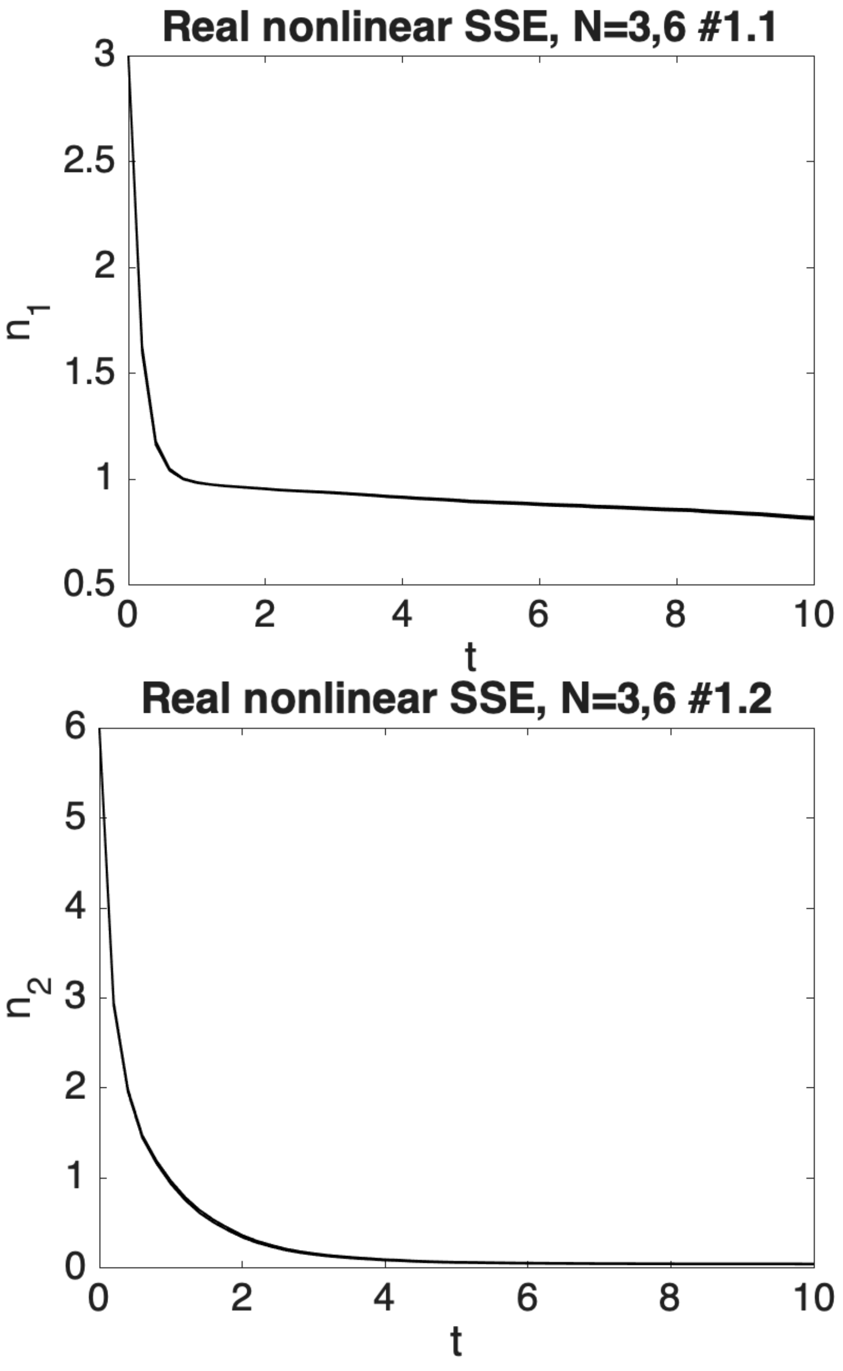


Figure 12.3.: *Example: SSE nonlinear decay, with a small linear decay., real noise and either odd or even number starting points.Top graph has $N = 3$, lower graph has $N = 6$.*

12.4. Two-mode Bose-Hubbard model

This solves the Bose-Hubbard model Hamiltonian for two modes [126]. Damping can be added, but this exampe treats unitary evolution. The initial condition is $\psi_j = \delta_{Nj}$, for $\boldsymbol{N} = [17, 0]$.

The Hamiltonian is:

$$\hat{H} = \frac{1}{2}\Omega\left[\hat{a}_1^\dagger \hat{a}_2 + \hat{a}_2^\dagger \hat{a}_1\right] + \sum_{n=1}^{2} \kappa \left(\hat{a}_n^\dagger \hat{a}_n\right)^2$$

The results below use $\Omega = 1, \kappa = 0.1$. All three methods give identical results, but in this case the functional method is faster.

Sparse operator method

```
function e = BoseHubbard2sp
%Uses a Schroedinger equation with an initial number state
%from Carusotto,Castin,Dalibard,Phys. Rev. A 63, 023606 (2001).
%Includes coupling in a B-H model
p.name = '2-mode coupled B-H, N = 17';
p.N = [17,0];
p.modes = 2;
p.points = 501;
p.order = 4;
p.K = 0.1;
p.O = 1.0;
p.nmax = (sum(p.N)+1)*ones(1,2);
p.a = Mkbose(p);
p.BH = Mkbosehubbard(p.O,p.K,p);
p.quantum = 1;
p.sparse = 1;
p.H = @(p) p.BH;
p.initial = @(~,p) Mknumber(p.N,p);
p.expect{1} = @(p) p.a{1}'*p.a{1}/p.N(1);
p.expect{2} = @(p) p.a{2}'*p.a{2}/p.N(1);
p.olabels = {'< n_1 >/N','< n_2 >/N'};
e = xspde(p);
```

The use of p.quantum=1 shows that it is a stochastic wave-function problem, while p.sparse=1 indicates sparse matrices. To use the master equation method, set p.quantum =2 . This runs approximately 5× slower, due to the use of a density matrix rather than a wavefunction.

Function operator method

```
function e = BoseHubbard2
%Uses a non-sparse functional method
%Includes coupling in a 1D B-H model
%Initial number state, no loss
p.name = '2-mode linear B-H, N = [17,0]';
p.N = [17,0];
p.modes = 2;
p.points = 501;
p.order = 4;
p.K = 0.1;
p.O = 1.0;
p.nmax = (sum(p.N)+1)*ones(1,2);
p.quantum = 1;
p.H = @(psi,p) BoseHubbardH(p.O,p.K,psi,p);
p.initial = @(~,p) Mknumber(p.N,p);
p.expect{1} = @(psi,p) N(1,psi)/p.N(1);
p.expect{2} = @(psi,p) N(2,psi)/p.N(1);
p.olabels = {'< n_1 >/N','< n_2 >/N'};
e = xspde(p);
end
```

With the function method, the function 'Mkbose' is not required. Instead, the effect of the operators is obtained through a function call to the handle '@A', which is part of the support function BoseHubbardH. This typically will run faster than either of the two sparse methods.

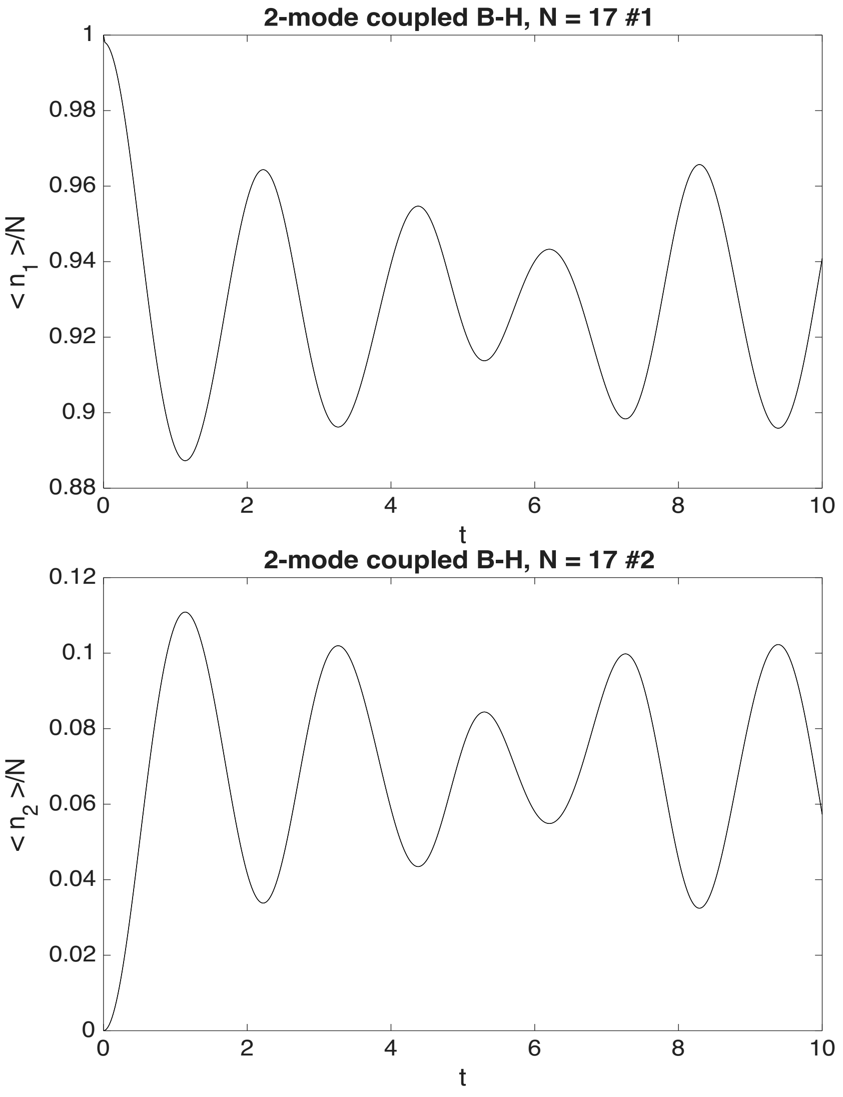


Figure 12.4.: Example: Bose Hubbard Hamiltonian with two modes.

Part VI.

Methods, API and Examples

13. Stochastic methods

This chapter describes the integration methods available, and how to add custom algorithms.

13.1. Introduction to algorithms

Stochastic, partial and ordinary differential equations are central to numerical mathematics. Ordinary differential equations have been known in some form ever since calculus was invented. There are an extraordinary number of algorithms used to solve these equations. This chapter provides an overview of the included algorithms.
xSPDE has six built-in choices of algorithm, with defaults. All built-in methods have an interaction picture and can be used with any space dimension, including $dimensions = 1$, which is an ordinary stochastic equation. All can be used with stochastic or with non-stochastic equations, and with order extrapolation.
For stochastic equations, the Euler method requires an Ito form of stochastic equation, the implicit Euler method requires an implicit Ito form, while the others should be used with the Stratonovich form of calculus. Each is chosen to be able to use an interaction picture to take care of exactly soluble linear terms.

The default methods will solve most DE, SDE, PDE and SPDE problems reliably, but other ones can be included if needed.

13.1.1. Standard methods

The standard xSIM algorithms given below are available for ODEs, PDEs, SDEs and SPDEs. More advanced algorithms for specialized cases are described in section 13.

For stochastic differential equations, which are non-differentiable, the usual rules of calculus do not apply because stochastic noise is non-differentiable. It has fluctuations proportional to $1/\sqrt{dtdV}$, for noise defined on a lattice with temporal cell-size dt and spatial cell-size dV. Hence, the usual differentiability and smoothness properties required to give high-order convergence for standard Runge-Kutta methods are simply not present. Instead, xSPDE has a built-in extrapolation to zero step-size for high-order stochastic convergence.

Many more complex higher order algorithms for stochastic integration exist but are not included in the current xSPDE distribution, and users are encouraged to contribute their favorite methods.

We note here that there are multiple error sources possible. SDE/SPDE errors are often dominated by the sampling error, not discretization. In addition, all convergence theorems only apply to the limit of zero step-size. One may be very far from this regime in a given practical calculation. Analytic error estimates also have pre-factors which are hard to calculate. However, xSPDE can numerically estimate both the discretization and sampling error for any given average observable.

13.1.2. Advanced methods

Three more advanced method libraries are included here, namely weighted, projected and forward-backward stochastic differential equations. If you have a favorite algorithm that is not included, user-defined algorithms and libraries can be added. The existing methods are listed below, and the corresponding .m-files can be used as a model.

Define the routine, for example ”myalgorithm.m”, set $p.method = @myalgorithm$, then adjust the input value of ipsteps and order if these need be changed to a new value. The interaction-picture transform, prop, can also be changed if the built-in choice is not sufficient.The xSPDE algorithms available currently treat

- ordinary (and partial) differential equations
- stochastic differential equations
- stochastic partial differential equations
- weighted stochastic differential equations
- projected stochastic differential equations,
- forward-backward stochastic differential equations

Some of the more advanced features of the libraries require additional input parameters. In particular:

backfields is used for forward-backward stochastic equations, describing backward time components. These are described in the Forward-backward section. Note that **fields** is still used, and it gives the *total* number of forward+backward fields.

auxfields gives the number of auxiliary fields. These have a functional definition (defines) that includes both a field and noise variable, as needed for spectral observables. Cell index numbers i greater than the maximum field cells access the auxiliary fields in the observe function.

13.2. General differential form

The general equation treated is given in differential form as

$$\frac{\partial \boldsymbol{a}}{\partial t} = \boldsymbol{A}\left[\boldsymbol{\nabla}, \boldsymbol{a}, t\right] + \underline{\mathbf{B}}\left[\boldsymbol{\nabla}, \boldsymbol{a}, t\right] \cdot \boldsymbol{\zeta}(t) + \underline{\mathbf{L}}\left[\boldsymbol{\nabla}\right] \cdot \boldsymbol{a}. \tag{13.1}$$

It is convenient for the purposes of describing interaction picture methods, to introduce an abbreviated notation as:

$$\mathcal{D}\left[\mathbf{a}, t\right] = \boldsymbol{A}\left[\boldsymbol{a}, t\right] + \underline{\mathbf{B}}\left[\boldsymbol{a}, t\right] \cdot \boldsymbol{\zeta}(t). \tag{13.2}$$

Hence, we can rewrite the differential equation in the form:

$$\frac{\partial \boldsymbol{a}}{\partial t} = \mathcal{D}\left[\mathbf{a}, t\right] + \underline{\mathbf{L}}\left[\boldsymbol{\nabla}\right] \cdot \boldsymbol{a}. \tag{13.3}$$

13.2.1. Linear propagator

Next, we define a linear propagator. This is given formally by:

$$\mathcal{P}\left(\Delta t\right) = \exp\left(\Delta t \underline{\mathbf{L}}\left[\boldsymbol{\nabla}\right]\right). \tag{13.4}$$

Typically, but not necessarily, this is evaluated in Fourier space, where it is a diagonal term in the momentum vector conjugate to the transverse space coordinate. It involves a Fourier transform, multiplication by a function of momentum, and an inverse Fourier transform. For simplicity, the stochastic noise is assumed constant throughout the interval dt. The reader is referred to the literature for more details.

It is simple to add your own algorithm if you prefer a different one. Note that if they use an interaction picture, then ipsteps must be given explicitly to specify the interaction picture duration, where ipsteps gives the number of sequential propagator steps in time required for the method.

13.3. Standard methods

The standard methods are listed below. All except the last can be used with any equation: ODE, SDE, PDE or SPDE, either with or without a linear interaction picture term. These can all treat cell arrays of vectors, as well as cell arrays of tensors in the ODE and SDE cases.

The basic equation used, where $\tilde{t}$ is the interaction picture origin, typically but not always at the midpoint, is

$$\frac{\partial \tilde{\mathbf{a}}}{\partial t} = \mathcal{P}^{-1}\left(t, \tilde{t}\right) \mathcal{D}\left[\mathcal{P}\left(t, \tilde{t}\right) \tilde{\mathbf{a}}, t\right] \equiv \tilde{\mathcal{D}}\left[\tilde{\mathbf{a}}, t\right].$$

If there are multiple cells of fields and noises, all the individual fields and noises are passed in the user defined functions like *deriv.*

13.3.1. Euler: Ito-Euler

This is an explicit Ito-Euler method using an interaction picture. While traditional, it is not generally recommended. If it is used, very small step-sizes will generally be necessary to reduce errors to a usable level. This is because it is is only convergent to first order deterministically and tends to have large errors.

It is designed for use with an Ito form of stochastic equation. It requires one IP transform per step (*p.ipsteps* = 1). Choosing the origin of the interaction picture at $\tilde{t} = t_n$, one has $\mathbf{a}_n \equiv \tilde{\mathbf{a}}_n$, so:

$$\Delta\tilde{\mathbf{a}}_{n+1} = \tilde{\mathbf{a}}_{n+1} - \tilde{\mathbf{a}}_n = \Delta t\mathcal{D}\left[\mathbf{a}_n, t_n\right]$$

To get the next time point at $t = t_{n+1} = t_n + \Delta t$, one calculates:

$$\begin{aligned}\Delta\tilde{\mathbf{a}}_{n+1} &= \Delta t\mathcal{D}\left[\mathbf{a}_n, t_n\right] \\ \mathbf{a}_{n+1} &= \mathcal{P}\left(\Delta t\right)\cdot\left[\mathbf{a}_n + \Delta\tilde{\mathbf{a}}_{n+1}\right]\end{aligned} \tag{13.5}$$

13.3.2. Implicit: implicit Ito-Euler

This is a fully implicit Ito-Euler method using an interaction picture. It is more robust, though slower, than the explicit form. If it is used, very small step-sizes will generally be necessary to reduce errors to a usable level.

This is because it is is only convergent to first order, and therefore tends to have large errors. It is designed for use with an implicit Ito form of stochastic equation. Note that this implies double the usual Stratonovich correction!

It requires one IP transform per step (*p.ipsteps* = 1). Choosing the origin of the interaction picture at $\tilde{t} = t_{n+1}$, one has $\mathbf{a}_{n+1} \equiv \tilde{\mathbf{a}}_{n+1}$, so:

$$\Delta\tilde{\mathbf{a}}_{n+1} = \tilde{\mathbf{a}}_{n+1} - \tilde{\mathbf{a}}_n = \Delta t\mathcal{D}\left[\mathbf{a}_{n+1}, t_n\right]$$

Starting from time $t = t_n$, to get the next time point at $t = t_{n+1} = t_n + \Delta t$, one calculates, using iteration to get the implicit result of the next time-point:

$$\begin{aligned}\bar{\mathbf{a}}^{(0)} &= \mathcal{P}\left(t_{n+1}, t_n\right)\cdot\left[\mathbf{a}_n\right] \\ \bar{\mathbf{a}}^{(i)} &= \bar{\mathbf{a}}^{(0)} + \Delta t\mathcal{D}\left[\bar{\mathbf{a}}^{(i-1)}, t_{n+1}\right] \\ \mathbf{a}_{n+1} &= \tilde{\mathbf{a}}_{n+1} = \bar{\mathbf{a}}^{(iter)}\end{aligned} \tag{13.6}$$

Here the result of $\bar{\mathbf{a}}^{(iter)}$ is obtained after a fixed number of iterations of $\bar{\mathbf{a}}^{(i)}$.

13.3.3. MP: Midpoint

This is a semi-implicit midpoint method using an interaction picture. It gives good results for stochastic and stochastic partial differential equations. It is convergent to second order in time for deterministic equations and for stochastic equations with commuting noise. It is strongly convergent and robust. It requires two half-length IP transforms per step (*p.ipsteps* = 2), with an interaction picture origin at $\tilde{t} = t_n + \Delta t/2$.

To get the next time point, one calculates a midpoint derivative iteratively at time to get the next time point at $\bar{t}_n = t_{n+1/2} = t_n + \Delta t/2$, to give an estimated midpoint field $\bar{\mathbf{a}}^{(i)}$, usually with four iterations. The number of iterations can be changed:

$$\begin{aligned}\bar{\mathbf{a}}^{(0)} &= \mathcal{P}\left(t_{n+1/2}, t_n\right)\cdot\left[\mathbf{a}_n\right] \\ \bar{\mathbf{a}}^{(i)} &= \bar{\mathbf{a}}^{(0)} + \frac{\Delta t}{2}\mathcal{D}\left[\bar{\mathbf{a}}^{(i-1)}, \bar{t}_n\right] \\ \mathbf{a}_{n+1} &= \mathcal{P}\left(t_{n+1}, t_{n+1/2}\right)\cdot\left[2\bar{\mathbf{a}}^{(iter)} - \bar{\mathbf{a}}^{(0)}\right]\end{aligned} \tag{13.7}$$

This is the default method for stochastic cases.

13.3.4. MPadapt: adaptive midpoint

This is an implicit midpoint method using an interaction picture, together with an adaptive technique for integrating highly nonlinear equations. At low amplitudes it is identical to the standard midpoint method. For amplitudes $|a_i|^2$ above a critical value, p.adapt, the amplitude is inverted and propagated using the differential equation for its inverse.

