xQSim Manual: v1.0

December 28, 2021

1 Introduction

xQSim is a software package for e**x**tensible **Q**uantum **Sim**ulations using phase-space representations. It can treat a variety of input types, networks and output measurements. It uses three phase-space simulation methods: the generalized P-distribution ($method = 1$), the Wigner distribution ($method = 2$), and the Q-function ($method = 3$), provided they have positive distributions. Currently, saturating detectors require the use of $method = 1$.

The code provided in v1.0 simulates gaussian quantum inputs with an admixture of thermal noise to model phase decoherence. Details are in a theoretical paper [1].

The package provided has six parts:

1. **xQSimCode.** This has the main quantum simulation code.
2. **xQSimExamples.** This has the scripts used to simulate experiments for comparisons, as well as scripts used for testing.
3. **xQSimData.** This has data extracted from a gaussian boson sampling experiment, in a format where it can be loaded for comparisons.
4. **xQSimDocumentation.** This provides the documentation, including this report and a preprint of a research paper that uses the software.
5. **xGraph.** This multidimensional graphics program is documented in a separate user guide.
6. **xQSimGBSExperiments.** This has recent public experimental data, and data extraction codes for reference purposes.

The code provided is written in Matlab, which is a proprietary language of The Mathworks Inc. It is fully compatible with Octave, which is a slower, free public domain clone of Matlab. Parallel features are only available in Matlab, and require the parallel toolbox. These features are optional, but give large speed improvements on multicore computers if the parallel toolbox is used. Faster GPU/Python versions exist also, for GPU enabled supercomputers.

Data corresponds to an experimental paper [2].

2 xQSim operation

The initial xQSim software distributions consists of Matlab functions, including *xQSim* itself, together with functions called by *xQSim*. There are also test scripts with inputs that can be modified. The functions used can be replaced as required for different applications. The input is a sequence of parameter structures in a cell array. The output data is a nested cell array giving first the sequence index, then one graph index for each average correlation calculated.

The output data is compatible with the *xGraph* multidimensional batch graphics program, documented separately. Other graphics codes can be used, too, if compatible with the data files. An example and description of the basic functions available is given below.

2.1 xQSim example

xQSim is the quantum simulation code. To run this, an input m-file is needed to define the simulation parameters and run the code. A simple example is given below. This makes use of xGraph, an accompanying batch graphics program, but use of xGraph is optional. More examples are given in the xQSimExamples folder.

```
p.matrix     = @Unitary;                    %matrix type
p.M          = 40;                          %matrix size m
p.N          = 20;                          %input size n
p.ensembles  = [1000,10,12];                %repeats for errors
p.r          = 1;                           %squeezing parameter(s)
p.observe    = {@n,@k};                     %correlation functions
p.glabels    = {{'Mode m'},{'Mode m'}};     %axis label
p.olabels    = {'<n>','<c>'};               %Numbers and clicks
[e1,d,p]     = xqsim(p);                    %simulation function
xgraph(d,p);                                %graphics function
```

2.2 xQSim parameters

The input is a cell array of parameter structures p. The code generates simulation data and sampling errors in an output cell data array d, including comparative experimental or analytic data if required. An input cell array can be replaced by a single parameter structure if required.

The simulations can treat an arbitrary sequence of networks and parameters. Some inputs are optional, and have specified defaults. These allow the input parameter list to be shortened. The main inputs in the structure p are:

Label	Type	Default	Description
M	integer	1	Total matrix size
N	integer	1	Number of excited input modes
cyc	integer	1	Number of cycles of network
matrix	matrix function	Identity	An $M \times M$ transfer matrix function
ensembles	vector	[1,1,1]	Size of vector, serial, parallel ensemble
method	integer	1	Phase-space method: $m = 1, 2, 3$
name	character	”	Name of simulation
pnames	array of characters	{’+P ’,’W ’,’Q ’}	Names of phase-space methods
Tname	unitary name	matrix(0)	Name of transmission matrix
r	vector or function	0	An n-dimensional squeezing vector
eps	vector	0	An n-dimensional decoherence factor
t	vector	1	An n-dimensional transmission factor
re	vector	1	An n-dimensional recycling factor
observe	function handles	$\{@(a,p)\, a\}$	Observable functions
compare	function handles	[]	Optional comparison functions
cutoff	real	-1.e100	Lower cutoff for graphs and χ^2
counts	integer	0	Total experimental counts for χ^2 tests
mincount	integer	0	Minimum count for χ^2 tests
graphs	integer	1	Maximum number of data outputs
gen	function handle	$@xqgen$	Stochastic generation code
print	integer	1	Flag for printing verbosity level: 0,1,2

- Note that vectors r, *eps*, t are N-dimensional. They are expanded to N dimensions by repeating the last element, if they are less than N-dimensional.
- If $ensembles(1) > 1$, all calculations are repeated in parallel and averaged using low-level, single core vector methods.
- If $ensembles(2) > 1$, calculations are repeated sequentially, in a loop, in order to estimate sampling errors.
- If $ensembles(3) > 1$, calculations are repeated using multi-core parallel loops. This is also used to estimate sampling errors.
- If $method > 1$, an alternative phase-space method is used: Wigner or Q. The default is the +P method.
- The squeezing vector $\boldsymbol{r}$ and transfer matrix *matrix* can be input as numbers or as a function handle. The transmission factor t is an additional prefactor applied to inputs prior to the measured transmission *matrix* to allow additional fine-tuning.