Initially a switch p is set to 1 for low amplitudes, and -1 for high amplitudes. To get the next time point, one calculates a midpoint derivative iteratively at time to get the next time point at $\bar{t}_n = t_{n+1/2} = t_n + \Delta t/2$, to give an estimated midpoint field $\bar{\mathbf{a}}^{(i)}$, as above, but with the derivative modified to give the derivative of a_i^p:

$$\begin{aligned}
\bar{\mathbf{a}}^{(0)} &= \mathcal{P}\left(t_{n+1/2}, t_n\right) \cdot [\mathbf{a}_n] \\
\tilde{\mathbf{a}}^{(0)} &= \mathbf{a}_n^p \\
\tilde{\mathbf{a}}^{(i)} &= \tilde{\mathbf{a}}^{(0)} + \frac{\Delta t}{2} p \left[\tilde{\mathbf{a}}^{(i-1)}\right]^{1-p} \left(\mathcal{D}\left[[\tilde{\mathbf{a}}^{(i-1)}]^p, \bar{t}_n\right]\right) \\
\mathbf{a}_{n+1} &= \mathcal{P}\left(t_{n+1}, t_{n+1/2}\right) \cdot \left[2\tilde{\mathbf{a}}^{(iter)} - \tilde{\mathbf{a}}^{(0)}\right]^p
\end{aligned} \tag{13.8}$$

13.3.5. RK2: second order Runge-Kutta

This is a second order Runge-Kutta method using an interaction picture. It is convergent to second order in time for non-stochastic equations, and for stochastic equations with additive noise, but otherwise it is first order. It often has higher errors than midpoint methods. It requires one IP transform per step (*p.ipsteps* = 1), with $\tilde{t} = t_n$ The basic RK2 method is defined by:

$$\begin{aligned}
\tilde{\mathbf{a}}^{(1)} &= \tilde{\mathbf{a}}_n + \Delta t \tilde{\mathcal{D}}\left[\tilde{\mathbf{a}}_n, t_n\right] \\
\tilde{\mathbf{a}}^{(2)} &= \tilde{\mathbf{a}}_n + \Delta t \tilde{\mathcal{D}}\left[\tilde{\mathbf{a}}^{(1)}, t_{n+1}\right] \\
\tilde{\mathbf{a}}_{n+1} &= \left(\tilde{\mathbf{a}}^{(1)} + \tilde{\mathbf{a}}^{(2)}\right)/2
\end{aligned}$$

Including the interaction picture transforms, based at $\tilde{t} = t_{n+1}$, one calculates:

$$\begin{aligned}
\tilde{\mathbf{a}}^{(0)} &= \mathcal{P}\left(t_{n+1}, t_n\right) \mathbf{a}_n \\
\tilde{\mathbf{a}}^{(1)} &= \mathcal{P}\left(t_{n+1}, t_n\right) \left(\mathbf{a}_n + \Delta t \cdot \mathcal{D}\left[\mathbf{a}_n, t_n\right]\right) \\
\tilde{\mathbf{a}}^{(2)} &= \tilde{\mathbf{a}}^{(0)} + \Delta t \mathcal{D}\left[\tilde{\mathbf{a}}^{(1)}, t_{n+1}\right] \\
\mathbf{a}_{n+1} &= \left(\tilde{\mathbf{a}}^{(1)} + \tilde{\mathbf{a}}^{(2)}\right)/2.
\end{aligned} \tag{13.9}$$

13.3.6. RK4: fourth order Runge-Kutta

This is a fourth order Runge-Kutta method using an interaction picture. It is convergent to fourth order in time for non-stochastic equations, but for stochastic equations it can be more slowly convergent than the midpoint method. It requires four half-length IP transforms per step (ipsteps = 2). To get the next time point, one calculates four derivatives sequentially:

$$\begin{aligned}
\bar{\mathbf{a}} &= \mathcal{P}\left(\frac{\Delta t}{2}\right)\cdot[\mathbf{a}_n] \\
\mathbf{d}^{(1)} &= \frac{\Delta t}{2}\mathcal{P}\left(\frac{\Delta t}{2}\right)\cdot\mathcal{D}\left[\mathbf{a}_n, t_n\right] \\
\mathbf{d}^{(2)} &= \frac{\Delta t}{2}\mathcal{D}\left[\bar{\mathbf{a}}+\mathbf{d}^{(1)}, t_{n+1/2}\right] \\
\mathbf{d}^{(3)} &= \frac{\Delta t}{2}\mathcal{D}\left[\bar{\mathbf{a}}+\mathbf{d}^{(2)}, t_{n+1/2}\right] \\
\mathbf{d}^{(4)} &= \frac{\Delta t}{2}\mathcal{D}\left[\mathcal{P}\left(\frac{\Delta t}{2}\right)\left[\bar{\mathbf{a}}+2\mathbf{d}^{(3)}, t_{n+1}\right]\right] \\
\mathbf{a}_{n+1} &= \mathcal{P}\left(\frac{\Delta t}{2}\right)\cdot\left[\bar{\mathbf{a}}+\left(\mathbf{d}^{(1)}+2\left(\mathbf{d}^{(2)}+\mathbf{d}^{(3)}\right)\right)/3\right]+\mathbf{d}^{(4)}/3
\end{aligned} \tag{13.10}$$

This might seem the obvious choice, having the highest order. However, it can converge at a range of apparent rates, depending on the relative importance of stochastic and non-stochastic terms. Due to its use of differentiability, it may converge more slowly than the midpoint method with stochastic terms present. It is the default for ODE and PDE cases.

13.3.7. RKWP21: second-order weak Runge-Kutta method

This propagates an Ito step with a 2nd-order, single variable weak Runge-Kutta method. It treats a single field and noise [29]. Because it requires separate functions for drift and noise, it uses derivA and derivB functions for the A and B functions. These functions are defined in the Ito calculus, where the noise term w is delta-correlated. For full details, see the reference.

14. Errors

This chapter describes the estimation and control of integration errors.

14.1. Time-step discretization errors

To check convergence, xSPDE default settings will repeat the calculations twice for checking time-steps, and many times more in stochastic cases to estimate sampling errors. Since the checks make xSPDE slower, they can be turned off, but then there are no error-estimate. Whatever the application, error-estimates useful, and generally should be used.

If the errors are too large relative to the application, you should decrease the time-steps or increase the number of samples. Which is needed depends on the type of error.

Errors caused by the finite time-domain step-size are checked automatically, since $p.checks(1) = 1$ is the default option. If $p.checks = 0$ is used, there is no time-domain error check.

Errors due to a finite step-size are estimated by running a check simulation with half the initial step-size and the same random sequence, extrapolating to zero step-size if $order > 0$ is specified. The program returns an error bound as the difference of the two most accurate results. Any 2D output graph plots error-bars if $checks = 1$ was specified, provided they are large enough to plot.

RMS output error summaries of all outputs are in the text outputs. These are normalized by the maximum value of each output. If p.verbose=1 is specified, xSPDE will give both the average RMS error of each output, and the maximum error of each output. Individual time-step error bounds, $e\left(o\right)$ are given in the output data, and the plots give $\bar{o} \pm e\left(o\right)$.

Error-bars below a minimum relative size compared to the vertical range of the plot, specified by the graphics variable $minbar$, are not plotted. The default for this is $minbar = 0.01$. All error bars are calculated individually for each type of data average. Minbar is a cell array that can can be set for each type of average or graph. If the cell argument is omitted, it applies globally. Error estimates are also given for functional transforms of averages.

If the errors are too large, one can either increase the points, which gives more plotted points and lower errors, or increase the steps, which reduces the step size without changing the data resolution. The default algorithm and extrapolation order can also be changed. Error bars on graphs can be removed either by setting $checks = 0$ or increasing $minbar$.

Discretization errors caused by the finite spatial lattice are not currently checked in the xSIM code. They must be checked by comparing results with different transverse lattice ranges and step-size. Similarly, errors from discrete probability bin sizes are not checked.

If computed, the discretization error is included in the graphical data outputs for all observables. It is accessed by setting the last index for the output data equal 2. The raw discretization error is a very cautious estimate, and may overestimate the errors. This estimate can be improved using extrapolation, explained next.

14.1.1. Extrapolation

xSPDE can use extrapolation to improve convergence, which requires input of the method order. If this is non-zero, and checks are set to 1 to allow successive integration with different step-sizes, the output of all data graphed will be extrapolated by assuming the method has the specified order. To implement extrapolation and obtain a less conservative mean and error-bar result, set $p.order > 0$. This user-defined value has a default of $p.order = 0$.

Extrapolation is valuable for improving the accuracy of a differential equation solver. It is valid for small time-steps. Suppose an algorithm has a correct solution R_0, but returns a numerical result R with an error order n. For small step-size, integration results $R(dt)$ with step-size dt have an error of order dt^n, that is:

$$R(dt) = R_0 + e(R) = R_0 + k.dt^n. \tag{14.1}$$

Hence, from two results at different values of dt, differing by a factor of 2, one would obtain

$$\begin{aligned} R_1 &= R(dt) = R_0 + k.dt^n \\ R_2 &= R(2dt) = R_0 + 2^n k.dt^n. \end{aligned} \tag{14.2}$$

The true result, extrapolated to the small-step size limit, is obtained by giving more weight to the fine step-size result, while subtracting from this a correction due to the coarse step-size calculation, to cancel the leading error term:

$$R_0 = \frac{[R_1 - R_2 2^{-n}]}{[1-2^{-n}]}. \tag{14.3}$$

Thus, if we define a factor ϵ as

$$\epsilon(n) = \frac{1}{[2^n - 1]} = \left(1, \frac{1}{3}, \frac{1}{7} \dots\right), \tag{14.4}$$

the true results are obtained from extrapolation to zero step-size as:

$$R_0 = (1+\epsilon) R_1 - \epsilon R_2. \tag{14.5}$$

The built-in algorithms have an order as ordinary differential equation integrators of 1, 1, 2, 2, 2, 4 respectively and will converge to this order at small step-sizes. Weak first order convergence is always obtainable for these single noise-step SDE methods [31]. Second order weak convergence is obtained in some cases with RK4 algorithms.

Higher order convergence for the raw data is not guaranteed for the built-in SDE algorithms. The algorithms used do **not** always converge to the standard ODE order when used for stochastic equations. Hence extrapolation to higher than first order should be used with caution in stochastic calculations, unless more complex methods are used [29].

14.1.2. Extrapolated error-bars

If extrapolation is used, the error bar half-size is the difference of the best raw estimate and the extrapolation. Extrapolated results are usually inside those given by the error-bars, however, note that:

- **extrapolation with too high an order may under-estimate error bars**
- **extrapolation with too low an order reduces the accuracy**

A conservative order estimate of $order = 1$ can be used for all SDE and SPDE cases, although there are higher order methods available. This gives an extrapolated weak order of 2 for stochastic cases. One can set order = 0, which gives the default with no extrapolation, or use a higher order if preferred. As explained above, this requires caution. For an ODE or PDE the deterministic order should be used. For the default RK4 deterministic method, $order = 4$ can be used. All orders are improved by one with extrapolation.

High-order convergence without extrapolation can also be obtained, either in special cases using the xSPDE methods, or by adding user-specified techniques. The xSPDE libraries can be readily extended by the user to include these, through defining a modified method function appropriately.

14.2. Iteration errors

For forward-backward solutions that require an overall iteration, the iteration error for each is reported in the same format as the step-size errors. These are calculated as the difference between the two most accurate iterations, in a similar way to the step errors. The resulting error-bars that are graphed will combine both the step-size and iteration errors.

14.3. Statistical errors

Sampling error estimation in xSIM uses three different techniques.

- xSIM uses sub-ensemble averaging, requiring high-level ensembles.
- For probability estimates, a Poissonian sampling error is used, based on counts.

- If there is a comparison probability, this is used for sampling error estimates.

This procedure leads to reliable sampling error estimates, and makes efficient use of the vector instruction sets used by Matlab. Ensembles are specified in three levels. The first, ensembles(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 observables. The two lines generated represent $\bar{o} \pm \sigma\left(o\right)$, where o is the observe function output, and σ is the standard deviation in the mean.

Here, ensembles(2) specifies ensembles computed in series. The highest level ensemble, ensembles(3), is used for parallel simulations. This is faster for a multiple core CPU or when the codes are run in a supercomputing environment, which requires the Matlab parallel toolbox. Either type of high-level ensemble, or both together, can be used to calculate sampling errors.

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 $ensembles(2)$, which 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, the second by time, the third by the number of cores. Sampling error control helps ensures accuracy.

14.3.1. Sampling error

Quantitative sampling error estimation in xSPDE uses sub-ensemble averaging. Ensembles are specified in three levels, using vector, serial and parallel methods, respectively. The vector ensemble length, *p.ensembles(1)*, is called the number of samples for brevity. All quantities returned by the observe functions are averaged over the samples, which are calculated efficiently using a vector of trajectories.

By the central limit theorem, the sample averages are distributed as a normal distribution at large sample number. Next, the sample averages are averaged over the two higher level ensembles, if specified. The variance of this data is used to estimate a standard deviation in the mean, since each is normally distributed.

The next level, p.ensembles(2), is for serial calculations of ensembles. The highest level ensemble, p.ensembles(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.

The statistical error due to finite samples of trajectories is called the sampling error. The RMS value of the relative sampling error for each computed function, normalized by the maximum modulus of the observable, is printed out after each xSPDE simulation. If the expected comparison value is zero, the absolute value is given.

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

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

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

Two lines are graphed for an upper and lower standard deviation departure from the mean. This is only plotted if the total number of serial or parallel ensembles is greater than one, preferably at least 10–20 to give reliable estimates. The sampling error is reasonably accurate, but may underestimate errors for observe function results that have highly non-Gaussian trajectory distributions, especially with asymmetries. These estimates are available for all observables in any dimension. The two lines generated in the graphs represent $\bar{o} \pm \sigma$, where o is the mean output, and σ is the computed standard deviation in the mean.

14.3.2. Comparisons: compare

Every observe function can be accompanied by a comparison function, with a function handle *compare*$\{n\}$. This generates a vector of analytic solutions or experimental datapoints which is compared to the average of the stochastic results. Results are plotted as additional lines on the two-dimensional graphical outputs, and a summary of comparison differences is printed.

A cell array of functions is used to obtain comparison results. These are calculated from the user-specified **compare{n}(p)** handle where the function argument is the parameter structure p, giving a extra dashed line on the two-dimensional graphs. Other graphics options are available as well. These optional comparisons can be input in all dimensions. When there are error estimates, a chi-squared test is carried out to determine if the difference is within the expected step-size and sampling error bars. If the comparison has errors, for example from experimental data, the chi-squared test will include the experimental errors.

14.3.3. Convergence: xcheck

The convergence checker, xcheck(checks,p), is designed for use where there are analytic results available for comparisons. This will automatically run xSIM a total of checks times, increasing the initial steps by 2 after each run, to reduce the step-size by 2. It then runs xGRAPH to display the most accurate result. It prints the time-step, the maximum difference with an input compare and the estimated errors found at the relevant point.

Exercise

- **Simulate the Kubo oscillator using the file, *Kubocheck.m*, with xcheck.**

```
function [e] = Kubocheck()
p.name = 'Kubo with convergence checks';
p.ensembles = [1000,10];
p.initial = @(w,p) 1;
p.range = 2;
p.deriv = @(a,xi,p) 1i*xi.*a;
p.observe{1} = @(a,p) real(a(1,:));
p.observe{2} = @(a,p) a(1,:).*conj(a(1,:));
p.olabels = {'<a> ','< a^2> '};
p.xlabels = {'\tau'};
p.compare{1} = @(p) exp(-p.t/2);
p.compare{2} = @(p) 1;
e = xcheck(2,p);
end
```

14.4. Chi-squared estimates

Chi-squared error estimates are reported in cases that have statistical sampling errors and comparison functions. These allow estimates of goodness of fit for probabilities. For N_p independent points graphed or measured, if O_i is an observable with measured mean $\bar{O}_i$ and statistical fluctuations ΔO_i, one has that:

$$\chi^2/N_p = \frac{1}{N_p}\sum_i \frac{\left\langle \left[(\bar{O}_i + \Delta O_i) - O_i^a\right]^2 \right\rangle}{\sigma_i^2} \tag{14.6}$$

Here σ_i^2 is an estimated variance. Provided that $\left\langle \Delta O_i^2 \right\rangle = \sigma_i^2$ and $\bar{O}_i = O_i^a$, one should obtain the expected result of $\chi^2/N_p \approx 1$. The exact distribution is known in special cases, but this requires that all data is independent and has a Gaussian distribution, which is not the case for stochastic trajectories.

Because of the lack of independence from point to point, these error sums are not identical to Pearson's original definition of χ^2, and therefore should be used with caution. Nevertheless, the definition provides a way of evaluating goodness of fit that is useful.

The value of σ_i^2 is obtained by including all known statistical error sources, so

$$\sigma_i^2 = \sum_{n=1}^{2} \left(\sigma_i^{(n)}\right)^2 . \tag{14.7}$$

where:

1. If higher ensembles are used, the estimated σ_i^2 includes numerical sampling errors.
2. If comparisons have known statistical errors, these are included as well.

14.4.1. Probability comparisons

Comparisons of trajectory probabilities and analytic probabilities do not always result in perfect agreement. This is because the limitations of memory and simulation time mean that trajectories have to be binned, which leads to an additional discretization error. Note that xSPDE approximates the comparison analytic probability of a bin by the central bin value of the probability, which is the simplest procedure.

To explain this, comparisons of probabilities ought to use the average probability density over the bin, which is different from the central value. Suppose one has a comparison distribution $p^a(x)$. Using Simpson's rule, the average analytic probability density integrated over a bin size Δx is approximately:

$$\begin{aligned} p_o^a &= \frac{1}{\Delta x}\int_{x_0-\Delta x/2}^{x_0+\Delta x/2} p^a(x)dx \qquad (14.8)\\ &\approx \frac{1}{6}\left[4p^a(x_0) + p^a\left(x_0+\frac{\Delta x}{2}\right) + p^a\left(x_0-\frac{\Delta x}{2}\right).\right] \end{aligned}$$

This is equivalent to a cubic polynomial fit. It can be used to improve the analytic binning comparisons. It is especially important for multi-dimensional comparisons. It results in 9 distinct terms for two dimensions. This correction should be inserted manually in the comparison functions.

14.4.2. Scaling of χ^2 errors

Because chi-squared probability tests are sensitive, it helps to understand how they scale with bin-size. With N_s total samples, the estimated probability P_i in a bin with probability density $p(\mathbf{a})$ and sampled counts of N_i is given by $P_i = N_i/N_s = p_i A$ for a bin b_i with area A, where:

$$p_i = \frac{1}{A}\int_{b_i} p(\mathbf{a})\, dA \tag{14.9}$$

The Poissonian variance of the counts in the bin is $\langle \Delta N_i \rangle = \langle N_i \rangle$. The expected probability variance is therefore

$$\left\langle \Delta P^2 \right\rangle = \left\langle \Delta N_i^2 / N_s^2 \right\rangle = \langle N_i \rangle / N_s^2. \tag{14.10}$$

Let $\langle N_i \rangle = N_i^a$, the analytic or expected count number. The expected probability density variance at a point is therefore

$$\left\langle \Delta p_i^2 \right\rangle = \left\langle \Delta N_i^2 / A^2 N_s^2 \right\rangle = N_i^a / A^2 N_s^2 = p_i^a / A N_s. \tag{14.11}$$

Here p_i^a is the analytic or comparison probability density, and $\left\langle \Delta p_i^2 \right\rangle^a = p_i^a / A N_s$ is the expected analytic variance. The χ^2 variable, that follows the Pearson χ^2 distribution, is defined as follows:

$$\chi^2/N_p = \frac{1}{N_p}\sum_i \frac{\left\langle [p_i - p_i^a]^2 \right\rangle}{\left\langle \Delta p_i^2 \right\rangle} \tag{14.12}$$

Here, p_i^a is obtained by integrating over the i-th probability bin. It can be estimated by using the central value, $p_i^a \approx p(\mathbf{a}_i)$, although cubic interpolation is more precise.

This could lead to a fixed error in the analytic probability density p_i^a, so $p_i^a \to p_i^a + \epsilon_i$, possibly localized to some fraction of bins f which may change with the bin size. Suppose, for simplicity, that ϵ is due to an integration error in integrating the exact distribution or any other error in the 'exact' distribution, and it does not change with changes to the bin area A.

From the definition of χ^2, if the generated samples have negligible step-size errors:

$$\chi^2/N_p = \frac{1}{N_p}\sum_i \frac{\left\langle [(p_i^a + \Delta p_i) - p_i^a - \epsilon_i]^2 \right\rangle}{\left\langle \Delta p_i^2 \right\rangle} \tag{14.13}$$

For simplicity, if we consider the large sample limit with uniform probabilities,

$$\chi^2/N_p = 1 + \frac{f\epsilon^2}{\langle \Delta p^2 \rangle} = 1 + \frac{f\epsilon^2 A N_s}{p^a} \tag{14.14}$$

Increasing the bin area A will increase χ^2/N_p above its usual value of 1 by an amount proportional to A. This is simply because smaller bins have less intrinsic accuracy, due to a larger sampling error. As a result, it is often preferable to use more accurate probability estimates with larger bins having more counts, since these are much more sensitive to effects like this.

Often, simulated and comparison graphs may appear identical visually, but even if they have small errors they may still be very significant. Such comparison binning errors can be reduced by using cubic spline interpolations, as explained above.

14.5. Error outputs

There are six types of data outputs: data, step errors, sampling errors, comparisons, comparison systematic errors, and comparison random errors. Summaries of this will appear in the printed outputs, with greater details if p.verbose>0 is chosen. Step errors and sampling errors, as well as comparison data are stored in the output data arrays.

14.5.1. Numerical error outputs

The last data index c is used to obtain errors and comparisons in data outputs. To obtain comparison data, a comparison function is defined for each output function. This can include, for example, experimental data, experimental errors or exact analytic comparisons where they are available.

1. Means are in $c = 1$ data, except if scatters>1, which gives individual trajectories.
2. If checks>0, all the step errors are in $c = 2$ data.
3. If $ensembles(2, 3) > 1$, the sampling errors are in $c = 3$ data.
4. Comparison values from compare functions are in $c = 4$ data.
5. Comparison systematic errors can be included in $c = 5$ data.
6. Comparison statistical errors can be included in $c = 6$ data.

14.5.2. Graphical error outputs

These are explained in detail in the xGRAPH reference section.

1. Mean values or trajectories are graphed as separate data lines.
2. Step errors generate graph error bars
3. Sampling errors are graphed as parallel solid lines
4. Dashed lines indicate comparison values from compare functions.
5. Comparison systematic errors give additional error bars
6. Comparison statistical errors can be included as parallel lines

Because multiple errors can generate very complex graphs, there is additional control of error bar generation, explained in the xGRAPH reference section. One can also obtain difference graphs with comparisons, which allow errors to be examined more closely, and error bars can be combined in different ways.

Graphics data is only available for two-dimensional graphs, and is subject to selection using the axes inputs.

14.5.3. Printed error outputs

Printed error summaries are generated for each data output if p.verbose > 0. The defaults are root mean square (RMS) and maximum errors, all normalized. Normalization is by the modulus of the largest data value in a given output data set, including all lines. If available, the largest comparison values is used. If it is zero or p.relerr $= 0$, then no normalization is carried out.

After computing RMS values for each output dataset, again including all lines and grid-points, a second RMS average is taken over all the outputs, weighting each total equally, and including all functions and sequence datasets where there are nonzero errors. Data with no errors, below a tolerance of 10^{-10}, are not included in the mean RMS total errors for each category.

There is a final RMS average taken over the step, sampling and comparison totals. This ignores categories with no errors. This printout occurs even with verbose $= 0$, to allow a rapid comparison in case there are unexpected errors, which might require a new simulation with more steps or random trajectories.

Printed errors are summarized in three main categories

1. Discretization or step errors
2. Sampling errors
3. Comparison or difference errors

Comparison data may not be available over an entire lattice. If this is the case, the axes point selections can be used to restrict the relevant datas points used for these comparisons. This also applies to the goodness of fit and error-vector outputs, since they make use of comparison data where it is available.

14.5.4. Goodness of fit (χ^2)

The χ^2 statistics are obtained by normalizing the comparison squared differences by the sum of squares of all the data and comparison errors at that point. These are summed over every data point with relevant data, and the number of relevant data points, k, is stored. The ratio of χ^2/k should be order 1 for statistical errors.

These are summarized for each functional data output type, as well as giving rise to an error total.

14.5.5. Error vector output

When used as a function call in batch mode, the first type of data returned by xSIM is a six-component error vector. This can be used for summarizing error data in a batch job, to determine if a specified error-threshold is reached, to allow an iterative increase in the number of time-steps or trajectories.

The error-vector components are all RMS averages:

1. Total error overall, including step, discretization and comparisons

2. Total step-size error
3. Total sampling error
4. Total comparison error
5. Total χ^2/k goodness of fit
6. Simulation elapsed time

14.5.6. Error summaries

There are six types of data outputs: data, errors, comparisons and comparison errors. Summaries will appear in the printed outputs, depending on the verbosity setting. Step errors and sampling errors, as well as comparison data are stored in output data arrays. These are also available graphically in two-dimensional graphs.

15. Simulation parameters and extensibility

This chapter gives a reference guide to the xSPDE simulation parameters and functions.

15.1. Overview: high-level xSPDE functions

Simulations carried out by xSPDE are performed by xSIM, then graphed by xGRAPH. Input parameters come from an **input** sequence of parameter structures, while output is saved in a **data** array, and optionally in data files. During the simulation, global averages are calculated for time-step and sampling errors, together with comparisons. When completed, timing and errors are printed.

The high-level xSPDE functions process the input parameters, producing simulation data and graphs.

- **xspde (input)**
  This is the combined xSPDE function. It accepts a simulation sequence, *input.* This can be a single structure, *p*, or else a list of structures, *p1,p2,..*, for sequences. Output graphs are then displayed automatically. It returns the output [*error, data, input, rawdata*], where *error* is a vector of simulation errors and difference errors found in the comparisons. If a filename is specified in the input, it writes an output data file including input and output data. Raw data is stored on request. It calls the functions *xsim* and *xgraph.*

- **xsim (input)**
  This is the xSPDE simulation function. Like *xspde*, it accepts input parameters in *input.* It returns *[error, data, input, raw]*, where: *error* is the vector of errors, *input* is the full input structure or cell array for sequences, including default values, and *data* is a cell array of average observables. If the *p.rawdata* option is used, data for the actual trajectories is output in *raw.* This can be run as a stand-alone function if no graphs are required.

- **xgraph (data [,input])**
  This is the xSPDE graphics function. It takes computed simulation *data* and *input.* It plots graphs, comparisons and error-bars. The *data* should have as many cells as *input* cells, for sequences. If *data= 'filename.mat'*, the specified file is read both for *input* and *data.* Here *.mat* indicates a Matlab file. When the *data* input is given as a filename, input parameters in the file are replaced by any of the the new *input* parameters that are specified. Any stored *input* can be overwritten when the graphs are generated. This allows graphs of data to be modified retrospectively.

15.1.1. Output data storage and batch jobs

An xSPDE session can either run simulations interactively, described in section 1, or else using a function file called a project file. 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**.

Alternatively, it can be 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.

15.1.2. Batch input template

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

```
function e = project.m
p.[label1] = [parameter1];
p.[label2] = ...;
p.file = '[myfile].mat'
[e,~,p] = xsim(p);
xgraph(p.file);
end
```

Alternatively, for an interactive session one can use the commands:

```
...
[e,data,p] = xsim(p);
xgraph(data,p);
...
```

This is specially useful if one wishes to have direct access to the data and graphics options, with possible multiple trials. When preparing a project file using the editor, click on the Run arrow above the editor window to run the job.

A typical batch job workflow is as follows:

- Create the metadata p, including a file name, eg, p.file='myfile.mat'.
- Change the Matlab directory path to your preferred directory.
- Run the simulation with [e,data, p] = xsim(p), or just xsim(p).
- Run xgraph(p.file), and the data will be graphed.
- Alternatively, xgraph(p.file,p) allows you to change the inputs in the structure p.
- Graph outputs can be stored using the p.saveeps=1 and/or p.savefig=1 options.

You can use Matlab (.mat) files for data storage. If raw data is generated it will be stored too, but the files can be large. For stored graphics files the options are a range of standard graphics files or Matlab figure files, obtained using the graphics input parameters p.savegraph and/or p.savefig.

15.2. Input, output and logic

To explain xSPDE in full detail,

- Simulation parameters are stored in the **input** list.
- This describes a sequence of parameter structures, so that **input=p1,p2,...**.
- Each structure **p1,p2,...** generates an output which is the input of the next.
- The main simulation function is called using **xsim(input).**
- The RMS errors and integration time are returned in the **error** vector
- Parameters including defaults are returned in the **output** cell array.
- Averages are recorded sequentially in the **data** cell array.
- Raw trajectory data is optionally stored in the **raw** cell array.

The sequence input defines a sequence of individual simulations, with parameters that specify the simulation functions and give the equations and observables. If there is only one simulation, just one data structure is needed, without a cell array. In addition, xSPDE can generates graphs with its own graphics program, xGRAPH.

15.2.1. Applications

The parameters that xSIM uses are divided into applications for ease of use. Almost all parameters have default values. The SDE parameters are common to all applications, but the default values may be changed in more specialized cases. Defaults are defined through preference functions that are included in each application folder. Parameters are shared between the applications where this is meaningful.

Current application folders are as follows:

SDE Stochastic differential equation data and methods

SPDE Partial differential equation extensions and grids.

Projections This is the projective library, used to solve projected SDE/SPDEs

Quantum Stochastic Schrödinger and master equations, including logic gates

Phase-space Functions for quantum phase-space simulatoins of networks

Forward-backward Methods and functions for forward-backward stochastic equations

All applications use cell arrays of integration variables and output averages. In all cases, a single variable, vector or array can be used instead of a multicomponent cell array. All data outputs are xGRAPH compatible, except for raw trajectory outputs that need to be further processed if graphs are needed.

15.2.2. Functions

The xSIM input objects include parameters and functions, with an extensible architecture. Most xSIM functions except system functions are modular and replaceable. This is as easy as just defining a new function handle to replace the default value.

There are two types of functions:

- **User** functions define equations, and have default values. The defaults are usually obtained by adding 'x' in front of the name. In the case of method, the default depends on the problem.
- **Support** functions start with a capital and provide commonly used functionality. In some cases these are defaults for user functions.
- All arguments in square brackets are optional, but may be needed only in specific cases.
- The last argument, p, is the parameter structure.

For example, to define your own integration function, include in the xSPDE/xSIM input the line:

```
p.method = @Mystep;
```

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

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

15.2.3. xSIM exported data

The following table show how xSPDE output data is stored, which helps customize and extend the code. There are several different types of arrays used. Averages are generated from the observe functions, p.observe. These are modified, if required, by user functions p.output, and exported as graphics data. The exported data has additional sequence and check indices.

The internal averages and the exported graphics data are as follows:

Label	Indices	Description
av	$\{n\}(\ell,\mathbf{j})$	Internal averages
d	$\{s\}\{n\}(\ell,\mathbf{j},c)$	Graph data

Here:

- s is the sequence index
- n is the graph index
- ℓ is the graphics line index
- j_1 is the time index
- $\mathbf{j} = j_1, j_2, \ldots j_d$ is the space-time index
- c is the check index

15.2.4. Check index uses

There are multiple uses for the last index, c. It can be omitted if needed. If present, it stores data for errors and comparisons. This is indicated by the input parameter field $p.errors > 0$, which is the index of the largest error field. If there are no parameters, or $p.errors = 0$, there is no error or comparison index. The standard value that xSIM outputs is $p.errors = 3$.

When the check index present, the index values are defined as follows:

$c = 1$ for the average of the n-th output function

$c = 2$ for the total systematic errors, like time-step error

$c = 3$ for the random sampling error.

$c = 4$ for (optional) comparisons

$c = 5$ for (optional) systematic comparison errors

$c = 6$ for (optional) statistical comparison errors

If xGRAPH is used with data from an other source, with no error fields, but with comparisons, then simply put $p.errors = 1$, or if there is only one input error field, set $p.errors = 2$.

15.2.5. Comparisons

For every type of observation in xSIM, the observe function can be accompanied by a comparison function, compare(p). This generates a vector of analytic solutions or experimental data which is compared to the stochastic results. Results are plotted as additional lines on the two-dimensional graphical outputs, and comparison differences can be graphed in any dimension.

Comparisons are possible for either moments or probabilities, and can be input in any number of dimensions. When there are error estimates, a chi-squared test is carried out to determine if the difference is within the expected step-size and sampling error bars. If the comparison has errors, for example from experimental data, the chi-squared test will include the experimental errors.

Comparison data can be added to the graphics files from any source. It must match the corresponding space-time lattice or probability bins that are in the graphed data. Note that the compare functions are specified during the simulation. The graphics code does not generate comparison data, as it is dedicated to graphics, not to generating data.

15.3. SDE and SPDE input parameters

Simulation parameters are stored in a parameter structure which is passed to the *xsim* program. Constants can be included, but must not be reserved . All names starting with a capital letter like 'A...' are available, except for derivatives 'D..'.. Globals are incompatible with the Matlab parallel toolbox. Graphics data is stored for the graphics program to use.

Standard inputs have default values, which are user-modifiable through the xpreferences function. Defaults can be checked by including the input $p.verbose = 2$. All the inputs are part of a structure passed to xSPDE. If a cell array of multiple structures are input, these are executed in sequence, with the output of the first simulation passed to the second, then the third, and so on.

If there is a single-component vector or cell array parameter input, just the component can be entered. Library functions do not have defaults. All inputs in the table below are preceded by the parameter structure label, usually *p*. This can be chosen by the user as long as it is used consistently when parameters are passed.

Label	Default value	Description
$A, B, C, \ldots$	-	User specified static parameters
adapt	1	Threshold for inversion in adaptive midpoint method
auxfields	[]	Auxiliary field dimensions
averages	1:max(observe)	Optional list of computed observe functions
axes{n}	{0,..}	Points stored for each axis
binranges{n}	{0,0,...}	Observable binning ranges for probabilities
boundaries{n}	$[0, 0; 0, 0]$	Boundary: '-1,0,1'=Neum, periodic, Dirichlet boundary.
checks	1	Check errors for time grid: 0 or 1
cutoff	-10^{20}	Global lower data cutoff
cutoffs{n}	$cutoff$	Lower data cutoff
dimensions	1	Space-time dimensions
ensembles	[1, 1, 1]	Size of [vector, serial, parallel] ensembles
fields	1	Forward stochastic field dimensions (or cell)
fieldsb	0	Backward stochastic field dimensions (or cell)
file	''	File-name: 'f.mat' = Matlab
ftransforms	{[0 0 0 0],..}	Fourier transforms in [t,x,y,z,..] per function
inrandoms	noises	Initial random fields (or cell)
ipsteps	2	IP transforms per time-step: depends on the method
iterations	4	Projection, implicit or midpoint iterations
iterfb	0	Forward-backward iterations
knoises	0	Filtered noise fields (or cell)
krandoms	0	Filtered initial randoms (or cell)
mincount	0	Lower count cutoff for chi-squared estimates
name	''	Simulation name
noises	fields	Number of noise fields (or cell)
octave	0	Force octave syntax: 1 for octave
olabels{n}	'a_1'	Labels to identify the calculated outputs
order	1	Extrapolation order: depends on the method
origins	[0,..]	Origin of coordinates in [t,x,y,z,..]
points	[51,...	Output lattice points in [t,x,y,z,..]
qc	-	Quadratic projection coefficients
ranges	[10, ..]	Range of coordinates in [t,x,y,z,..]
rawdata	0	Raw data switch: 1 for raw output
relerr	1	Flag for normalising the error estimates
scatters	{0,..}	Specify to obtain scatter plots, not averages
seed	0	Seed for random number generator
steps	1	Integration steps per output point
thresholdw	0	Threshold for weighted simulation breeding
tol	10^{-20}	Threshold for error normalisation
transforms	{[0 0 0 0],..}	Fourier transforms in [t,x,y,z,..] per observable
unoises	0	Uniform noise fields (or cell)
urandoms	noises	Uniform initial randoms (or cell)
vc	-	Vector projection coefficients
verbose	0	0 for brief, 1 for informative, 2 for full output
version	'xSIM4.xx'	Current version number

Detailed descriptions are as follows:

15.3.1. A,B,C...

Capital letters and words starting with capitals are reserved to store the user-specified constant values. They are passed to user functions and can be any data. All inputs — including user constants — are copied into the stored data files via the lattice structure p, to give a permanent record of all the simulation parameter values and outputs.

Example: p.Constant = 2*pi

15.3.2. adapt

Default: 1

Threshold for adaptive inversion in the *MPadapt* midpoint method.

Examples: p.adapt = 2

15.3.3. auxfields{c}

Default: 0

These are real or complex auxiliary fields stored at each lattice point, specified using *p.define*. They are used for input/output spectral calculations, and can be functions of the noise. Like fields, there can be several of these, defined in a cell array. Also available for storing noise fields if needed.

Examples: p.auxfields = 2 ,p.auxfields = {1,2}

15.3.4. averages

Default: 1:maximum_observable

This optional input gives a vector of average indices to calculate. Default value is all the observe functions. This is used to suppress unwanted outputs, for example, while testing an input script.

Examples: p.averages = [2,3,4];

15.3.5. axes{n}

Default: {0,0,0,..}

Gives the axis points used for comparisons in the n-th output function, in each dimension. For each function, the axes can be individually specified in each dimension. Each entry value is a vector range for a particular dimension, for d=1,...p.dimensions. Thus, 5 gives the fifth point only in that dimension, and an input 1:4:41 plots every fourth point. Zero or negative values are shorthand: -1 generates a default point at the midpoint, -2 the endpoint, and 0 is the default value that gives the vector for the every axis point. This data is also used to control graphics outputs. It can be input separately for each graph if required. If there are extra space points using the p.steps input, then the spatial points are expanded internally, and axes is used to select the output points. If desired, this can be changed, but the larger number of space points should be taken into account.

Example: p.axes{4} = {1:2:10,0,0,-1}

15.3.6. binranges{n}

Default: {}

Nested cell array, $binranges\{n\}\{m\}$, that defines the probability plotted for observable n. If null or zero, the mean of the observable is calculated as usual. The second cell index, $m = 1, \ldots M$, corresponds to the line index returned by the corresponding n-th observe function. When nonzero, the probability of the n-th observable is calculated and plotted according to the specified vector of axis points. This sets extra dimensions in the data, depending on the range of m values, with $[o_1, o_2, \ldots o_K]$, being the start and end of each of the bins used to accumulate probabilities. The $k - th$ bin is centered at $(o_k + o_{k+1})/2$. In this version of xSPDE, each bin must have the same width for an observable and line number. The output is the average probability density versus the (vector) value of the observable. Hence M extra output dimensions are added to the generated probability data.

Example: p.binranges{n}{1} = {-5:0.1:5,-2:0.1:2}

15.3.7. boundaries{c,d}

Default: [0, 0]

Cell array for type of spatial boundary conditions used, set for each dimension and field component independently, and used in the equation solutions. The cell index is $dir = 2, 3, ..$, indicating the dimension. The boundary conditions are defined as a matrix. The first index is the field index i and the second index the boundary j, with $j = 1$ for the lower and $j = 2$ for the upper boundary. The options are $b = -1, 0, 1$.

- The default option, or 0, is periodic.
- If -1, Robin/Neumann boundaries are used, with derivatives set to prescribed values.
- If 1, Dirichlet boundaries are used, with fields set to prescribed values.

In the current code, only default boundaries are available using spectral (linear) methods. Using arbitrary non-periodic boundaries requires the use of finite difference derivatives, without the option of an interaction picture derivative. In such general cases, arbitrary boundary values are set by boundfun(a,d,p). If the cell index c is omitted, the first cell is used.

Example: p.boundaries{d} = [-1,1;0,0;1,-1]

15.3.8. checks

Default: 1

This defines if a repeat integration is carried out for error-checking purposes. If p.checks = 0, there is one integration, with no checking at smaller time-steps. For error checking, set p.checks = 1, which repeats the calculation at half the time-step — but with identical noise — to obtain error bars. This is the default value, taking three times longer overall, but with increased accuracy and error-estimates.

Also see the order parameter, below.

Example: p.checks = 0

15.3.9. cutoff

Default: -10^{20}

This is a global lower cut-off in data used in χ^2 tests and graphs. Can be over-ridden by using individual output *cutoffs.*

Example: p.cutoff= 0

15.3.10. cutoffs{n}

Default: $cutoff$

This is an individual lower cut-off in data used in χ^2 tests and graphs, specific to output n

Example: p.cutoff{2}= 0

15.3.11. dimensions

Default: 1

This is the space-time dimension for an SPDE. If omitted, dimensions=1, giving an SDE. It is arbitrary apart from the obvious memory requirements at large dimensionality.

Example: p.dimensions = 4

15.3.12. ensembles

Default: [1, 1, 1]

Number of independent stochastic trajectories simulated. This has three levels to maximize efficiency. The first is within-thread parallelism, allowing vector instructions. The second gives a number of independent trajectories calculated serially. The third gives multi-core parallelism and requires the Matlab parallel toolbox. Either p.ensembles(2) or p.ensembles(3) are required to obtain sampling error-bars. The total number of stochastic trajectories or samples is $ensembles(1) \times ensembles(2) \times ensembles(3)$. The second and third ensembles cannot be changed during a sequence of simulations.

Example: p.ensembles = [1000,100,10]

15.3.13. fields

Default: 1

These are variables stored at each lattice point that are the independent variables for integration. The fields are vectors or arrays that can have any number of components or dimensions. The $fields$ input is the number of forward-propagating real or complex components initialized by the initial function and integrated using the deriv derivative. One array can be used, or cell arrays of multiple named fields. See the specific method for details.

Example: p.fields = {2,[3,3]}

15.3.14. fieldsb

Default: 0

The fieldsb input is the number of real or complex components propagating in the backward time direction, for a forward-backward equation. One array can be used, or cell arrays of multiple named fields.

Example: p.fieldsb = {1}

15.3.15. file

Default: ' '

Matlab file name for output data. Includes all data and parameter values, including raw trajectories if $p.rawdata = 1$. If not needed just omit this. A Matlab filename should end in .mat. For a sequence of inputs, the filename should be given in the first structure of the sequence, and the entire sequence is stored. This cannot be changed for successive parts of the overall sequence.

Example: p.file = 'file-name'

15.3.16. ftransforms{n}

Default: transforms{n}

Cell array defining the Fourier transform switches for output n. There is one ftransform vector per output function. The n-th flag indicates a Fourier transform if it is set to one, and none if set to zero. The default value is the observe transform switch. If there are more functions than observe handles, the additional transform switches default to zero.

This is used to identify which outputs are from an initially transformed observe average, so they can be graphed with the correct axis labels. This is only needed if there are multiple outputs generated from one transformed observe average. Otherwise, the default is completely adequate.

Example: p.ftransforms{n} = [1,0,0,1]

15.3.17. inrandoms{n}

Default: noises

This defines the initial random Gaussian fields generated per lattice point in coordinate and momentum space. Set to zero ($p.inrandoms = 0$) for no random fields. Random fields are delta-correlated in x-space. This can be a scalar, vector or an array. It can optionally be a cell array of multiple vectors. The maximum number of inrandom cells equals the number of field cells plus auxiliary field cells.

Example: p.inrandoms = 2

15.3.18. ipsteps

Default: 1 for Euler, Implicit and RK2; 2 for MP, MPadapt and RK4; 0 otherwise

This specifies the number of interaction picture time-steps needed in an integration time-step. Default values are specified in method. Can always be changed for custom integration methods. This must be initialized if a non-standard integration method is used that requires an interaction picture, and the relevant data isn't returned by method.

Example: p.ipsteps = 1

15.3.19. iterations

Default: 4

For iterative algorithms like the implicit midpoint method, the iteration count is set here, typically around 3-4. Will increase the integration accuracy if set higher, but it may be better to increase steps if this is needed. With non-iterated algorithms, this input is not used. Also used to specify the iterations in projection methods.

Example: p.iterations = 3

15.3.20. iterfb

Default: 0

For iterative forward-backward stochastic equation methods. Set to 0 for standard forward equations.

Example: p.iterfb = 3

15.3.21. knoises

Default: [ ]

This gives the number of Gaussian noises generated per lattice point in momentum space. This allows use of finite correlation lengths, by including a frequency filter function that is used to modify the noise in Fourier-space. The Fourier-space random variance is defined by the filter function. This takes the noises in Fourier space and returns a filtered version, which is inverse Fourier transformed before use. Filtered noises have a finite correlation length. This can be a scalar, vector or an array. It can optionally be a cell array of multiple vectors. The maximum number of cells is the number of field cells plus auxiliary cells. Omitted if it is not input, or null.

Example: p.knoises = [2,4].

15.3.22. krandoms

Default: [ ]

This gives the number of initial random Gaussian fields generated per lattice point in momentum space. The fields are delta-correlated in momentum space, with a variance modified by the filter function. This takes initial random fields in Fourier space and returns a filtered version, which is inverse Fourier transformed before use. This can be a scalar, vector or an array. It can optionally be a cell array of multiple vectors. The maximum number of cells equals the number of field cells plus auxiliary field cells. The filtered random inputs have a finite correlation length.

Example: p.krandoms = 2

15.3.23. mincount

Default: 0

This gives the minimum count in a χ^2 calculation involving sampled probabilities.