Other notation used in the input structure is given in the associated preprint. Note that *graphs* is the number of distinct graph and data types. More than one output plot can be generated per graph.

2.3 Simulation notes

- Starting from a vacuum state, $\boldsymbol{a}^{(0)}$, xQSim iteratively combines the n-th output $\boldsymbol{a}^{(n)}$ with a locally generated Gaussian, or another state in general. The quantum state is transformed through a network to generate $\boldsymbol{a}^{(n+1)}$ and detected.
- Each network is repeated *cyc* times, with a default of $cyc = 1$. If $cyc > 1$, output data is indexed by cycle, and graphs vs cycle number are made. If $cyc > 1$ and $re > 0$, the field is recycled with recycling amplitude re.
- Non-gaussian inputs are generated by changing the *qgen* code, which generates the local input and simulates the network.
- For saturating or 'click' detectors, only +P-distributions can be used. Saturating detectors have a more complex operator form.
- The Wigner distribution is optimal for quadrature detection, while the P-distribution is best for number and click detectors.
- Non-optimal representations may cause a growth in sampling error for high-order correlations with growing network size.

2.4 Data structures in xqsim and xgraph

2.4.1 xqsim inputs

When *xqsim* is used, it is called with the argument (*input*). Each simulation is defined as a cell array of structures:

$$input := \{p_1, p_2, \ldots\}$$

Each structure p contains parameters relevant to a particular step in the sequence:

$$p := (p.M, p.N, \ldots)$$

If the sequence has just one member, a structure can used directly, without an enclosing cell array.

2.4.2 xqsim internal phase-space

Although this is transparent to the user, each network in the sequence transforms a matrix of quantum amplitudes:

$$\boldsymbol{a} := [\boldsymbol{\alpha}, \boldsymbol{\beta}]_j = a_{ij}$$

The first index is the mode index from $1, \ldots 2M$, where indices from $1, \ldots, M$ indicate amplitudes equivalent to $\hat{a}_i$, and indices from $M+1, \ldots 2M$ are amplitudes equivalent to $\hat{a}_i^\dagger$. These quantum amplitudes can be recycled in a sequence of network operations.

The second index is from $1, \ldots p.ensembles(1)$, for ensemble averaging using vector parallel processing. These transformations are repeated $p.ensembles(2)$ times in a local loop, and $p.ensembles(3)$ times nonlocally, with parallel loops. At least one loop ensemble larger than 1 is required to estimate errors.

2.4.3 xqsim outputs

The output data is a nested cell array, whose length is the total sequence length:

$$data := \{d_1, d_2, \ldots\}$$

At each step in the sequence, a data cell array d of individual quantum expectation values or graphs is generated, where:

$$d := \{g_1, g_2, \ldots\}$$

Here each graph g_n is the output of the $n{-}th$ defined observe function, described below. These are real, multidimensional arrays of the data. They include averages calculated by the observe functions, and error-bars generatd by the *qsim* code:

$$g := (g_{\ell j_1 j_2 \ldots j_n, e})$$

There is a standardized form of g. The first index ℓ is a line index, reserved for subsequent plotting software. The next n indices are a multidimensional data array with arbitrary dimensionality. If $cyc > 1$, j_1 is the cycle index. Other dimensions may correspond to a mode index or to a count number. The last index e is an error index, to address the data and sampling error. This is also used to index the comparison or experimental data, if it is available from the optional compare functions.

2.4.4 xGraph inputs

If *xGraph* is used, it is called with the arguments (*input*, *data*). Here *input* is a cell array, a sequence of graphics structures. It can be the same cell array used to define the simulation parameters. However, the graphics parameters are a distinct set. The *data* is the output data from xQSim, or some combination of the output data in more complex examples. One can run *xQSim* multiple times with different parameters to show functional dependences. The purpose of xGraph is to allow batch generation of many graphs from a multidimensional data array.

3 Observations and comparisons

3.1 Observe

The *observe* functions are user-specified functions that calculate each observable average and return it, given the phase-space amplitude matrix $\boldsymbol{a}$. Each *observe* function can have arbitrary output dimensionality. Each can be assigned its own graphics parameters, and an optional initialize function that initialises a given graph, specified by the graph number n. Parameters are passed to the observe functions through the parameter structure, p. Observe functions are passed as handles, in a cell array, as @x2 (etc).