Example: p.mincount = 10

15.3.24. name

Default: ' '

Name used to label simulation, usually corresponding to the equation or problem solved. This can be removed from graphs using headers equal to a single blank space when running xgraph.

Example: p.name = 'your project name'

15.3.25. noises

Default: [ ], or p.fields if no other noises are specified

This gives the number of Gaussian noises generated per lattice point, in coordinate and momentum space, respectively. Set to zero ($p.noises = 0$) for no noises. Noises are delta-correlated in x-space. This can be a scalar, vector or an array. It can optionally be a cell array of multiple vectors. The maximum number of cells equals the number of field cells plus auxiliary field cells.

Example: p.noises = {2,4}.

15.3.26. octave

Default: 0

Switch for Octave program syntax, which can differ from Matlab standards. Octave is an open source, free alternative to Matlab.

Example: p.octave = 1

15.3.27. olabels{n}

Default: 'a'

Cell array of labels for the graph axis observables and functions. These are text labels that are used on the graph axes. The default value is 'a_1' if the default observable is used, otherwise it is blank. This is overwritten by any subsequent label input when the graphics program is run:

Example: p.olabels{4} = 'v'

15.3.28. order

Default: 0

This is the extrapolation order, which is only used if $p.checks = 1$. The program uses the estimated convergence order to extrapolate to zero step-size, with reduced errors. If p.order = 0, no extrapolation is used, which is the most conservative input. The specific default order returned by the method can be used if one specifies p.order = -1.

The extrapolation order cannot be changed during a sequence. The default deterministic orders of the six preset methods used without stochastic ensembles are:

1 for Euler and Implicit;

2 for RK2, MP and MPadapt;

4 for RK4.

Example: p.order = 0

15.3.29. origins

Default: [0, -p.ranges/2]

This displaces the graph origin for each simulation to a user-defined value. If omitted, all initial times in a sequence are zero, and the space origin is set to -p.ranges/2 to give results that are symmetric about the origin. As an example, for the x-dimension, the problem is solved on an interval of $x = [O_2, O_2 + R_2]$, with a default origin of $-R_2/2$, so that $x = [-R_2/2, R_2/2]$. There is no cell index used.

Example: p.origins = [0,-20,-20]

15.3.30. points{n}

Default: [51, 35, ..., 35]

The rectangular lattice of points plotted for each dimension and field cell n, are defined by a vector giving the number of points in each dimension. The default values are given as a guide for initial calculations. Large, high dimensional lattices take more time to integrate. Increasing points improves graphics resolution and gives better accuracy in each relevant dimension as well, but requires more memory.

Cells for n >1 can be reduced to singleton dimensions to treat boundary fields, but the smallest space-steps used in the integrations are defined relative to points{1}. Speed when using spectral methods is improved when the lattice points are a product of small prime factors. In order to discretize the problem, the p_i lattice points are fitted into the range R_i so that $dx_i = R_i/(p_i - 1)$, ie:

$$x_i = O_i + (i - 1)dx_i \,. \tag{15.1}$$

Example: p.points = [30,40,40]

15.3.31. qc

Default: 0

Quadratic surface coefficients for projected stochastic equations. For details, refer to the projected methods section.

15.3.32. ranges

Default: [10, 10, ...]

Each lattice dimension has a coordinate range. The default value is 10 in each dimension. In the temporal graphs, the first coordinate is plotted over $0 : p.ranges(1)$. All other coordinates are plotted over $-p.ranges(n)/2 : p.ranges(n)/2$. The starting value in any dimension can be changed using the origins variable. This is not a cell array, since the ranges are the same for all field cells (see: points).

Example: p.ranges = [1, 10]

15.3.33. rawdata

Default: 0

Flag for storing raw trajectory data. If this flag is turned on, raw trajectories are stored in memory. The raw data is returned in function calls and also written to a file on completion, if a file-name is included.

Example: p.rawdata = 1

15.3.34. relerr

Default: 1

Flag for normalizing the error data. If p.relerr = 1 then all errors are normalized either by the maximum output value, or else by the maximum comparison value, if there is one. If p.relerr = 0, or the normalisation factor is less than *p.tol*, the absolute error values are output without normalization.

Example: p.relerr = 0

15.3.35. scatters{n}

Default: 0

Cell array that defines the number of scatter trajectories plotted for observable n. If absent or zero, the mean of the observable is calculated as usual. If nonzero, a set of s observables that correspond to independent stochastic fields are accumulated, with no averaging. This cannot be combined with probabilities or with parallel ensembles. There must be at least s trajectories in ensembles(1), otherwise the number of stored trajectories is reduced.

Example: p.scatters{n} = 20

15.3.36. seed

Default: 0

Random noise generation seed, for obtaining reproducible noise sequences. This is automatically set to unique, distinct values for different parallel ensembles. Used if *p.noises* > 0 or *p.inrandoms* > 0.

Example: p.seed = 42

15.3.37. steps

Default: 1

Number of internal steps per plotted point. The total number of integration time-steps in a simulation is therefore p.steps×(p.points(1)-1). Thus, steps can be increased to improve the accuracy, but gives no change in graphics resolution. Increasing the steps will give a lower time-discretization error. If this is a vector, then the number of internal space points is also increased, with each dimension changed independently, otherwise only the time-step is changed.

Example: p.steps = [1, 2, ...]

15.3.38. thresholdw

Default: 0

Threshold weight for breeding method in weighted simulations. The default is zero, which means no breeding is used.

Example: p.thresholdw = 0.01

15.3.39. tol

Default: 10^{-20}

Threshold size for normalising error estimates. There is no relative error normalisation if the normalising factor is less than *p.tol*. In addition, this tolerance is used to indicate which quantities are included in error summaries. Data with absolute magnitude below this is regarded as not significant, since round off errors will typically be at least as large as this. This value can be adjusted or set to zero by the user if the data is significantly larger or smaller, but it can result in division by zero if the errors are normalised and the tolerance is zero.

Example: p.tol = 10^{-100}

15.3.40. transforms{n}

Default: [0,0,..]

Cell array defining the Fourier transforms used for an observable n. There is one transform vector per observable. The n-th flag indicates a Fourier transform on the n-th axis if set to one, starting with the time axis. The default value is zero, indicating no transform. The normalization of the Fourier transform is such that the $k = 0$ value in momentum space corresponds to the integral over space with a factor of $1/\sqrt{2\pi}$ in each transformed dimension. The Fourier transform that is graphed has $k = 0$ as the central value. The default is no Fourier transform. Must be set for any functional transform of a Fourier observable, to give the correct graph axes.

Example: p.transforms{n} = [1,0,0,1]

15.3.41. unoises

Default: [ ]

This gives the number of uniform noises generated per lattice point, in coordinate space. This can be a scalar, vector or an array. It can optionally be a cell array of multiple vectors. The maximum number of cells equals the number of field cells plus auxiliary field cells. Omitted if it is not input, or null.

Example: p.unoises = [2,4].

15.3.42. urandoms{n}

Default: [ ]

This gives the number of initial uniform random fields generated per lattice point. The fields are uncorrelated in ordinary space. This can be a scalar, vector or an array. It can also be a cell array of multiple vectors. The maximum number of cells equals the number of field cells plus auxiliary field cells. Omitted if it is not input, or null.

Example: p.urandoms = {1,2}

15.3.43. vc

Default: 0

Vector projection coefficients for projected stochastic differential equations.

Example: p.vc = [2,3,4]

15.3.44. verbose

Default: 0

Print flag for output information while running xSIM. Print options are:

- Brief if verbose = 0: Additionally prints the final, total integration errors
- Informative if verbose = 1: Also prints the individual function RMS errors and progress indicators
- Full if verbose = 2: Prints everything, including the internal parameter structure data.

In summary, if verbose = 0, most output is suppressed except the final data, while verbose = 1 displays a progress report, and verbose = 2 additionally generates a readable summary of the parameter input as a record.

Example: p.verbose = 2

15.3.45. version

Default: 'xSIM4'

Sets the current version number of the simulation program. There is no need to input this except for project documentation for a customized version.

Example: p.version = 'current version name'

15.3.46. User-defined functions.

These functions define the stochastic problem. The three most important ones are given in boldface. These are generated automatically by the quantum application, to simplify the user inputs and interface. Their calling arguments, and purpose, are:

Label	Arguments	Purpose
initial{n}	$((y,)z,p)$	**Functions to initialize fields**
deriv{n}	$(a,..w,..p)$	**Total stochastic derivatives**
observe{n}	(a,p)	**Observable functions**
derivA{n}	(a,p)	Drift derivative term
derivB{n}	(a,p)	Noise derivative term
linear {n}	(p)	Linear derivative function
transfer {n}	$(a0,z,p)$	Transfer inside a sequence
method	(a,w,p)	Algorithm defining a time-step*
output{n}	(o,p)	Output function
compare{n}	(p)	Function for differences and χ^2
define {n}	(a,w,p)	Defines an auxiliary field value
firstfb{n}	(p)	Defines the first forward-backward estimate
boundfun	(a,c,d,p)	Boundary function
project	$(d,a,n,(c,)p)$	Defines projections

*In all cases except for method, the calling variables are a list of field and noise arrays. The method function inputs cell arrays of fields and noises, and a parameter structure. It outputs a field cell array.

If cell arrays have more than one member, then $(a,w,p) \to (a,b,c,..w,x,y,..p)$, where a,b,c are the fields and $w,x,y,$ are the noises. The cell array of deriv, initial, or transfer functions must be as large as the cell array of integrated fields.

For the case of forward-backward equations, the *initial* function argument is (y,z,p), where y is the field that is propagating in the opposite time direction at the time boundary. This is the backward field for a forward initialisation, and vice-versa.

15.3.47. Integrals and derivatives

For details of the internal integration and differentiation functions that can be used in deriv, observe and define see section 15.6 and sections 13 and 13. All xSPDE internal functions are capitalized. Note that D1, D2 use finite differences, DS uses spectral methods. These require a cell index c to specify the boundary conditions. The functions are:

Label	Arguments	Purpose
Ave	$(a,[av,]p)$	Averages over a spatial lattice
D1	$(a,[d,c,ind,]p)$	First derivative
D2	$(a,[d,c,ind,]p)$	Second derivative
DS	$(a,[n,d,c,ind,]p)$	Spectral derivative, n-th order
Int	$(a,[dx\,or\,dk,bounds],p)$	Integrates over space or momentum

- For derivatives, d is the dimension, c the cell, ind the first index values.
- Defaults are $d=1$, $c=1$, and all indices.

- For *Int*, one can integrate either with respect to dx or dk, in either ordinary space or momentum space, by changing the second argument passed to *xint*.
- For integration in momentum space, fields that are passed to *Int* are only transformed if the observe function is used with Fourier transforms selected using transforms.
- For integrating functions like function{n} with transforms, the transform flags transforms{n} should be used both for the function and any observe averages used, to ensure correct graphical output. Data is always transformed before averaging.

15.3.48. Extensible functions

Extensible functions define the numerical methods used. They use a similar pattern of (fields..,noises.., parameters). For generality, these all pass and return cell arrays of fields and noises.

They all have defaults, and needn't be input in user code when the default available is used. Any compatible user function can be employed instead. If required, use the following syntax:

```
p.method = @My_extended_method;
```

.

The system default values don't usually have to be changed unless required.

Label	Default Value	Arguments	Purpose
method	@$MP, RK4$	(a, w, p)	Algorithm defining a time-step
grid	@xgrid	(p)	Grid calculator for the lattice
prop	@xprop	(a, p)	Interaction picture propagator
propfactor	@xpropfactor	(nc, p)	Propagator array calculation
randomgen	@xrandom	(p)	Initial random generator
noisegen	@xnoise	(p)	Noise generator

15.3.49. SDE methods table

For details of the internal methods available, see section 15.6 and section 13. All xSIM internal method functions are capitalized. Currently only the MP method is available for jump processes as well as SDEs.

They are:

Label	Arguments	Purpose
*Euler	(a, w, p)	Euler algorithm
MP	(a, w, p)	Midpoint algorithm
MPadapt	(a, w, p)	Midpoint adaptive algorithm
RK2	(a, w, p)	Runge-Kutta (2) algorithm
RK4	(a, w, p)	Runge-Kutta (4) algorithm
**Implicit	(a, w, p)	Implicit or time-reversed
*RKWP21	(a, w, p)	Weak Runge-Kutta

All standard methods except the last can use the xprop interaction picture propagator, which also projects onto boundaries. They can all normalize quantum wave-functions and density matrices if $p.quantum > 0$, and can treat vectors, arrays and cells. The MP and RK methods are intended for Stratonovich equations.The '*' methods are for Ito equations, while '**' methods are for time-reversed Ito equations.

In general, the input and output,'a' is a cell array of fields. The fields themselves can be scalars, vectors or tensors. These can have definitions that include spatial indices. However, there are some restrictions also:

- Field tensors can't be integrated in space.
- Currently, quantum fields and non-quantum fields can't be mixed.
- The first six methods use single derivative functions with variable length arguments, $p.deriv(a, b, ..w_a, w_b.., p)$.
- Weak stochastic methods require two derivative functions, $p.derivA(a, p)$ and $p.derivB(a, p)$.
- A weak, second order Ito method, RKWP21 is available for scalar ODEs with scalar noise.

The standard xSPDE methods return a five integer information cell array after the main field. This gives information about the stochastic and deterministic order, interaction picture handling, and capabilities for treating multicomponent vectors and cells:

{Stochastic order, deterministic order, ipsteps, vector? (1,0), cell? (1,0)}.

15.3.50. Projection methods

More advanced standard methods are also available:

Label	Arguments	Purpose
Catproj	$(d, a, n, (c,)p)$	Catenoid projector
Quadproj	$(d, a, n, (c,)p)$	General quadratic projector
Polproj	$(d, a, n, (c,)p)$	Diagonal polynomial projector
Enproj	(a, w, p)	Euler normal projection method
MPproj	(a, w, p)	Midpoint projection method
MPnproj	(a, w, p)	Midpoint normal projection method

- Projection algorithms with a 'proj' suffix require a user-defined project function.
- The method functions take cell inputs and outputs.
- The projectors with 4 arguments take individual arrays as inputs and outputs.
- If the cell index c is added, the projector field input a must be a cell array

15.4. Quantum parameters

15.4.1. Quantum Monte-Carlo parameters

The QUANTUM parameters are identical to the xSIM parameters, with additional functions and methods. Currently only one cell-array index is available.

There are three switchable options that can be chosen, as well as parameters and functions:

Label	Default value	Description
quantum	$0, 1, 2$	Classical (0), wave-functions (1), density matrices (2)
sparse	0	Use functional (0) or sparse (1) operators
jump	0	Use continuous (0) or jump (1) methods
nmax	2	Vector of Hilbert-space dimensions per mode
modes	1	Number of modes
gamma	{@(p) 0}	Decay rate function per process and mode
L	{[]}	Cell array of loss operators
theta	{0}	Cell array of loss operator phases

Wave-functions are stored in a packed, one-dimensional form with sparse operators, in a packed, two-dimensional form with density matrices, and in a multidimensional array with functional operators. Sparse operators usually give faster results, but require greater overall memory storage for larger Hilbert spaces.

For operators with two mode indices, the second mode index can be omitted if identical to the first. Hermitian conjugate operators are returned if the mode index is negative.

15.4.2. Bosonic operator table

These are system functions that are permanently available.

Label	Inputs	Output(s)
a	(m, psi)	$\hat{a}_m \|\psi\rangle$
a2	(m, psi)	$\hat{a}_m^2 \|\psi\rangle$
n	$([m_1 (, m_2)], psi)$	$\hat{a}_{m_1}^\dagger \hat{a}_{m_2} \|\psi\rangle$

15.4.3. Sparse bosonic operator table

Requires use of *Mkbose* to access these sparse matrix operators, which are temporarily created as a cell array when required, since they use up memory.

Label	Inputs	Output(s)
p.a	$\{m\}$	$\hat{a}_m$
p.a'	$\{m\}$	$\hat{a}_m^\dagger$

15.4.4. Qubit and Pauli spin operators

These are functions that are permanently available.

Label	Inputs	Output(s)
sx	(m, psi)	$\hat{\sigma}_m^x \|\psi\rangle$
sy	(m, psi)	$\hat{\sigma}_m^y \|\psi\rangle$
sz	(m, psi)	$\hat{\sigma}_m^z \|\psi\rangle$
sx2	$([m_1\,(,m_2)], psi,)$	$\hat{\sigma}_{m_1}^x \hat{\sigma}_{m_2}^x \|\psi\rangle$
sy2	$([m_1\,(,m_2)], psi)$	$\hat{\sigma}_{m_1}^y \hat{\sigma}_{m_2}^y \|\psi\rangle$
sz2	$([m_1\,(,m_2)], psi)$	$\hat{\sigma}_{m_1}^z \hat{\sigma}_{m_2}^z \|\psi\rangle$

15.4.5. Quantum logic gate operators

These are functions that are permanently available.

Label	Inputs	Output(s)
ha	(m, psi)	$h \|\psi\rangle$
ph	(m, psi)	$p \|\psi\rangle$
p8	(m, psi)	$t \|\psi\rangle$
cx	$([m_1\,(,m_2)], psi,)$	$cx \|\psi\rangle$

15.4.6. Quantum functions

For details of the internal quantum functions that can be used see section 15.6 and sections 13 and 13. All xSPDE internal functions are capitalized. Note that D1, D2 use finite differences, DS uses spectral methods. These require a cell index c to specify the boundary conditions. The functions are:

Label	Arguments	Purpose
Mkbose	$([list,]p)$	Makes sparse bosonic operator matrices
Mknumber	(nv, p)	Makes number states with occupations nv
D2	$(a, [d, c, ind,]p)$	Second derivative
DS	$(a, [n, d, c, ind,]p)$	Spectral derivative, n-th order
Int	$(a, [dx\, or\, dk, bounds], p)$	Integrates over space or momentum

p.a = Mkbose((list,) p) Returns a cell array of annihilation operators defined either at all modes, if there is no list, or at the listed mode locations. Here list is a vector of integers, p is the parameter structure.

15.4.7. Quantum phase-space parameters

The quantum phase-space parameters are identical to the xSIM parameters, with additional functions and methods. Currently only one cell-array index is available.

There are three switchable phase-space options that can be chosen, that correspond to normal, symmetric and anti-normal ordering. The other parameters define types of Gaussian inputs, and measurements.

Label	Value	Purpose
phase	$1, 2, 3$	Use +P, Wigner or Q ordering
sqz	*real* vector	Squeezing vector
alpha	complex vector	Coherent input amplitude
matrix	complex matrix function	Returns transmission matrix
tr	*real* vector	Amplitude transmission vector
thermal	*real* vector	Thermal fraction of input

15.5. User function reference

The following user-defined function inputs define the differential equation that is solved. They are specified in an xSPDE/xSIM input file using p.(fun) = @(Myfun). They can be inline or externally defined functions. Externally defined functions must be in the same file as the input parameters, or on the execution path.

15.5.1. boundfun(a, c, d, p)

Default: xboundfun()

The boundary function boundfun(a,c,d,p) is called for specified boundary conditions for field cell c in the d-th dimension. This returns the boundary values used for the fields or their first derivatives in space dimension $d > 1$, as an array indexed as $b(f, \mathbf{i}, e)$ in the standard way. Here f is the field index, $\mathbf{i} \equiv [j_2, \dots j_d]$ are the space indices, and e is the ensemble index.

Only two values are needed for j_d, which is the index of the dimension whose boundary values are specified. These are $j_d = 1, 2$, for the lower and upper boundary values, which are either field values or derivatives. Boundary values may be constant or a function of the fields a and space-time $t, \mathbf{x}$.

If boundary values have stochastic values which are calculated only once, they must be initialized. To allow for this, boundfun(a,c,d,p) is initially called with time $t = origins(1) - 1$, and with the input field a set to random values from randomgen, which are independent of those that initialize the field at $t = origins(1)$.

They are reproducible for different *check* cycles, to allow noise-independent error-checking. The initial results for the boundaries are stored in an array boundval{c,d} for later use by boundfun.

The default boundary value is zero, or equal to boundval if it is specified initially. It is automatically set by the default boundary function xboundfun(a,c,d,p).

15.5.2. compare{n}(p)

Default: compare{n}= []

This is for comparisons to experimental or analytic data. The output is an array with $d+2$ dimensions. The first dimension is the line index, the next d dimensions are time and space, while the last index is an error index. This can have up to two additional entries for systematic and/or statistical error bars in the comparison data, from analytic or experimental results. Error-bars are optional if not available.

15.5.3. define(a{:},w{:}, p)

Default: xdefine()

Calculates a list of auxiliary fields, which are combinations of fields and noises. They can be accessed in observe functions as part of the input cell list, after the propagating fields. These are used in spectral calculations to access the noise fields, which are needed in quantum input-output calculations. The default, xdefine(), sets the auxiliary fields to zero.

15.5.4. deriv(a{:},w{:},p)

Default: deriv()= 0

This defines the stochastic time derivative, given the current field cells a, delta-correlated noise terms w, and parameters p. It is defined explicitly in (5.15). This is the right-hand-side of (2.1) or (5.1), without the linear term if it is specified separately. In the case of multiple cell calculations, this user defined function must return a full list of all propagating derivative terms in the form of [da{1}, da{2},..].

15.5.5. firstfb{c}(p)

Default: xfirstfb()

Returns the first field estimate in a forward-backward iteration in each cell. The default function is xfirstfb, which sets each field equal to zero. Other estimates may give faster convergence.

15.5.6. grid(p)

default xgrid

Calculates the spatial grid for specialized purposes like non-uniform grids. The default, xgrid, returns a homogeneous rectangular grid in both ordinary and momentum space, as part of the parameter structure p. Grids are removed from stored data.

15.5.7. initial{c}(rv, p)

Default: xinitial()

This is used to initialize each field cell integration in time. It is a user-defined function which can involve random numbers for an initial probability distribution. This creates a stochastic field on the spatial lattice. The returned first dimension is p.fields(1). The initial Gaussian random field variable, rv, has unit variance if dimension is 1 or else is delta-correlated in space, with variance $1/p.dv = 1/(dx_2...dx_d))$ for d space-time dimensions. If inrandoms is given in the input parameter structure, rv has a first dimension of inrandoms(1) + inrandoms(2). If not specified, the default for inrandoms is noises. The default function is xinitial, which sets fields to zero. The function can be either a cell array of initial functions, or a single function if there is just one cell.

15.5.8. linear{c}(p)

Default: xlinear()

A cell array of user-defined linear response functions. It is a vector for an SDE or ODE. For an SPDE or PDE, it includes transverse derivatives in space, returning linear coefficients L in FFT/DST/DCT space, which are assumed diagonal in the field index. These are functions of differential terms Dx, Dy, Dz, which correspond to $\partial/\partial x$, $\partial/\partial y$, $\partial/\partial z$, respectively. Each component has a dimension the same as the coordinate lattice. For axes that are numbered, use D{2}, D{3} etc. The default, xlinear, sets L to zero. The function can be either a cell array of linear functions, or a single function if there is just one cell.

15.5.9. method(a, w, p)

Default: @MP (stochastic); @RK4 (deterministic)

Gives the integration method for the field cell array a, noise cell array w, parameters p. It returns the new field cell array. It uses the current reduced step in time p.dtr and current time p.t. This function can be set to any of the predefined stochastic integration routines provided with xSPDE, described in the Algorithms section. User-written functions can also be used. The default deterministic method, RK4, is a fourth-order interaction picture Runge-Kutta. The default stochastic method, MP, is an interaction picture midpoint integrator which is used if *ensembles* is not [1,1,1].

15.5.10. nfilter{n} (w,p)

Default: xnfilter()

Returns the $n-th$ momentum-space filter function for the propagation noise terms in momentum-space. Each component has an array dimension the same as the random noises in momentum space, that is, the return dimension is [knoises{n}, d.lattice].

15.5.11. noisegen(p)

Default: xnoisegen(p)

Generates arrays of noise terms for each point in time. The default, xnoisegen() returns noises Gaussian real noises that are delta-correlated in time and space, and knoises Gaussian real noises that are delta-correlated in time and momentum space, unless nfilter is used to modify momentum space correlations.

15.5.12. observe{n}(a..., p)

Default: xobserve{1}=@(a,p) a

Cell array of function handles that take the current field(s) and returns an observable o. Note the braces for cell arrays! One can input these as p.observe{n} = @(a,p) o(a,p). An omitted function less than the maximum index is replaced by the default. This is all the real field amplitudes in the first cell, with tensor fields and field arrays reshaped into vectors.

15.5.13. output{n}(o,p)

Default: @(o,p) o{n}

This is a user-defined cell array of output functions of the observe results after averaging over ensembles(1), possibly involving combinations of several observed averages. The input to the n-th output function is the cell array of all averages, and the output is the data for the n-th graph. This function is compatible with all error estimates. The default values generate all the observe averages that are in the data.

The output data format of the output functions is an array with $d + 1$ dimensions. The first dimension is the line index, the next d dimensions are time and space.

The xSIM program augments the outputs with columns of errors and comparison data, if available, before graphing.

15.5.14. project(d,a,n,(c,)p)

Default: none

A *project* function is required if a projected integration method is used. The calling arguments for the *project* function are: *(d,a,n,(c,)p)*, where d is a vector to be tangentially projected at location a, a is the current (near)-manifold location, n is an option switch, c is the (optional) cell index, and p is the parameter structure.

The options available in any *project* implementation are defined as:

- $n = 0$ returns the tangent vector for testing
- $n = 1$ returns the tangential projection of d at a

- $n = 2$ returns the normal projection of a, where d is not used
- $n = 4$ returns the constraint function at a for testing

The projections defined in a *project* function can be of any type. Arbitrary dimension reduction and manifold geometry is possible. Currently in the examples, dimensionality is reduced by 1. Normal projections use fixed point iterations, defined by *p.iterations*, with a default of *p.iterations=4*. Only the $n = 1, 2$ are essential, as the other options are only used for testing purposes.

15.5.15. prop(a, p)

Default: xprop()

Returns the fields propagated for one step in the interaction picture, given an initial field a, using the propagator array. The time-step used in propagator depends on the input time-step, the error-checking and the algorithm. The default, xprop, takes a Fourier transform of **a**, multiplies by propfactor to propagate in time, then takes an inverse Fourier transform.

15.5.16. propfactor(nc, p)

Default: xpropfactor()

Returns the interaction picture propagator used by the prop function. The time propagated is a fraction of the current integration time-step, dt. It is equal to $1/ipsteps$ of the integration time-step. It uses data from the **linear** function to calculate this.

15.5.17. randomgen(p)

Default: xrandom()

Generates a cell array of initial random fields v to initialize the fields simulated. The default, **xrandomgen**, returns Gaussian real fields that have inrandoms{n}(1) components delta-correlated in space, with inrandoms{n}(2) delta-correlated in momentum space. The default uses a user-defined cell array, rfilter{n}, of filter functions, to modify correlations in momentum space, if specified.

15.5.18. rfilter{n}(w, p)

Default: xrfilter()

Returns the momentum-space filter function for the momentum-space random terms. Each component has an array dimension the same as the input random fields in momentum space, that is, the return dimension of cell n is [inrandoms{n}(2),... nlattice].

15.5.19. transfer{c}(a0{:},v{:},p)

Default: xtransfer()

This function initializes sequential simulations, where the previous field a0 can be used as an input to the next stage in the integration sequence. The default, xtransfer(), takes each output, a0{c} of the previous simulation to initialize the field $a\{c\}$. Otherwise, this function is identical to initial().

15.6. xSIM support functions

These general xSIM support functions are used to define the differential equations and averages. They all start with a capital letter. Algorithms are documented in section 13. Fields can be differentiated or integrated only in space, observables in space or time.

15.6.1. Ave(o[, av], p)

This function takes a field or observable and returns an average over one or more dimensions. The input includes an optional averaging switch *av*. If $av(j) > 0$, an average is taken over dimension j. If the av vector is omitted, the average is taken over all space directions.

15.6.2. Bin(o[, dx], p)

The *Bin* function takes a field o and returns probabilities on space axes that are defined by a vector dx. This allows binning of position probabilities if the observable is a mean position that is plotted on an axis. If j is the first index with $dx(j) > 0$, the binning is taken over dimension j. The results returned are the probability of o in the bin, normalized by $1/dx\,(j)$. If the input array is Fourier transformed, by using the transforms attribute in the observe function, then one must set $dx(j) = p.dk(j)$ for transformed dimensions j. If the dx vector is omitted, or a scalar dx is used, the binning is over the first space direction.

15.6.3. D1(a[, d,c,ind], p)

Takes a scalar or vector field a and returns a derivative in dimension d using central finite differences. Set $d = 2$ for an x-derivative, $d = 3$ for a y-derivative, etc. The default value is $d = 2$. The cell index is the index of the cell that is differentiated, which is needed when there is more than one cell with different boundary types. An index list *ind* can be included to take a derivative of one component or a specified list. If omitted, derivatives of all components are returned.

Boundary values are stored in p.boundval, and are needed for Neumann/Robin boundaries. Hence, in multi-cell computations, if a is from a cell index $c > 1$, the cell index c must be included to identify which boundary value to use. If there are Neumann/Robin boundaries, the entire field a must be input unless all boundary values are the same.

For other types of boundaries, the cell index is not needed, and D1 can differentiate a single field component without having to identify the component. The method is of second order in the space step. It is used in the deriv, observe and output functions, with automatic compensation for the presence/absence of a time index.

For Dirichlet boundaries, the derivatives are ambiguous at the boundaries, and a periodic derivative is returned. This is not needed for time-evolution, as the boundary value overrides it.

15.6.4. D2(a[, d,c,ind], p)

This takes a scalar or vector field a and returns the second derivative in dimension d using central finite differences. Other properties are the same as D1(). The method is of second order in the space step, except at a Neumann/Robin boundary, where the boundary result is of first order.

15.6.5. DS(a[,order,d,c,ind], p)

This takes a scalar or vector field a and returns the spectral derivative of a given order in dimension d. Other properties are the same as D1(). The method uses Fourier, sine or cosine transforms to compute derivatives on an equally spaced grid, depending on the boundary conditions used. For Fourier transforms, with periodic boundaries, any integer order can be used. With non-periodic boundaries, only even orders are available, and use is restricted to the midpoint (MP) method.

15.6.6. Int(o[, dx, bounds,c], p)

This function takes any vector or scalar field or observable and returns a space integral over selected dimensions with vector measure dx. If $dx(j) > 0$, dimension j is integrated. Time integrals are only possible for observables. Space dimensions are labelled from j = 2,3,...dimensions. To integrate over the lattice, set dx = p.dx, otherwise set dx(j) = p.dx(j) for integrated dimensions and dx(j) = 0 for non-integrated dimensions.

If the input array is Fourier transformed by using the p.transforms attribute, **one must set dx(j) = p.dk(j) for transformed dimensions j, to get correct results**. If the dx vector is omitted, the integral is over all available space dimensions, assuming no Fourier transforms.

The optional input bounds is an array of size [p.dimensions,2], which specifies lower and upper integration bounds in each direction. This is only available if dx or dk is input. If omitted, integration is over the whole domain. The optional input switch $c$is used to indicate that the input should be reshaped to the implicit shape of cell c.

15.7. Quantum support functions

The following xSIM predefined support functions are available for toolbox applications fpr quantum Monte-Carlo and phase-space simulations:

15.7.1. Bosonic operator functions

These are support functions that are permanently available.

Label	Inputs	Output(s)
A	(m, psi)	$\hat{a}_m \|\psi\rangle$
A2	(m, psi)	$\hat{a}_m^2 \|\psi\rangle$
N	$([m_1\,(,m_2)], psi)$	$\hat{a}_{m_1}^\dagger \hat{a}_{m_2} \|\psi\rangle$
BoseHubbardH	(O, K, p, psi)	Bose-Hubbard Hamiltonian
Mkbose	$([list,]p)$	Makes sparse bosonic operator matrices
Mknumber	(nv, p)	Makes number states with occupations nv
MkBoseHubbard	(O, K, p)	Makes Bose-Hubbard Hamiltonian

Sparse bosonic operator matrices Requires *Mkbose* to create these sparse matrix operators, which are temporarily created as a cell array when required, since they use up memory. These are not support functions, they are temporary matrices, but are required by *MkBoseHubbard* to define the Hamiltonian. Otherwise, any label can be used.

Label	Inputs	Output(s)
p.a	$\{m\}$	$\hat{a}_m$
p.a'	$\{m\}$	$\hat{a}_m^\dagger$

15.7.2. Qubit and Pauli spin operator functions

These are support functions that are permanently available.

Label	Inputs	Output(s)
Sx	(m, psi)	$\hat{\sigma}_m^x \|\psi\rangle$
Sy	(m, psi)	$\hat{\sigma}_m^y \|\psi\rangle$
Sz	(m, psi)	$\hat{\sigma}_m^z \|\psi\rangle$
Sx2	$([m_1\,(,m_2)], psi,)$	$\hat{\sigma}_{m_1}^x \hat{\sigma}_{m_2}^x \|\psi\rangle$
Sy2	$([m_1\,(,m_2)], psi)$	$\hat{\sigma}_{m_1}^y \hat{\sigma}_{m_2}^y \|\psi\rangle$
Sz2	$([m_1\,(,m_2)], psi)$	$\hat{\sigma}_{m_1}^z \hat{\sigma}_{m_2}^z \|\psi\rangle$

15.7.3. Quantum logic gate operators

These are support functions that are permanently available.

Label	Inputs	Output(s)
Ha	(m, psi)	$h \|\psi\rangle$
Ph	(m, psi)	$p \|\psi\rangle$
P8	(m, psi)	$t \|\psi\rangle$
Cx	$([m_1\,(,m_2)], psi,)$	$cx \|\psi\rangle$

15.7.4. GBS phase-space support functions

The main observe suppport functions correspond to binned photon-counting probability distributions and marginal moments of Gaussian boson sampling operator observables.
:

Label	Return type	Description
X	vector	x-quadrature per channel
X2	vector	x-quadrature squared per channel
Y	vector	y-quadrature per channel
Y2	vector	Y-quadrature squared per channel
Pn	vector	Output photon number, $\hat{n}'_j$, per channel
Nm	vector	Output photon number correlation in sequence $\hat{n}'_1\hat{n}'_2\ldots$
K	vector	Clicks $\hat{\pi}_j(1)$ per channel
Km	vector	Click correlation in sequence $\hat{\pi}_1(1)\hat{\pi}_2(1)\ldots$
Km2	vector	Click correlation over two channels $\hat{\pi}_j(1)\hat{\pi}_k(1)$
Km3	vector	Click correlation over three channels $\hat{\pi}_j(1)\hat{\pi}_k(1)\hat{\pi}_h(1)$
Kmsub	vector	Subset of click correlations per CO channels
K1	vector	Binned click probability - single partition
N1	vector	Binned photon number probability - single partition
Kn	array	Binned click probability - n-fold partition
Nn	array	Binned photon number probability - n-fold partition

15.7.5. GBS compare functions

The compare function is used for testing. Functions labeled with an asterisk can be used with Gaussian states having a coherent component *p.alpha*:

Label	Return type	Description
Xc	vector	x-quadrature per channel
X2c	vector	x-quadrature squared per channel
Yc	vector	y-quadrature per channel
Y2c	vector	y-quadrature squared per channel
*Pnc	vector	Output photon number, $\hat{n}'_j$, per channel
Nmc	vector	Output photon number correlation in sequence $\hat{n}'_1\hat{n}'_2\ldots$
Kc	vector	Clicks $\hat{\pi}_j(1)$ per channel
Kmc	vector	Click correlation in sequence $\hat{\pi}_1(1)\hat{\pi}_2(1)\ldots$
Km2c	vector	Click correlation over two channels $\hat{\pi}_j(1)\hat{\pi}_k(1)$
Km3c	vector	Click correlation over three channels $\hat{\pi}_j(1)\hat{\pi}_k(1)\hat{\pi}_h(1)$
Kmsubc	vector	Subset of click correlations per CO channels
K1c	vector	Binned click probability - single partition
N1tc	vector	Thermal input photon number probability - single partition
N1lsc	vector	Pure squeezed input photon number probability - single partition
Knc	array	Binned click probability - n-fold partition
Nntc	array	Thermal input photon number probability - n-fold partition

15.8. Internal xSIM parameters

Knowing the details of array indexing inside xSPDE becomes important if you want to write your own functions to extend xSPDE, interface xSPDE with other functions, or read and write xSPDE data files with external programs. It also helps to understand how the program works.

15.8.1. Array tables

There are two main internal xSPDE arrays: fields labelled a and output data labelled d. The fields contain stochastic variables, the data contains the averaged outputs and errors estimates.

Important array and index definitions are:

Label	Indices	Description
a	$\{n\}\,[f,\mathbf{i},e_1]$	Stochastic field array
v	$\{n\}\,[m_1,\mathbf{i},e_1]$	Initial random variable array
w	$\{n\}\,[m_2,\mathbf{i},e_1]$	Noise field array
r{2},k{2}....	$(1,\mathbf{i},1)$	Numbered space/momentum coordinates
x,y,z,kx,ky,kz	$(1,\mathbf{i},1)$	Labelled space/momentum coordinates
ave	$\{o\}(\ell,\mathbf{j})$	Cell array of all observed averages
d	$\{s\}\{o\}(\ell,\mathbf{j},c)$	Cell array of output data with checks
raw	$\{s,c,h\}\{n\}(f,\mathbf{j},e_1)$	Raw trajectories
points	$[pt_1,pt_2\dots pt_d]$	Vector of lattice sizes
ensembles	$[h_1,h_2,h_3]$	Vector of ensemble sizes

Here:

- n is the cell index of a computational field or noise
- f is the field internal index
- $\mathbf{i}$ is the space index
- e_1 is the first ensemble index
- c is the check index for all error checking
- $\mathbf{m}$ is the random or noise index
- $\mathbf{j}=[j_1,\mathbf{i}]$ is the space-time index
- s is the sequence index
- o is the cell index of an observe and/or output function
- ℓ is the line index of an output
- e is the high-level ensemble index (combines e_2,e_3 indices)

When fields are passed to observe or to raw outputs, the defined auxiliary fields are included as well. Apart from the internal field dimension(s), the common dimensionality for internal arrays used in computations is $[d.space, ensembles(1)]$. The number of points in *d.space* can change depending on the cell index, for different integrated fields.

15.8.2. Simulation data in xSIM

In xSIM, the space-time dimension d is unlimited. xGRAPH can plot up to three chosen axes. All fields are stored in cell arrays that contain real or complex numerical arrays. Average results are stored are stored in cell arrays of real numerical arrays, usually of rank $2+d$, although this can change in special cases like the plot of a probability, which requires extra axes.

The array index ordering in xSPDE integrated fields is $\{n\}(\boldsymbol{f},\mathbf{i},e_1)$, where:

- The internal cell index n labels distinct integrated variables.
- The internal field index $\boldsymbol{f}$, is a field index or indices, not including auxiliary fields
- The next $d-1$ indices are $\mathbf{i}$, which is a space index with no time index.
- The last is an ensemble index e_1, to store low-level parallel trajectories.

The array index ordering in graphical averaged output data is $(\ell, \mathbf{j}, c)$, where:

- The first index is a line index ℓ.
- The next d indices are $\mathbf{j} = [j_1, \ldots j_d] = [j, \mathbf{i}]$, for time and space.
- The last is a check index c, for combined errors and comparisons

Stored data uses heterogenous cell arrays to package numerical arrays with additional high level indices. The first cell index is the sequence index, s. Inside each sequence, data cell arrays have a graph index n. This distinguishes the different averages generated for output graphs and data. Raw data has cell indices for the sequence, time-step and high level ensembles.

In summary, the xSPDE internal arrays are as follows:

- **Field** arrays $a\{n\}(\boldsymbol{f}, \mathbf{i}, e)$ - these have a field index, a space index and low-level ensemble index e.
- **Auxiliary** arrays $a_x\{n\}(\boldsymbol{f}, \mathbf{i}, e)$ - these are appended to the field cells for raw data and observables.
- **Random** and **noise** arrays $w\{n\}(m, \mathbf{i}, e)$ - these are initial random fields or noise fields. The first index may have a different range to the field index.
- **Coordinate** arrays $x(1, \mathbf{i})$ - these contain the coordinates at grid-points, with labels x, y, z, and $j_1 = 1$. Numeric labels $x\{l\}$ are used for $d > 4$, where $l = 2, \ldots d$. The same sizes are used for:
  - momentum coordinates kx, ky, kz (alternatively $k\{2\}, k\{3\}, \ldots$)
  - spectral derivative arrays Dx, Dy, Dz (alternatively $D\{2\}, D\{3\}, \ldots$) .
- **Raw data** arrays $r\{s, c, e\}\{n\}(\boldsymbol{f}, \mathbf{j}, e_1)$ - these are cell arrays of generated trajectories, including integrated and defined field values. They are optional, as they use large amounts of memory. These are saved in cell arrays with indices s for the sequence, c for the time-step error-check and h for high level ensemble index. The cell indices are:
  - $s = 1, \ldots S$ for the sequence number,
  - $c = 1, 2, 3..$ for the error-checking step used: first fine, then coarse in each dimension checked, then any iteration errors

  - $e = 1, \ldots ensembles(2) * ensembles(3)$ for a high level parallel and serial ensemble index.

- **Average** arrays $ave\{o\}(\ell, \mathbf{j})$ - these are generated in xSIM by the observe functions, then used to store generated average data at all time points. The cell index n is the observe index, which indexes overs the observe functions.The internal index ℓ is a line index generated by an observe function.

- **Data** arrays $d\{s\}\{o\}(\ell, \mathbf{j}, c))$ - these store the final results. The $\mathbf{j}$ indices may be Fourier indices if transforms are specified, and may include extra axes for probabilities. If this data is modified by an xSIM output function, the data index s equals the relevant output function index.
  Check indices are used for error estimates and comparisons, where $c = 1$ for the average, $c = 2$ for the systematic error, and $c = 3$ for the sampling error. The systematic error is a composite of all step and convergence errors that are checked. If there is comparison data, it uses $c = 4$ up to $c = 6$, to allow for any error bars. The output data uses cell indices $\{s\}$ for the sequence index, and $\{o\}$ for the data index. The default index corresponds to the observe function, but if there are output functions, those indices are used.

15.8.3. Internal parameter table

The internal parameter structures in xSPDE are available to the user if required. Internally, all xSPDE parameters are stored in the parameter structures passed to functions. This includes the data given above from the input structures. In addition, it includes the computed parameters given below, which includes internal array dimensions.

When accessing these in a function, prefix them by the structure label, usually $p.$ in the examples, eg, $p.t.$ Where the space points change with the cell, the labels below refer to properties of the first cell index. Fields, noises and random fields have a cell index when stored internally, but are passed to user functions as arrays, in order of the index. Spatial cell volumes are reduced if there are multiple spatial steps for increased spatial resolution.

Data in $k-$space is stored in two alternative lattices, each having their own axis vectors. The propagation grid is used while propagating, and is compatible with numerical FFT conventions where the first index value is $k = 0$. The graphics grid is centered around $k = 0$, and is used for graphics and data storage, following scientific conventions.

For more than four total dimensions, the spatial grid, momentum grid and derivative grid notation of t, x, y, z, ω, kx, ky, kz and Dx, Dy, Dz is changed to use numerical labels that correspond to the dimension numbers, i.e., $D\{2\}, \ldots D\{d\}$, $r\{1\}, \ldots r\{d\}$, $k\{1\}, \ldots k\{d\}$.

Numeric dimension labeling can also be used even for lower dimensionality if preferred.

Label	Type	Typical value	Description
t, x, y, z	array	-	Space-time grid of t, x, y, z
ω, kx, ky, kz	array	-	Frequency-momentum grid of k_x, k_y, k_z
Dx, Dy, Dz	array	-	Derivative grid of D_x, D_y, D_z
$r\{1\}, \ldots r\{d\}$	array	-	Space-time grid of $r_1, \ldots r_d$
$k\{1\}, \ldots k\{d\}$	array	-	Graphics momentum grid of $k_1, \ldots k_d$
$D\{2\}, \ldots D\{d\}$	array	-	Derivative grid of $D_2, \ldots D_d$
dx	vector	[0.2,..]	Steps in $[t, x, y, z]$
dk	vector	[0.61,....]	Steps in $[\omega, k_x, k_y, k_z]$
dt	double	0.2000	Output time-step
dtr	double	0.1000	Computational time-step
v	real	1	Spatial lattice volume
kv	real	1	Momentum lattice volume
dv	real	1	Spatial cell volume
dkv	real	1	Momentum cell volume
xc{d}	cells of vectors	[-5,... 5]	Coordinate axes in t, x, y, z
kc{d}	cells of vectors	[-5,..5]	Momentum axes in$[\omega, k_x, k_y, k_z]$
nspace	integer	35	Number of spatial lattice points
inrandoms	vector	{2}	Initial random fields per cell
krandoms	vector	{2}	Initial random fields in kspace per cell
noises	vector	{2}	Number of noise fields per cell
knoises	vector	{2}	Noise fields in kspace per cell

15.9. Examples, testing and structure

Additional examples are given in the Examples folder distributed with xSPDE. These can all be run using Batchtest.m, which has a typical runtime of $50 - 100s$, and runs 42 different case studies. This shows your distribution is intact. All the graphs produced are deleted. It lists the different examples available, some of which are given below.

The batch testing code will run each different example sequentially. It prints the RMS relative errors for the step-size, sampling and difference error, as well as the total RMS error combining all three, the chi-square error normalized by the number of points, and the timing. The geometric mean of the 42 RMS total errors is computed as a benchmark.

As Matlab random noise is reproducible with a fixed seed, this geometric mean error is fixed. The total is printed to more than six decimals for verification, and an error is indicated if it varies by a factor of more than $\pm 10^{-3}$. Due to different random noise algorithms used in some Octave versions, the Octave error may vary by up to $\pm 20\%$.

15.9.1. xSPDE structure

The control program, *xspde*, calls the xsim integration and xgraph graphics functions successively

$$\mathbf{xspde} \rightarrow \begin{cases} \mathbf{xsim}\ (simulations) \\ \mathbf{xgraph}\ (graphics) \end{cases}$$

For convergence checking, a useful alternative to xspde which repeats the calculation checks times while halving the time-step each time, and reports the resulting errors for averaged observables, is:

- xcheck (checks,p)

15.9.2. xSIM

The integration function, *xsim*, generates all data. It first carries out elementary checks in xpreferences and constructs the grid of lattice points in xlattice. Then it generates the nested ensembles in xensemble, and integrates each subensemble using xpath. The output data is written to files, if required, in xwrite.

$$\begin{aligned} \mathbf{xsim} &\rightarrow \mathbf{xpreferences} \rightarrow \mathbf{xlattice}\ (checks\, inputs) \\ &\rightarrow \mathbf{xensemble} \leftrightarrow \mathbf{xpath} \leftrightarrow \mathbf{xdata}\ (simulates) \\ &\rightarrow \mathbf{xwrite}\ \ (stores\, data) \end{aligned}$$

16. Graphics parameters

This chapter gives a reference guide to the xGRAPH parameters and functions.

16.1. xGRAPH overview

The graphics function provided is a general purpose multidimensional batch graphics code, xGRAPH, which is automatically called by xSPDE when xSIM is finished. The results are graphed and output if required. Alternatively, xGRAPH can be replaced by another graphics code, or it can be used to process the data generated by the xSIM function at a later time.

The xgraph function call syntax is:

- **xgraph (data [,input]**)

This takes simulation data and input cell arrays, then plots graphs. The data should have as many cells as there are input cells, for sequences.

If data = 'filename.h5' or 'filename.mat', the specified file is read both for input and data. Here .h5 indicates an HDF5 file, and .mat indicates a Matlab file.

When the data input is a filename, parameters in the file can be replaced by new input parameters that are specified. Any stored input in the file is then overwritten when graphs are generated. This allows graphs of data to be modified retrospectively, if the simulation takes too long to be run again in a reasonable timeframe.

16.1.1. Parameter and data structures

This is a batch graphics function, intended to process quantities of graphics data, input as a cell array of multi-dimensional data. Theoretical and/or experimental data is passed to the graphics program, including the complete data cell array and a cell array of graphics parameters for plotting each graph.

To explain xGRAPH in full detail,

- Data to be graphed are recorded sequentially in a cell array, with data={d1,d2,...}.
- Graphics parameters including defaults are given in the input cell array.
- This describes a sequence of graph parameters, so that input={p1,p2,...}.
- For a one member sequence, a dataset and parameter structure can be used on its own.

- Each dataset and parameter structure describes a set of graphs.

The data input to xGRAPH can either come from a file, or from data generated directly with xSIM. The main graphics data is a nested cell array. It contains several numerical graphics arrays. Each defines one independent set of averaged data, the observed data averages, stored in a cell array indexed as $data\{s\}\{n\}(\ell, \mathbf{j}, c)$. To graph these also requires a corresponding cell array of structures of graphics parameters.

The output is unlimited, apart from memory limits. The program also generates error comparisons and chi-squared values if required. The data structure for input is as follows:

1. The input data is a cell array of datasets, which can be collapsed to a single dataset
2. The parameters are also a cell array of parameter structures, which can be collapsed to one structure
3. The dataset is a cell array of multidimensional graphs, each with arbitrary dimensionality.
4. The first or line index of each graph array allows multiple lines, with different line-styles
5. The last or check index of each graph array is optionally used for error and comparison fields.
6. Each graph array can generate multiple graphic plots, as defined by the parameters.

16.2. Parameter table

The complete cell array of the simulation data is passed to the xGRAPH program, along with graphics parameters for each observable, to create a graphics data structure. Graphics parameters have default values which are user-modifiable by editing the xg-preferences function. There is some duplication between the graphics and simulation parameters.

Some input parameters are global parameters for all graphs. However, most xGRAPH parameters are cell arrays indexed by graph index. These graphics parameters are individually set for each output that is plotted, using the cell index $\{n\}$ in a curly bracket. If present they replace the global parameters like labels.

If a graph index is omitted, and the parameter is not a nested array, the program will use the same value for all graphs. The axes, glabels, legends, lines, logs, and xfunctions of each graph are nested cell arrays, as there can be any number of lines and axis dimensions. In the case of the logs switch, the observable axis is treated as an extra dimension.

The plotted result can be an arbitrary function of the generated average data, by using the optional input gfunction. If this is omitted, the generated average data that is input is plotted.

Comparisons are plotted if present in the input data indexed by the last or check index c, with $c > errors$, where $errors = 3$ is the usual maximum value.

A table of the graphics parameters is given below.

Label	Default value	Description
chisqplot{n}	0	Chi-square plot options
cutoff	1.e-12	Global lower cutoff for chi-squares
cutoffs{n}	cutoff	Probability cutoff for n-th graph
diffplot{n}	0	Comparison difference plot options
errors	0	Index of last error field in data
esample{n}	1	Size and type of sampling error-bar
font{n}	18	Font size for graph labels
gfunction{n}	@(d,˜) d{n}	Functions of graphics data
glabels{n}	{'t' ,'x' ,'y' ,'z'}	Graph-specific axis labels
graphs	[1 : *max*]	Vector of all the required graphs
gsqplot{n}	0	G-square (likelihood) plot options
headers{n}	”	Graph headers
images{n}	0	Number of movie images
imagetype{n}	0	Type of 3D image
klabels	{'\omega' ,'k_x' ,'k_y' ,'k_z'...}	Global transformed axis labels
legends{n}	{'label1',..}	Legends for multi-line graphs
limits{n}	{[lc1,uc1],[lc2,uc2]}	Axis limits, first lower then upper
linestyle{n}	{'-',..}	Line styles for multiline 2D graphs
linewidth{n}	0.5	Line width for 2D graphs (in points)
logs{n}	{0,..}	Axis logarithmic switch: 0 linear, 1 log
minbar{n}	0.01	Minimum relative error-bar
mincount	10	Global counts for chi-square cutoffs
name	”	Global graph header
numberaxis	0	Switch for numbering the graphics axes
octave	0	0 for Matlab, 1 for octave environment
olabels{n}	'a_1'	Observable labels
pdimension{n}	3	Maximum plot dimensions
savefig	0	Switch, set to 1 to save figure files
savegraph	0	Set '.pdf', '.pdf', '.png', '.jpg' or '.tif' to save
scale{n}	1	Scaling: Counts/ probability density
transverse{n}	0	Number of transverse plots
xfunctions{n}	{@(t,˜) t,@(x,˜) x,..}	Axis transformations
xlabels	{'t' ,'x' ,'y' ,'z'...}	Global axis labels
verbose	0	0 for brief, 1 for informative, 2 for full output

- Up to 6 types of input data can occur, including errors and comparisons, indexed by the last index. The original mean data always has c =1. If there are no errors or comparisons, one graph is plotted for each dimensional reduction.

- The input data has up to two error bars (I and II), and optional comparisons also with up to two error bars.
- Type I errors labeled $c = 2$ have standard vertical error bars. Type II errors labeled $c = 3$, which are usually standard deviation errors from sampling, have two solid lines.
- If esample = -1, both error bars are combined and the RMS errors are plotted as a single error bar.
- If $diffplot > 0$, differences are plotted as unnormalized ($diffplot = 1$), or normalized ($diffplot = 2$) by the total RMS errors. If $diffplot = 3$, raw comparison data is plotted.
- When differences are plotted, the total comparison errors are treated as type I error bars, while total simulation errors are treated as type II errors with parallel lines in the graphs, in order to distinguish them.

Saved graphics files appear in the Matlab current directory. A detailed description of each parameter is listed in Sec (16.3).

16.2.1. Example

A simple example of data and input parameters, but without errors or comparisons is as follows

```
p.name = 'Sine and cosine functions';
p.olabels = {'sine(m_1\pi/100)','cosine(m_1\pi/100)'};
data = {sin([1:100*pi]/100),cos([1:100*pi]/100)};
xgraph(data,p);
```

Note that in this case the default setting of p.errors=0 is used, with no check index used in the data arrays, because these are simple graphs without error-bars or comparisons.

16.2.2. xGRAPH data arrays

The data input to xGRAPH can come from a file, or from data generated directly from any compatible program.

The data is stored in a cell array *data* with structure:

$$data\{s\}\{n\}(\ell,\mathbf{j},c)$$

Each member of the outer cell array data{s} defines a number of related sets of graphical data, all described by common parameters input{s}. Comparisons and errors are plotted if there are errors and comparison data in the input, indexed by c. This generates comparison plots, as well as error totals and χ- squared error estimate when there are statistical variances available.

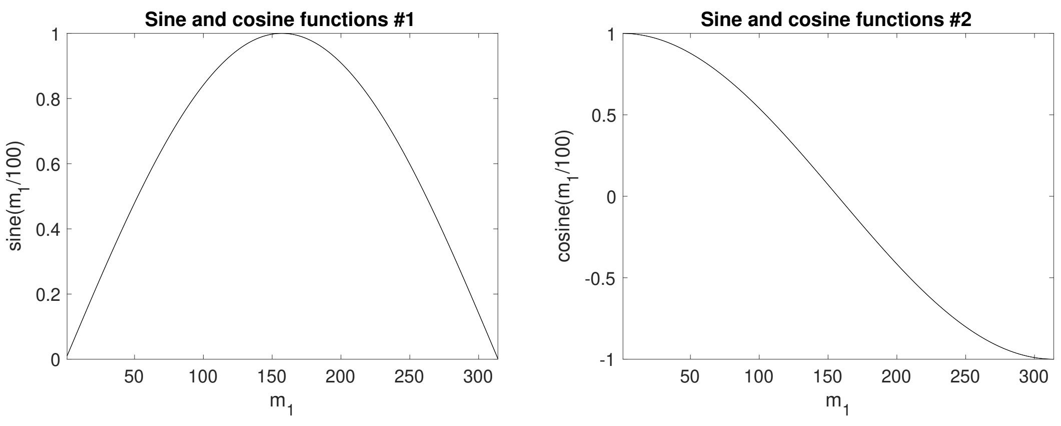


Figure 16.1.: *Example: xgraph output of two plots.*

An individual member of data{s}{n} is a multidimensional array, called a graph in the xSPDE User's guide. For each graph, multiple different plots with different dimensionality can be obtained from the dataset data{s}{n}, either through projections and slices or by generating additional data defined with graphics functions. Either or both alternatives are available.

Note that:

- If a sequence has one member, the outer cell array can be omitted.
- In this simplified case, if there is only one graph array, the inner cell array can be omitted.

The graphics data for a single dataset is held in a multidimensional real array, where:

- ℓ is the index for lines in the graph. Even for one line, the first dimension is retained.
- $\mathbf{j} = j_1, \ldots j_d$ is the array index in each dimension, where $d \geq 1$.
- Averages in momentum space have the momentum origin as the central index.
- If integrals or spatial averages are used, the corresponding dimension has one index $j_d = 1$.
- With probabilities, extra dimensions are added to $\mathbf{j}$ to store the bin indices.
- c indexes error-checks and comparisons. If not present, omit p.errors and the last dimension.
- If $c > p.errors$, the extra fields are comparison inputs, where $p.errors$ is the largest data index.

When the optional comparison fields are used, an input parameter $errors$ is required to indicate the maximum error index, to distinguish data from comparisons. Parameter structures from xSIM have $errors = 3$ set to allow for both sampling errors and discretization errors. If this is omitted, the default is $errors = 0$, which implies that there is no error or comparison data

If $errors > 0$, the last index can have larger values with $c > errors$, for comparisons. The special case of $errors = 1$ is used if the data has no error bars, but there are comparisons in the data. Larger indices are used to index the comparison data, which can also have two types of errors. The largest usable last index is $errors + 3$.

It is possible to directly plot the raw data using xGRAPH. One can even combine the raw data with a graphics parameter input. But since the raw data has no error estimates - it is raw data - one must set $p.errors = 0$, since the xsim output parameters have a normal setting of $p.errors = 3$. This will give a single trajectory.

However, the raw data from a simulation typically includes many trajectories if $ensembles(1) > 0$. One must select particular trajectory datasets from the raw cell array, to plot just one.

16.2.3. Input parameters and defaults

A sequence of graph parameters is obtained from inputs in a cell array, as input = {in1, in2, ...}. The input parameters of each simulation in the sequence are specified in a Matlab structure. The inputs are numbers, vectors, strings, functions and cell arrays. All metadata has preferred values, so only changes from the preferences need to be input. The resulting data is stored internally as a sequence of structures in a cell array, to describe the simulation sequence.

The graphics parameters are also stored in the cell array input as a sequence of structures p. This only need to be input when the graphs are generated and can be changed at a later time to alter the graphics output. A sequence of simulations is graphed from input specifications.

If there is one simulation, just one structure can be input, without the sequence braces. The standard way to input each parameter value is:

$$p.label = parameter$$

The standard way to input a function handle is:

$$p.label = @function$$

The inputs are scalar or vector parameters or function handles. Quantities relating to graphed averages are cell arrays, indexed by the graph number. The available inputs, with their default values in brackets, are given below.

Simulation metadata, including default values that were used in a particular simulation, can be included in the input data files. This is done in both the .mat and the .h5 output files generated by xSIM, so the entire graphics input can be reconstructed or changed.

Parameters can be numbers, vectors, strings or cell arrays. Conventions that are used are that:

- All input parameters have default values
- Vector inputs of numbers are enclosed in square brackets, [...].
- Cell arrays of strings, functions or vectors are enclosed in curly brackets.
- Vector or cell array inputs with only one member don't require brackets.
- Incomplete parameter inputs are completed with the last used default value.
- Function definitions can be handles pointing elsewhere, or defined inline.

If any inputs are omitted, there are default values which are set by the internal function xgpreferences. The defaults can be changed by editing xgpreferences.

In the following descriptions, graphs is the total number of graphed variables of all types. The space coordinate, image, image-type and transverse data can be omitted if there is no spatial lattice, that is, if the dimension variable is set to one.

For uniformity, the graphics parameters that reference an individual data object are cell arrays. These are indexed over the graph number using braces {}. If a different type of input is used, like a scalar or matrix, xSPDE will attempt to convert the type to a cell array.

Axis labels are cell arrays, indexed over dimension. The graph number used to index these cell arrays refers to the data object. In each case there can be multiple generated plots, depending on the graphics input.

16.2.4. Cascaded plots

The xGRAPH function generates a default range of graphs, but this can be modified to suit the user. In the simplest case of one dimension, one graph dataset will generate a single plot. For higher dimensions, a cascade of plots is generated to allow visualization, starting from 3D movies, then 3D static plots and finally 2D slices. These can also be user modified.

Note that for all probabilities, the plot dimension is increased by the bin range dimensionality.

16.2.5. Plot dimensions

The pdimension input sets the maximum plotted dimensions. For example, $pdimension\{1\} = 1$ means that only plots vs r_1 are output for the first function plotted. Default values are used for the non-plotted dimensions, unless there are axes specified, as indicated below.

The graphs cascade down from higher to lower dimensions, generating different types of graphs. Each type of graph is generated once for each function index.

16.2.6. Plot axes

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 graph or set of generated graph data.

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

Examples:

- $axes\{1\} = \{0\}$ - For function 1, plot all the first dimensional points; higher dimensions get defaults.
- $axes\{2\} = \{-2, 0\}$ - For function 2, plot the maximum value of r_1 (the default) and all higher-dimensional x-points.
- $axes\{3\} = \{1 : 4 : 51, 32, 64\}$ - For function 3, plot every 4-th x_1 point at x_2 point 32, x_3 point 64
- $axes\{4\} = \{0, 2 : 4 : 48, 0\}$ - For function 4, plot every x_1 point , every 4-th x_2 point, and all x_3-points.

Points labelled -1 indicates a default 'typical' point, which is the midpoint. If one uses -2, this is the last point.

Lower dimensions are replaced by corresponding higher dimensions if there are dimensions or axes that are suppressed. Slices can be taken at any desired point, not just the midpoint. The notation of $axes\{1\} = \{6 : 3 : 81\}$, is 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, which is equal to the actual input data dimension unless there is an axes suppression, described above. Since the plot has to include a data axis, the plot itself will usually have an extra data axis.

One can plot only three axes directly using standard graphics tools. The strategy to deal with the higher effective dimensionality is as follows. For simplicity, "time" is used to label the first effective dimension, although in fact any first dimension is possible:

dimensions = 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.

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

dimensions = 3 For three lattice dimensions, if *images* > 1, a movie of distinct 3D graphic images of observables are plotted as *images* slices versus the first plot dimension. Otherwise, a slice through the chosen point, is used at the highest dimension to reduce the lattice dimension to 2.

dimensions = 4,5.. For higher lattice dimensions, a slice through a chosen point, or the default midpoint is used to reduce the lattice dimension to 3.

As explained above, in addition to graphs versus x_1 the **xGRAPH** function can generate images (3D) and transverse (2D) plots at specified points, up to a maximum given by the number of points specified. The number of these can be individually specified for each graph number. The images available are specified as imagetype= $1, \ldots 4$, giving:

1. 3D perspective plots (Matlab surf - the default)
2. 2D filled color plots (Matlab contourf)
3. contour plots (Matlab contour)
4. pseudo-color plots (Matlab pcolor)

Error bars, sampling errors and multiple lines for comparisons are only graphed for 2D plots. Error-bars are not plotted when they are below a user-specified size, with a default of 1% of the maximum range, to improve graphics quality. Higher dimensional graphs do not output error-bar data, but they are still recorded in the data files.

16.2.7. Probabilities and parametric plots

Probability data can be input and plotted like any other data. It is typically generated from simulation programs using the *binranges* data for binning. It is plotted like any other graph, with any dimension, except that the total dimension is extended by the number of variables or lines in the observe function.

16.2.8. Chi-squared plots

In addition the program can make a χ^2 plot, which is a plot of the χ^2 comparison with a comparison probability density against space and/or time. This allows a test of the simulated data against a known target probability distribution, provided that the following input data conditions are satisfied:

- The input data dimension exceeds the p.dimensions parameter,
- The switch p.chisqplot is set to 1or 2, and
- The input data includes comparison function data.

The χ^2 plots, depending on *p.chisqplot* are:

1. a plot of χ^2 and k, where k is the number of valid data points,

2. a plot of $\sqrt{2\chi^2}$ and $\sqrt{2k-1}$, which should have a unit variance.

Here, for one point in space and time, with m bins, N_j counts per bin and E_j expected counts:

$$\chi^2 = \sum_{j=1}^{m} \frac{(N_j - E_j)^2}{E_j}. \tag{16.1}$$

The number k is the number of valid counts, with $N_j, E_j > mincount$. This is partly determined from the requirement that the probability count data per bin is greater than the *p.mincount* parameter. The default is set to give a number of samples > 10. The program prints a summary that sums over of all the χ^2 data.

The *p.scale*$\{n\}$ parameter gives the number of counts per bin at unit probability density. This is needed to set the scale of the χ^2 results, ie, $N_j = scale\{n\} \times p_j$, where p_j is the probability density that is compared and plotted in the simulation data. Note that a uniform bin size is assumed here, to give a uniform scaling.

16.2.9. Comparisons with variances

It can be useful to compare two probability distributions with different variances. For one point in space and time, with m bins, p_j probability density and e_j expected probability density,

$$\chi^2 = \sum_{j=1}^{m} \frac{(p_j - e_j)^2}{\sigma_j^2 + \sigma_{e,j}^2}. \tag{16.2}$$

In this case, σ_j^2 and $\sigma_{e,j}^2$ are the sampling errors in the simulation data and comparison data, so that built-in error fields in the data are used to work out the χ^2 results. This option is chosen if *p.scale*$\{n\} = 0$, and the cutoff for the data is then specified so that $p_j, e_j > p.cutoffs\{n\}$. The default value is the global cutoff, $p.cutoff$, which has a default of 10^{-12}.

This output only has a χ^2 distribution with $\chi^2 \approx m$ if all the points are independent. The measured χ^2/m includes all the space-time points above the cutoff value. This can be less than unity when comparing an expected exact result with a computed SDE solution where data is often correlated. At the other extreme, if the cutoff is too low, the data may not be reliable, and one can obtain too large a value.

16.2.10. Maximum likelihood

It is also possible to plot the G^2 or maximum likelihood plot of the data, which is an alternative means to compare distributions, where

$$G^2 = 2\sum_{j=1}^{m} N_j \ln\left(N_j/E_j\right). \tag{16.3}$$

The expected values E_j are automatically scaled so that $\sum N_j = \sum E_j$,with the same minimum count cutoff that is used for the χ^2 data. The result is similar to the χ^2 results. It is obtained if p.gsqplot is set to 1 or 2 and requires for the input that *p.scale*$\{n\} > 0$. It is sometimes regarded as a preferred method for comparisons.

16.2.11. Parametric plots

Any input dataset can be converted to a parametric plot, where a second data input is plotted along the horizontal axis instead of the time coordinate. It is also possible to substitute a second data input for the x-axis data if a parametric plot in space is required instead. This allows visualization of how one type of data changes as a function of a second type of data input.

The two datasets that are plotted must have the same number of lines, that is, the first index range should be the same, in order that multiple lines can be compared. This is achieved where required using the p.scatters input in the simulation code. The details of the parametric plot are specified using the input:

$$p.parametric\{n\} = [n1, p2] \tag{16.4}$$

Here n is the graph number which is plotted, and must correspond to an input dataset. The number $n1$ is the graph number of the observable that is plotted on the horizontal axis, ignoring functional transformations. The second number is the axis number where the parametric value is substituted, which can be the time (axis 1) or the x-coordinate (axis 2), if present.

In all cases the vertical axis is used to plot the original data. The specified horizontal axis is used for the parametric variable. Only vertical error-bars are available. An example is given in xAMPLES/SDE_1/SHO, which is a noise-driven harmonic oscillator, with several lines plotted of x vs y.

16.3. xGRAPH Parameter reference

16.3.1. diffplot{n}

Default: 0

Differences are plotted as a comparison dashed line on $2D$ plots as a default. Otherwise, a separate difference plot is obtained which is unnormalized (diffplot = 1), or normalized (diffplot = 2) by the total RMS errors. If diffplot = 3, the comparison data is plotted directly as an additional graph.

Example: p.diffplot{3} = 2

16.3.2. errors

Default: 0

Indicates if the last index in the graphics input data arrays is used for error-bars and/or comparisons. Should be set to zero if there is no error or comparison data. If non-zero, this will give the highest last index used for errors. The standard xsim output sets $p.errors = 3$ automatically. As a special case, $p.errors = 1$ is used to indicate that there is comparison data but no error data.

If $p.errors > 0$, the data indexed up to p.errors gives the data, then a maximum of two types of error bars. Up to three further index values, up to $p.errors+3$, are available to index all comparison data and its error fields. The maximum last index value used is 6.

Example: p.errors = 2

16.3.3. esample{n}

Default: 1

This sets the type and size of sampling errors that are plotted. If esample = 0, no sampling error lines are plotted, just the mean. If $esample = -n$, $\pm n\sigma$ sampling errors are included in the error-bars. If $esample = n$, separate upper and lower $\pm n\sigma$ sampling error lines are plotted. In both cases, the magnitude of esample sets the number of standard deviations used.

Example: p.esample{3} = -1

16.3.4. font{n}

Default: 18

This sets the default font sizes for the graph labels, indexed by graph. This can be changed per graph.

Example: p.font{4}=18

16.3.5. functions

Default: number of functional transformations

This gives the maximum number of output graph functions and is available to restrict graphical output. The default is the length of the cell array of input data. Normally, the default will be used.

Example: p.functions = 10

16.3.6. gfunction{n} (d,p)

This is a cell array of graphics function handles. Use when a graph is needed that is a functional transformation of the observed averages. The default value generates the n-th graph data array directly from the n-th input data. The input is the data cell array for all the graphs in the current sequence number with their graph parameters x, and the output is the n-th data array that is plotted.

An arbitrary number of functions of these observables can be plotted, including vector observables. The input to graphics functions is the observed data averages or functions of averages in a given sequence, each stored in a cell array $d\{n\}(\ell, \mathbf{j}, c)$. If there are more graphics functions than input data cells, this generate additional data for plotting.

16.3.7. glabels{n}

Default: xlabels or klabels

Graph-dependent labels for the independent variable labels. This is a nested cell array with first dimension of graphs and second dimension of dimensions. This is used to replace the global values of xlabels or klabels if the axis labels change from graph to graph, for example, if the coordinates have a functional transform. These can be set for an individual coordinate on one graph if needed.

Example: p.glabels{4}{2} = 'xˆ2'

16.3.8. graphs

Default: observables to plot

This gives the observables to plot. The default is a vector of indices from one to the length of the cell array of observe functions. Normally not initialized, as the default is used. Mostly used to reduce graphical output on a long file.

Example: p.graphs = 10

16.3.9. gtransforms{n}

Default: [0,0,...]

This switch specifies the Fourier transformed graphs and axes for graphics labeling. Automatically equal to ftransforms if from an earlier xSIM input, but can be changed. If altered for a given graph, all the axis Fourier switches should be reset. This is ignored if there is no dimensions setting to indicate space dimensions.

Example: p.gtransforms{1} = [0,0,1]

16.3.10. headers{n}

Default: ''

This is a string variable giving the graph headers for each type of function plotted. The default value is an empty string. Otherwise, the header string that is input is used. Either is combined with the simulation name and a graph number to identify the graph. This is used to include simulation headers to identify graphs in simulation outputs. Graph headers may not be needed in a final published result. For this, either edit the graph, or use a space to make plot headers blank: p.headers{n} = ' ', or p.name = ' ' .

Example: p.headers{n} = 'my_graph_header'

16.3.11. images{n}

Default: 0

This is the number of 3D, transverse o-x-y images plotted as discrete time slices. Only valid if the input data dimension is greater than 2. If present, the coordinates not plotted are set to their central value when plotting the transverse images. This input should have a value from zero up to a maximum value of the number of plotted points. It has a vector length equal to graphs.

Example: p.images{4} = 5

16.3.12. imagetype{n}

Default: 1

This is the type of transverse o-x-y movie images plotted. It has a vector length equal to graphs.

- imagetype = 1 gives a perspective surface plot
- imagetype = 2, gives a 2D plot with colors
- imagetype = 3 gives a contour plot with 10 equally spaced contours
- imagetype = 4 gives a pseudo-color map

Example: p.imagetype{n} = 1, 2, 3, 4

16.3.13. klabels

Default: {'\omega', 'k_x', 'k_y', 'k_z'}“ or “{'k_1', 'k_2', 'k_3', 'k_4',...}

Labels for the graph axis Fourier transform labels, vector length of dimensions. The numerical labeling default is used when the “p.numberaxis“ option is set. Note, these are typeset in Latex mathematics mode! When changing from the default values, all the required new labels must be set.

Example: p.klabels= {'\Omega', 'K_x', 'K_y',}

16.3.14. legends{n}

Default: {”,”}

Graph-dependent legends, specified as a nested cell array of strings for each line.

Example: p.legends{n} = {labels(1), ..., labels(lines)}

16.3.15. limits{n}

Default: {0,0,0,0; ...}

Graph-dependent limits specified as a cell array with dimension graphs. Each entry is a cell array of graph limits indexed by the dimension, starting from $d = 1$ for the time dimension. The limits are vectors, indexed as 1,2 for the lower and upper plot limits. This is useful if the limits required change from graph to graph. If an automatic limit is required for either the upper or lower limit, it is set to inf.

An invalid, scalar or empty limit vector, like [0,0] or 0 or [] is ignored, and an automatic graph limit is used.

Example: p.limits{n} = {[t1,t2],[x1,x2],[y1,y2] ...,}

16.3.16. linestyle{n}

Default: {'-k','--k',':k','-.k','-ok','--ok',':ok','-.ok','-+k','--+k'}

Line types for each line in every two-dimensional graph plotted. If a given line on a two-dimensional line is to be removed completely, set the relevant line-style to zero. For example, to remove the first line from graph 3, set p.linestyle{3} ={0}. This is useful when generating and changing graphics output from a saved data file. The linestyle uses Matlab terminology. It allows setting the line pattern, marker symbols and color for every line. The default lines are black ('k'), but any other color can be used instead.

The specifiers must be chosen from the list below, eg, '-ok', although the marker can be omitted if not required.

- Line patterns: '-' (solid), '–' (dashed), ':' (dotted) ,'-.' (dash-dot)
- Marker symbols: '+','o','*','.','x','s','d','^','v','>','<','p'
- Colors: 'r','g','b','c','m','y','k','w'

Example: p.linestyle{4} = {'-k','--ok',':g','-.b',}

16.3.17. linewidth{n}

Default: 0.5

Line width for plotted lines in two-dimensional graphs. For example, to make the lines wider in graph 3, set p.linewidth{3} =1. This is useful for changing graphics output appearance if the default lines are too thin.

Example: p.linewidth{n} = 1

16.3.18. minbar{n}

Default: {0.01, ...}

This is the minimum relative error-bar that is plotted. Set to a large value to suppress unwanted error-bars, although its best not to ignore the error-bar information! This can be changed per graph.

Example: p.minbar{n} = 0

16.3.19. name

Default: ''

Name used to label simulation graphs, usually corresponding to the equation or problem solved. This can be removed from individual graphs by using headers{n} equal to a single blank space. The default is a null string. To remove all headers globally, set name equal to a single blank space: name = ' '.

Example: p.name = 'Wiener process simulation'

16.3.20. numberaxis

Default: 0

Switch for numbering graphics axes numerically. Otherwise the labels $t, x, , y, z$ are used, unless the total dimensions are too large.

Example: p.numberaxis = 1

16.3.21. octave

Default: 0

Switch for Octave program syntax, which can differ from Matlab standards. Octave is an open source, free alternative to Matlab.

Example: p.octave = 1

16.3.22. olabels{n}

Default: 'a'

Cell array of labels for the graph axis observables and functions. These are text labels that are used on the graph axes. The default value is 'a_1' if the default observable is used, otherwise it is blank. This is overwritten by any subsequent label input when the graphics program is run:

Example: p.olabels{4} = 'v'

16.3.23. parametric{n}

Default: [0,0]

Cell array that defines parametric plots, for each graph number. The first number is the graph number of the alternative observable plotted on the horizontal axis. The second number is the axis number where the parametric value is substituted, which can be the time (axis 1) or the x-coordinate (axis 2), if present.

If both are zero, the plot against an independent space-time coordinate is calculated as usual. If nonzero, a parametric plot is made for two-dimensional plots. In all cases the vertical axis is used to plot the original data. The specified horizontal axis is used for the parametric variable. Only vertical error-bars are available. Can be usefully combined with scatters{n} to plot individual trajectories, but the number of scatters should be the same in each of the two graphs that are parametrically plotted against each other.

Example: p.parametric{n} = [p1,p2] >= 0

16.3.24. pdimension{n}

Default: 3

This is the maximum plotted space-time dimension for each plotted quantity. The purpose is eliminate unwanted graphs. For example, it is useful to reduce the maximum dimension when averaging in space. Higher dimensional graphs are not needed, as the data is duplicated. Averaging can be useful for checking conservation laws, or for averaging over homogeneous data to reduce sampling errors. All graphs are suppressed if it is set to zero. Any three dimensions can be chosen to be plotted, using the axes parameter to suppress the unwanted data points in other dimensions.

Example: p.pdimension{4} = 2

16.3.25. savefig

Default: 0

If set to a string ending in .fig , all plots are saved to the current folder as .fig files, numbered consecutively as Fig1.fig, Fig2.fig.. . It is best to use *close all* to remove existing graphs, before running xgraph. The total number of files saved includes subfigures, and may be greater than the number of '*observe*' functions. A descriptive string before the ending can be used to identify the dataset.

Example: p.savefig = 'gx.fig' saves Matlab files with names Fig1gx.fig, Fig2gx.fig...).

16.3.26. savegraph

Default: 0

If set to a string ending in '.jpg', '.pdf', '.pdf', '.png', or '.tif' all plots are saved to the current folder as .jpg (etc) files, numbered as Fig1.jpg, and so on. One can use the *close all* command first to remove unwanted displayed graphs, before running xgraph with this option, since all displayed graphs are stored. A descriptive string before the ending can be used to identify the dataset.

Example: p.savegraph = 'gx.png' saves png files with names Fig1gx.png, Fig2gx.png...).

16.3.27. transverse{n}

Default: 0

This is the number of 2D transverse images plotted as discrete time slices. Only valid if dimensions is greater than 2. If present, the y, z-coordinates are set to their central values when plotting transverse images. Each element can be from 0 up to the number of plotted time-points. The cell array has a vector length equal to graphs.

Example: p.transverse{n}= 6

16.3.28. verbose

Default: 0

Print flag for output information while running xGRAPH. Print options are:

- Minimal if verbose = -1: Prints just the start-up time and hard error messages
- Brief if verbose = 0: Additionally prints the final, total chi-squared errors where present
- Informative if verbose = 1: Also prints the graph progress indicators
- Full if verbose = 2: Prints everything including the internal parameter structure data.

In summary, if verbose = 0, most output is suppressed except the final data, verbose = 1 displays a progress report, and verbose = 2 additionally generates a readable summary of the graphics parameter input.

Example: p.verbose = 0

16.3.29. xlabels

Default: {'t', 'x', 'y', 'z'} or {'x_1', 'x_2', 'x_3', 'x_4',...}

Global labels for the independent variable labels, vector length equal to dimensions. The numerical labeling default is used when the numberaxis option is true. These are typeset in Latex mathematics mode. When changing from the default values, all the required new labels must be set.

Example: p.xlabels = {'tau'}

16.3.30. xfunctions{n} {nd} (ax,p)

This is a nested cell array of axis transformations. Use when a graph is needed with an axis that is a function of the original axes. The input is the original axis coordinates, and the output is the new coordinate set. The default value generates the input axes. Called as xfunctions{n}{nd}(ax,p) for the n-th graph and axis direction dir, where ax is a vector of coordinates for that axis.There is one graphics function for each separate graph dimension or axis. The default value is the coordinate vector $xk\{nd\}$ stored in the input parameter structure p, or else the relevant index if xk{nd} is omitted.

16.4. xGRAPH structure

The graphics function, *xgraph*, plots the simulation data. The general structure is:

$$\begin{aligned}
\textbf{xgraph} &\rightarrow \textbf{xgpreferences}\ (\textit{checks inputs})\\
&\rightarrow \textbf{xmultigraph} \leftrightarrow \textbf{xreduce} \leftrightarrow \textbf{xcompress}\ (\textit{structures data arrays})\\
&\rightarrow \textbf{ximages} \rightarrow \textbf{xtransverse} \rightarrow \textbf{xplot3} \rightarrow \textbf{xplot2}\ (\textit{graphs all data})
\end{aligned}$$

Most graphics functions simply work, but two important functions are listed below for reference.

16.4.1. xgraph(data,input)

The xgraph function graphs multidimensional data files.

- Input: graphics data cells data, input parameter cells input.
- Output: graphs, displayed and/or stored as eps or fig files.
- If no numeric data present, reads data from a file named data.
- If data is present but without any input parameters it plots using default parameters.
- First data dimension is the line index, last dimension are the error-bars and comparisons
- Needs: xread, xmakecell, xgpreferences, xmultiplot

16.4.2. xgpreferences (input,oldinput)

The xgpreferences function sets default values for graphics inputs.

- Input: input cell array and optionally previous inputs from a datafile, oldinput.
- Note that each cell array is a sequence of graphics parameter structures
- Output: the updated plus default graphics parameters
- Called by: xgraph
- Needs: xprefer, xcprefer

Acknowledgements

We would like to thank the users whose feedback was invaluable, including Rodney Polkinghorne, Simon Kiesewetter, Bogdan Opanchuk, King Ng, Jesse van Rhijn and Thomas Rodriguez. This work was funded through the Australian Research Council Discovery Project scheme under Grants DP180102470 and DP190101480. The authors also wish to thank NTT Research and the Templeton Foundation for their financial and technical support.

Bibliography

[1] C. W. Gardiner, *Stochastic Methods: A Handbook for the Natural and Social Sciences*, Springer-Verlag, Berlin, Heidelberg, ISBN 978-3-540-70712-7 (2009). I, 2.1, 2.2, 2.6, 8.3, 11.3

[2] P. D. Drummond and M. Hillery, *The Quantum Theory of Nonlinear Optics*, Cambridge University Press, ISBN 9780511783616, doi:10.1017/CBO9780511783616 (2014). I, 2.1, 8.2, 8.3, 9.2, 11.1

[3] P. Langevin, *Sur la théorie du mouvement brownien*, C. R. Acad. Sci. (Paris) **146**, 530 (1908). I, 2.1

[4] I. Karatzas and S. Shreve, *Brownian motion and stochastic calculus*, Springer, 2nd edn., ISBN 978-0-387-97655-6, doi:10.1007/978-1-4612-0949-2 (1991). I, 2.1

[5] P. Glasserman, *Monte Carlo Methods in Financial Engineering*, Springer, reprint edn., ISBN 978-0-387-00451-8, doi:10.1007/978-0-387-21617-1 (2010). I

[6] B. Opanchuk, R. Polkinghorne, O. Fialko, J. Brand and P. D. Drummond, *Quantum simulations of the early universe*, Annalen der Physik **525**(10-11), 866 (2013), doi:10.1002/andp.201300113. I

[7] K. L. Ng, R. Polkinghorne, B. Opanchuk and P. D. Drummond, *Phase-space representations of thermal bose–einstein condensates*, Journal of Physics A Mathematical General **52**(3) (2019), doi:10.1088/1751-8121/aaeeb1. I

[8] V. Ramesh, K. Peters and S. Rodriguez, *Arcsine laws of light*, arXiv preprint arXiv:2208.07432 (2022). I

[9] K. Peters and S. Rodriguez, *Limit cycles and chaos induced by a nonlinearity with memory*, The European Physical Journal Special Topics **231**(3), 247 (2022), doi:10.1140/epjs/s11734-021-00407-3. I

[10] R. Y. Teh, S. Kiesewetter, M. D. Reid and P. D. Drummond, *Simulation of an optomechanical quantum memory in the nonlinear regime*, Phys. Rev. A **96**, 013854 (2017), doi:10.1103/PhysRevA.96.013854. I

[11] R. Y. Teh, *Quantum correlations in mesoscopic systems*, Ph.D. thesis, Swinburne University of Technology (2018). I

[12] S. Kiesewetter and P. D. Drummond, *Phase-space simulations of feedback coherent ising machines*, Optics Letters **47**(3), 649 (2022), doi:10.1364/ol.434114. I

[13] R. Teh, S. Kiesewetter, P. D. Drummond and M. Reid, *Creation, storage, and retrieval of an optomechanical cat state*, Physical Review A **98**(6), 063814 (2018), doi:10.1103/physreva.98.063814. I

[14] K. Dechoum, L. Rosales-Zárate and P. D. Drummond, *Critical fluctuations in an optical parametric oscillator: when light behaves like magnetism*, JOSA B **33**(5), 871 (2016), doi:10.1364/josab.33.000871. I

[15] P. D. Drummond and B. Opanchuk, *Truncated wigner dynamics and conservation laws*, Physical Review A **96**(4), 043616 (2017), doi:10.1103/physreva.96.043616. I

[16] B. Opanchuk and P. D. Drummond, *One-dimensional bose gas dynamics: Breather relaxation*, Phys. Rev. A **96**, 053628 (2017), doi:10.1103/PhysRevA.96.053628. I

[17] P. Drummond, *Higher-order stochastic differential equations and the positive wigner function*, Physical Review A **96**(6), 062104 (2017), doi:10.1103/physreva.96.062104. I

[18] K. L. Ng, B. Opanchuk, M. Thenabadu, M. Reid and P. D. Drummond, *Fate of the false vacuum: Finite temperature, entropy, and topological phase in quantum simulations of the early universe*, PRX Quantum **2**(1), 010350 (2021), doi:10.1103/prxquantum.2.010350. I

[19] R. R. Joseph, J. van Rhijn and P. D. Drummond, *A hybrid projection algorithm for stochastic differential equations on manifolds*, arXiv preprint arXiv:2112.03391 (2021). I

[20] P. D. Drummond, *Time evolution with symmetric stochastic action*, Physical Review Research **3**(1), 013240 (2021), doi:10.1103/physrevresearch.3.013240. I

[21] P. D. Drummond and B. Opanchuk, *Initial states for quantum field simulations in phase space*, Physical Review Research **2**(3), 033304 (2020), doi:10.1103/physrevresearch.2.033304. I

[22] K. L. Ng, B. Opanchuk, M. D. Reid and P. D. Drummond, *Nonlocal pair correlations in a higher-order bose gas soliton*, Physical review letters **122**(20), 203604 (2019), doi:10.1103/physrevlett.122.203604. I

[23] J. Busink, P. Ackermans, K. Cognee and S. Rodriguez, *Stochastic light in a cavity: A brownian particle in a scalar potential?*, arXiv preprint arXiv:2107.01414 (2021). I

[24] K. Peters, Z. Geng, K. Malmir, J. Smith and S. Rodriguez, *Extremely broadband stochastic resonance of light and enhanced energy harvesting enabled by memory effects in the nonlinear response*, Physical Review Letters **126**(21), 213901 (2021), doi:10.1103/physrevlett.126.213901. I

[25] K. J. Peters and S. R. Rodriguez, *Exceptional precision of a nonlinear optical sensor at a square-root singularity*, Physical Review Letters **129**(1), 013901 (2022), doi:10.1364/np.2022.nptu4f.3. I

[26] S. Kiesewetter, R. Polkinghorne, B. Opanchuk and P. D. Drummond, *xspde: Extensible software for stochastic equations*, SoftwareX **5**, 12 (2016), doi: 10.1016/j.softx.2016.02.001. I

[27] S. Kiesewetter, R. R. Joseph and P. D. Drummond, *xspde3: Extensible software for stochastic ordinary and partial differential equations*, SciPost Physics Codebases p. 017 (2023). I

[28] P. D. Drummond and I. K. Mortimer, *Computer simulations of multiplicative stochastic differential equations*, J. Comput. Phys. **93**, 144 (1991), doi:10.1016/0021-9991(90)90131-j. I, 2.2, 11.3

[29] P. E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations*, Springer-Verlag, Berlin, doi:10.5194/gmd-2016-45-rc1 (1992). I, 13.3.7, 14.1.1

[30] M. J. Werner and P. D. Drummond, *Robust algorithms for solving stochastic partial differential equations*, J. Comput. Phys. **132**, 312 (1997), doi:10.1006/jcph.1996.5638. I, 5.5

[31] K. Burrage, P. Burrage, D. J. Higham, P. E. Kloeden and E. Platen, *Comment on "numerical methods for stochastic differential equations"*, Physical Review E **74**(6), 068701 (2006), doi:10.1103/physreve.74.068701. I, 14.1.1

[32] J. W. Eaton, *Gnu octave and reproducible research*, Journal of Process Control **22**(8), 1433 (2012), doi:10.1016/j.jprocont.2012.04.006. I

[33] G. Collecutt and P. D. Drummond, *xmds: extensible multi-dimensional simulator*, Computer physics Communications **142**(1-3), 219 (2001), doi:10.1016/s0010-4655(01)00309-5. I

[34] G. R. Dennis, J. J. Hope and M. T. Johnsson, *Xmds2: Fast, scalable simulation of coupled stochastic partial differential equations*, Computer Physics Communications **184**(1), 201 (2013), doi:10.1016/j.cpc.2012.08.016. I

[35] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex fourier series*, Mathematics of computation **19**(90), 297 (1965), doi: 10.2307/2003354. 1.4.6

[36] R. R. Joseph, J. van Rhijn and P. D. Drummond, *Midpoint projection algorithm for stochastic differential equations on manifolds*, Physical Review E **107**(5) (2023), doi:10.1103/physreve.107.055307. 1.9, 11.3, 11.3

[37] S. Kiesewetter, *Coherent ising machine with quantum feedback: The total and conditional master equation methods*, Physical Review A **106**(2) (2022), doi: 10.1103/PhysRevA.106.022409. 1.10

[38] J. Dalibard, *Wave-function approach to dissipative processes in quantum optics*, Physical Review Letters **68**(5), 580 (1992), doi:10.1103/PhysRevLett.68.580. 1.10, 11.4.3

[39] H. J. Carmichael, *Quantum trajectory theory for cascaded open systems*, Physical Review Letters **70**(15), 2273 (1993), doi:10.1103/PhysRevLett.70.2273. 1.10, 11.4.3

[40] M. R. Hush, S. S. Szigeti, A. R. R. Carvalho and J. J. Hope, *Controlling spontaneous-emission noise in measurement-based feedback cooling of a bose–einstein condensate*, New Journal of Physics **15**(11), 113060 (2013), doi: 10.1088/1367-2630/15/11/113060. 1.10

[41] N. Wiener, *Generalized harmonic analysis*, Acta mathematica **55**(1), 117 (1930), doi:10.1007/bf02546511. 2.1

[42] K. Itô and H. P. McKean, *Diffusion processes and their sample paths: Reprint of the 1974 edition*, Springer Science & Business Media, doi:10.1007/978-3-642-62025-6 (1996). 2.1

[43] R. L. Stratonovich, *On the theory of non-equilibrium random processes*, Soviet Physics JETP **11**(3) (1960). 2.1, 11.3

[44] N. G. Van Kampen, *Stochastic Processes in Physics and Chemistry*, North Holland, 3rd edn., ISBN 978-0444529657 (2007). 2.1

[45] F. C. Klebaner, *Introduction to stochastic calculus with applications*, World Scientific Publishing Company, doi:978-1848168329 (2012). 2.1

[46] L. Arnold, *Stochastic differential equations: theory and applications*, Folens Publishers, reprint edn., doi:10.1002/zamm.19770570413 (1992). 2.1, 2.2

[47] R. Graham, *Path integral formulation of general diffusion processes*, Zeitschrift für Physik B Condensed Matter **26**(3), 281 (1977), doi:10.1007/BF01312935. 2.1.1

[48] P. D. Drummond, *Forward, backward, and weighted stochastic bridges*, Phys. Rev. E **96**, 042123 (2017), doi:10.1103/PhysRevE.96.042123. 2.1.1

[49] D. J. Higham and P. E. Kloeden, *Numerical methods for nonlinear stochastic differential equations with jumps*, Numerische Mathematik **101**(1), 101 (2005). 2.5

[50] H. Risken, *The Fokker-Planck Equation*, Springer-Verlag, Berlin, 2nd edn., doi: 10.1007/978-3-642-61544-3 (1996). 2.6

[51] J. Ma and J. Yong, *Forward-backward stochastic differential equations and their applications*, 1702. Springer Science & Business Media (1999). 2.9, 2.9, 3.8

[52] P. D. Drummond and M. D. Reid, *Retrocausal model of reality for quantum fields*, Phys. Rev. Research **2**, 033266 (2020), doi:10.1103/PhysRevResearch.2.033266. 2.9

[53] F. Black, *The pricing of commodity contracts*, Journal of Financial Economics **3**(1), 167 (1976), doi:https://doi.org/10.1016/0304-405X(76)90024-6. 3.2

[54] H. Quastel, Jand Spohn, *The one-dimensional kpz equation and its universality class*, Journal of Statistical Physics **160**(4), 965 (2015), doi:https://doi.org/10.1007/s10955-015-1250-9. 5.1

[55] C.-H. Lam and F. G. Shin, *Improved discretization of the kardar-parisi-zhang equation*, Physical Review E **58**(5), 5592 (1998), doi:10.1103/PhysRevE.58.5592. 5.1

[56] P. D. Drummond and A. D. Hardman, *Simulation of Quantum Effects in Raman-Active Waveguides*, Europhys. Lett. **21**, 279 (1993), doi:10.1209/0295-5075/21/3/005. 5.5

[57] W. Bao and J. Shen, *A fourth-order time-splitting laguerre–hermite pseudospectral method for bose–einstein condensates*, SIAM Journal on Scientific Computing **26**(6), 2010 (2005), doi:10.1137/030601211. 5.5.1

[58] P. B. Blakie and M. J. Davis, *Projected gross-pitaevskii equation for harmonically confined bose gases at finite temperature*, Phys. Rev. A **72**, 063608 (2005), doi:10.1103/PhysRevA.72.063608. 5.5.1

[59] A. Jentzen and P. E. Kloeden, *The numerical approximation of stochastic partial differential equations*, Milan Journal of Mathematics **77**, 205 (2009), doi:https://doi.org/10.1007/s00032-009-0100-0. 5.5.1

[60] M. Frigo and S. G. Johnson, *FFTW: An adaptive software architecture for the FFT*, In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing*, vol. 3, pp. 1381–1384. IEEE, doi:10.1109/ICASSP.1998.681704 (1998). 5.7.1

[61] M. Frigo and S. G. Johnson, *The design and implementation of FFTW3*, Proceedings of the IEEE **93**(2), 216 (2005), doi:10.1109/JPROC.2004.840301, Special issue on "Program Generation, Optimization, and Platform Adaptation". 5.7.1

[62] J. Crank and P. Nicolson, *A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type*, In *Mathematical proceedings of the Cambridge philosophical society*, vol. 43, pp. 50–67. Cambridge University Press, doi:10.1017/S0305004100023197 (1947). 5.10.2

[63] A. Scott, F. Chu and D. McLaughlin, *The soliton: A new concept in applied science*, Proceedings of the IEEE **61**(10), 1443 (1973), doi:10.1109/proc.1973.9296. 6.3

[64] C. W. Gardiner and M. J. Davis, *The stochastic gross–pitaevskii equation: II*, Journal of Physics B: Atomic, Molecular and Optical Physics **36**(23), 4731 (2003), doi:10.1088/0953-4075/36/23/010. 6.5

[65] R. Courant and D. Hilbert, *Methods of mathematical physics: partial differential equations*, vol. 2, John Wiley & Sons, ISBN 978-0471504399 (2008). 6.6

[66] P. W. Anderson, *Absence of diffusion in certain random lattices*, Physical Review **109**(5), 1492 (1958), doi:10.1103/PhysRev.109.1492. 6.7

[67] J. Billy, V. Josse, Z. Zuo, A. Bernard, B. Hambrecht, P. Lugan, D. Clément, L. Sanchez-Palencia, P. Bouyer and A. Aspect, *Direct observation of anderson localization of matter waves in a controlled disorder*, Nature **453**(7197), 891 (2008), doi:10.1038/nature07000. 6.7

[68] A. S. Pikovsky, *Destruction of anderson localization by a weak nonlinearity*, Physical Review Letters **100**(9) (2008), doi:10.1103/PhysRevLett.100.094101. 6.7

[69] E. Wigner, *On the Quantum Correction For Thermodynamic Equilibrium*, Phys. Rev. **40**, 749 (1932), doi:10.1103/PhysRev.40.749. 7.1, 8.1, 8.2

[70] R. J. Glauber, *Coherent and incoherent states of the radiation field*, Phys. Rev. **131**, 2766 (1963), doi:10.1103/PhysRev.131.2766. 7.1, 8.1.1, 8.2, 8.9

[71] E. C. G. Sudarshan, *Equivalence of Semiclassical and Quantum Mechanical Descriptions of Statistical Light Beams*, Phys. Rev. Lett. **10**, 277 (1963). 7.1

[72] P. D. Drummond and C. W. Gardiner, *Generalised P-representations in quantum optics*, J. Phys. A **13**, 2353 (1980), doi:10.1088/0305-4470/13/7/018. 7.1, 8.1.2

[73] P. P. Deuar and P. D. Drummond, *Gauge P representations for quantum-dynamical problems: Removal of boundary terms*, Phys. Rev. A **66**(3), 033812 (2002), doi:10.1103/PhysRevA.66.033812. 7.1, 8.2.1, 8.4

[74] W. H. Louisell, *Quantum statistical properties of radiation*, Wiley, New York, ISBN 978-0471523659 (1973). 7.2, 8.2

[75] H. J. Carmichael, *Statistical methods in quantum optics 1. Master Equations and Fokker-Planck Equations.*, Springer, Berlin, ISBN 978-3-540-54882-9, doi:https://doi.org/10.1007/978-3-662-03875-8 (2002). 7.2, 8.2, 11.1

[76] C. W. Gardiner and P. Zoller, *Quantum noise: a handbook of Markovian and non-Markovian quantum stochastic methods with applications to quantum optics*, Springer Science & Business Media, ISBN 978-3-540-22301-6 (2004). 7.2, 8.2, 8.5, 9.2

[77] P. A. M. Dirac, *On the analogy between classical and quantum mechanics*, Rev. Mod. Phys. **17**, 195 (1945), doi:10.1103/RevModPhys.17.195. 8.1

[78] B. Opanchuk, L. Rosales-Zárate, R. Y. Teh, B. J. Dalton, A. Sidorov, P. D. Drummond and M. D. Reid, *Mesoscopic two-mode entangled and steerable states of 40 000 atoms in a Bose-Einstein-condensate interferometer*, Phys. Rev. A **100**, 060102 (2019), doi:10.1103/PhysRevA.100.060102. 8.2

[79] B. Opanchuk, L. Rosales-Zárate, M. D. Reid and P. D. Drummond, *Robustness of quantum fourier transform interferometry*, Optics letters **44**(2), 343 (2019), doi:10.1364/OL.44.000343. 8.2

[80] E. Schrödinger, *The constant crossover of micro-to macro mechanics*, Naturwissenschaften **14**, 664 (1926). 8.2

[81] J. E. Moyal, *Quantum mechanics as a statistical theory*, Mathematical Proceedings of the Cambridge Philosophical Society **45**(01), 99 (1949), doi:10.1017/S0305004100000487. 8.2

[82] S. J. Carter, P. D. Drummond, M. D. Reid and R. M. Shelby, *Squeezing of quantum solitons*, Phys. Rev. Lett. **58**, 1841 (1987), doi:10.1103/PhysRevLett.58.1841. 8.2.1

[83] A. Gilchrist, C. W. Gardiner and P. D. Drummond, *Positive P-representation: Application and validity*, Phys. Rev. A **55**, 3014 (1997). 8.2.1

[84] M. J. Steel, M. K. Olsen, L. I. Plimak, P. D. Drummond, S. M. Tan, M. J. Collett, D. F. Walls and R. Graham, *Dynamical quantum noise in trapped bose-einstein condensates*, Phys. Rev. A **58**, 4824 (1998), doi:10.1103/PhysRevA.58.4824. 8.2.1

[85] U. M. Titulaer and R. J. Glauber, *Correlation functions for coherent fields*, Phys. Rev. **140**, B676 (1965), doi:10.1103/PhysRev.140.B676. 8.4

[86] M. D. Reid and D. F. Walls, *Violations of classical inequalities in quantum optics*, Phys. Rev. A **34**(2), 1260 (1986). 8.4

[87] S. Kiesewetter, Q. Y. He, P. D. Drummond and M. D. Reid, *Scalable quantum simulation of pulsed entanglement and einstein-podolsky-rosen steering in optomechanics*, Phys. Rev. A **90**, 043805 (2014), doi:10.1103/PhysRevA.90.043805. 8.5

[88] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu *et al.*, *Quantum computational advantage using photons*, Science **370**(6523), 1460 (2020). 8.6

[89] Y.-H. Deng, Y.-C. Gu, H.-L. Liu, S.-Q. Gong, H. Su, Z.-J. Zhang, H.-Y. Tang, M.-H. Jia, J.-M. Xu, M.-C. Chen, J. Qin, L.-C. Peng *et al.*, *Gaussian boson sampling with pseudo-photon-number-resolving detectors and quantum computational advantage*, Phys. Rev. Lett. **131**, 150601 (2023), doi:10.1103/PhysRevLett.131.150601. 8.6

[90] A. Deshpande, A. Mehta, T. Vincent, N. Quesada, M. Hinsche, M. Ioannou, L. Madsen, J. Lavoie, H. Qi, J. Eisert, D. Hangleiter, B. Fefferman *et al.*, *Quantum computational advantage via high-dimensional Gaussian boson sampling*, Sci. Adv. **8**(1), eabi7894 (2022), doi:10.1126/sciadv.abi7894. 8.6, 8.8.2

[91] L. S. Madsen, F. Laudenbach, M. F. Askarani, F. Rortais, T. Vincent, J. F. F. Bulmer, F. M. Miatto, L. Neuhaus, L. G. Helt, M. J. Collins, A. E. Lita, T. Gerrits *et al.*, *Quantum computational advantage with a programmable photonic processor*, Nature **606**(7912), 75 (2022), doi:10.1038/s41586-022-04725-x. 8.6

[92] R. Loudon, *The Quantum Theory of Light*, Oxford Science Publications. Clarendon Press, Oxford, 2nd ed. edn., ISBN 978-0-19-851155-7 (1983). 8.7.1

[93] D. Walls and G. Milburn, *Quantum Optics*, Springer (2008). 8.7.1, 8.8

[94] P. Marian, *Higher-order squeezing and photon statistics for squeezed thermal states*, Physical Review A **45**(3), 2044 (1992). 8.7.2, 8.8.1

[95] C. S. Hamilton, R. Kruse, L. Sansoni, S. Barkhofen, C. Silberhorn and I. Jex, *Gaussian boson sampling*, Phys. Rev. Lett. **119**, 170501 (2017), doi:10.1103/PhysRevLett.119.170501. 8.8

[96] N. Quesada, J. M. Arrazola and N. Killoran, *Gaussian boson sampling using threshold detectors*, Physical Review A **98**(6), 062322 (2018). 8.8

[97] C. Zhu and C. M. Caves, *Photocount distributions for continuous-wave squeezed light*, Phys. Rev. A **42**(11), 6794 (1990), doi:10.1103/PhysRevA.42.6794. 8.8.2, 8.8.2

[98] J. Huang and P. Kumar, *Photon-counting statistics of multimode squeezed light*, Phys. Rev. A **40**(3), 1670 (1989), doi:10.1103/PhysRevA.40.1670. 8.8.2

[99] P. D. Drummond, B. Opanchuk, A. Dellios and M. D. Reid, *Simulating complex networks in phase space: Gaussian boson sampling*, Phys. Rev. A **105**(1), 012427 (2022), doi:10.1103/PhysRevA.105.012427. 8.9, 8.10

[100] A. S. Dellios, M. D. Reid and P. D. Drummond, *Validation tests of gaussian boson samplers with photon-number resolving detectors*, arXiv preprint arXiv:2411.11228 (2024). 8.9, 8.10

[101] P. Drummond, K. McNeil and D. Walls, *Non-equilibrium transitions in sub/second harmonic generation*, J. Mod. Opt. **28**(2), 211 (1981), doi:10.1080/713820226. 9.2

[102] F.-X. Sun, *Schrödinger cat states and steady states in subharmonic generation with kerr nonlinearities*, Physical Review A **100**(3) (2019), doi:10.1103/PhysRevA.100.033827. 9.2

[103] F.-X. Sun, Q. He, Q. Gong, R. Y. Teh, M. D. Reid and P. D. Drummond, *Discrete time symmetry breaking in quantum circuits: exact solutions and tunneling*, New Journal of Physics **21**(9), 093035 (2019), doi:10.1088/1367-2630/ab3f5e. 9.2

[104] L. A. Lugiato, C. Oldano, C. Fabre, E. Giacobino and R. J. Horowicz, *Bistability, self-pulsing and chaos in optical parametric oscillators*, Il Nuovo Cimento D **10**(8), 959 (1988), doi:10.1007/bf02450197. 9.2

[105] Z. Leghtas, S. Touzard, I. M. Pop, A. Kou, B. Vlastakis, A. Petrenko, K. M. Sliwa, A. Narla, S. Shankar, M. J. Hatridge, M. Reagor, L. Frunzio *et al.*, *Confining the state of light to a quantum manifold by engineered two-photon loss*, Science **347**(6224), 853 (2015), doi:10.1126/science.aaa2085. 9.2

[106] A. Marandi, Z. Wang, K. Takata, R. L. Byer and Y. Yamamoto, *Network of time-multiplexed optical parametric oscillators as a coherent ising machine*, Nature Photonics **8**(12), 937 (2014), doi:10.1038/nphoton.2014.249. 9.2

[107] P. L. McMahon, A. Marandi, Y. Haribara, R. Hamerly, C. Langrock, S. Tamate, T. Inagaki, H. Takesue, S. Utsunomiya, K. Aihara, R. L. Byer, M. M. Fejer *et al.*, *A fully programmable 100-spin coherent ising machine with all-to-all connections*, Science **354**(6312), 614 (2016), doi:10.1126/science.aah5178. 9.2

[108] T. Inagaki, Y. Haribara, K. Igarashi, T. Sonobe, S. Tamate, T. Honjo, A. Marandi, P. L. McMahon, T. Umeki, K. Enbutsu, O. Tadanaga, H. Takenouchi *et al.*, *A coherent ising machine for 2000-node optimization problems*, Science **354**(6312), 603 (2016), doi:10.1126/science.aah4243. 9.2

[109] P. D. Drummond, *Quasiprobability methods for nonlinear chemical and optical systems*, Physical Review A **24**(2), 914 (1981), doi:10.1103/PhysRevA.24.914. 9.2

[110] P. Kinsler, *Critical fluctuations in the quantum parametric oscillator*, Physical Review A **52**(1), 783 (1995), doi:10.1103/PhysRevA.52.783. 9.2

[111] S. Kiesewetter, *Pulsed entanglement of two optomechanical oscillators and furry's hypothesis*, Physical Review Letters **119**(2) (2017), doi:10.1103/PhysRevLett.119.023601. 9.2

[112] M. D. Reid, *Demonstration of the einstein-podolsky-rosen paradox using nondegenerate parametric amplification*, Physical Review A **40**(2), 913 (1989), doi:10.1103/PhysRevA.40.913. 9.2

[113] M. D. Reid, *Correlations in nondegenerate parametric oscillation: Squeezing in the presence of phase diffusion*, Physical Review A **40**(8), 4493 (1989), doi:10.1103/PhysRevA.40.4493. 9.2

[114] L. Rosales-Zárate, *Probabilistic quantum phase-space simulation of bell violations and their dynamical evolution*, Physical Review A **90**(2) (2014), doi:10.1103/PhysRevA.90.022109. 9.2

[115] M. D. Reid, *Quantum probabilistic sampling of multipartite 60-qubit bell-inequality violations*, Physical Review A **90**(1) (2014), doi:10.1103/PhysRevA.90.012111. 9.2

[116] N. Gisin and I. C. Percival, *The quantum-state diffusion model applied to open systems*, Journal of Physics A: Mathematical and General **25**(21), 5677 (1992). 11.1, 11.2, 11.2.1

[117] P. Pearle, *Toward explaining why events occur*, International Journal of Theoretical Physics **18**, 489 (1979). 11.2

[118] G. C. Ghirardi, A. Rimini and T. Weber, *Unified dynamics for microscopic and macroscopic systems*, Physical Review D **34**, 470 (1986), doi:10.1103/PhysRevD.34.470. 11.2

[119] M. Rigo, F. Mota-Furtado and P. O'Mahony, *Continuous stochastic schrödinger equations and localization*, Journal of Physics A: Mathematical and General **30**(21), 7557 (1997). 11.2.1

[120] L. Diósi, *Models for universal reduction of macroscopic quantum fluctuations*, Physical Review A **40**(3), 1165 (1989). 11.3

[121] J. Gambetta and H. M. Wiseman, *Non-markovian stochastic schrödinger equations: Generalization to real-valued noise using quantum-measurement theory*, Phys. Rev. A **66**, 012108 (2002), doi:10.1103/PhysRevA.66.012108. 11.3.1

[122] B. Mollow, *Pure-state analysis of resonant light scattering: Radiative damping, saturation, and multiphoton effects*, Physical Review A **12**(5), 1919 (1975). 11.4.3

[123] P. Zoller, M. Marte and D. F. Walls, *Quantum jumps in atomic systems*, Phys. Rev. A **35**, 198 (1987), doi:10.1103/PhysRevA.35.198. 11.4.3

[124] K. Mølmer, Y. Castin and J. Dalibard, *Monte carlo wave-function method in quantum optics*, JOSA B **10**(3), 524 (1993). 11.4.3

[125] C. W. Gardiner, A. S. Parkins and P. Zoller, *Wave-function quantum stochastic differential equations and quantum-jump simulation methods*, Phys. Rev. A **46**, 4363 (1992), doi:10.1103/PhysRevA.46.4363. 11.4.3

[126] I. Carusotto, Y. Castin and J. Dalibard, *n-boson time-dependent problem: A reformulation with stochastic wave functions*, Phys. Rev. A **63**, 023606 (2001). 12.4