Standard observe functions provided are:

Label	Return type	Description
x2	vector	Mean x quadrature squared per channel
p2	vector	Mean p quadrature squared per channel
n	vector	Mean photon number per channel
nm	vector	Mean number moments in sequence
k	vector	Mean clicks per channel
km	vector	Mean click moments in sequence
k1	vector	Click probability - single partition
kn	array	Click probability - n-fold partition

3.2 List of observe functions

3.2.1 x2

This computes the x quadrature moment per channel, $\langle \hat{x}_k^2 \rangle$.

3.2.2 p2

This computes the p quadrature moment per channel, $\langle \hat{p}_k^2 \rangle$.

3.2.3 n

This computes the mean photon number per channel, $\left\langle \hat{a}_k^\dagger \hat{a}_k \right\rangle = \langle \hat{n}_k \rangle$.

3.2.4 nm

This computes the mean photon number correlation over adjacent channels, $\langle \hat{n}_1 \hat{n}_2, \ldots \hat{n}_k \rangle$. The correlation order values are input as a cell array of index vectors, $p.On = [k_1, k_2, ..]$.

3.2.5 k

This computes the mean clicks per channel, from the click projectors, $\langle \hat{\pi}_k (1) \rangle$.

3.2.6 km

This computes the mean click correlation over a list of channels, $\langle \hat{\pi}_1 (1) \hat{\pi}_2 (1), \ldots \hat{\pi}_k (1) \rangle$. The correlation order values are input as a cell array of index vectors, $p.On = [k_1, k_2, ..]$.

3.2.7 k1

This calculates the total click probability in a partition of channels. The partition indices can be input as a cell array giving the number of channels, if the channels are numbered from $1 \ldots p.Part\{n\}$.

3.2.8 kn

This calculates the total click probability in a multi-dimensional partition of channels. The partition values can be input as a cell array of vectors containing the number of channels, if the channels are numbered sequentially from $1 \ldots p.Part\{1\}$. Otherwise, the individual channel number vectors are input in a nested cell array of vectors.

3.3 Compare

The *compare* function is a user-specified function that evaluates a function and returns it. This is used for testing and also to input experimental data for comparisons. The comparisons may include error data, which can be zero for exact results. Comparisons are also plotted using *compare*. This generates additional comparison plots, as well as error totals that are converted into a χ^2- error estimate when there are statistical variances available.

Label	Return type	Description
x2c	vector	Mean x quadrature squared per channel
p2c	vector	Mean p quadrature squared per channel
nc	vector	Mean photon number per channel
nmc	vector	Mean number moments in sequence
kc	vector	Mean clicks per channel
kmc	vector	Mean click moments in sequence
k1c	vector	Click probability - single partition
knc	array	Click probability - n-fold partition

In all analytic comparisons, there are two options:

- an identity transmission matrix can be used with an arbitrary Gaussian input, uniform or non-uniform.
- any unitary with a scaled transmission ($t \leq 1$) can be used with a fully thermalised ($eps = 1$) and uniform input.

3.3.1 x2c

This compares the analytic x quadrature moment per channel, $\langle \hat{x}^2 \rangle$.

3.3.2 p2c

This compares the analytic p quadrature vmoment per channel, $\langle \hat{p}^2 \rangle$.

3.3.3 nc

This gives a comparison of photon numbers per channel.

3.3.4 nmc

This comparison of number moments uses the same input as **nm**.

3.3.5 kc

This comparison of click probabilities per channel uses the same input as **k**.

3.3.6 kmc

This comparison of click moments uses the same input as **km**.

3.3.7 k1c

In this comparison the clicks are split into an n-fold partition, using the same input as **k1**.

3.3.8 knc

In this comparison the clicks are split into an n-fold partition, using the same input as **kn**.

4 Experimental data and χ^2 tests

This is generated by the Matlab program qcounts.m, which analyses experimental count data, from binary files in *.bin. It generates individual Matlab data files for each observable and records the total number of valid counts, which are also saved in the data files. All experimental data located in GBS_experimental_data should be on the MATLAB path if comparisons are required.

Label	Type	Description
@expmatrix	function handle	Experimental matrix, loaded in xQSim
@expsqueeze	function handle	Experimental squeezing data, loaded in xQSim
expk.mat	file	Experimental clicks per channel
expkm.mat	file	Experimental K-th order moment
expk1.mat	file	Experimental single partition clicks
expk2.mat	file	Experimental two-fold partition clicks
expk4.mat	file	Experimental four-fold partition clicks

4.1 Experimental comparisons

Using the *compare* function, experimental data is called from user specified functions which also sets error data to zero. Experimental error data is then calculated in xQSim and, following the χ-squared theory below, is converted into a χ-squared error estimate.

Experimental comparison functions are located in qDATA.

Label	Type	Description
expk	vector	Experimental click probability - Clicks per channel
expkm	vector	Experimental click probability - K-th order moment
expk1	vector	Experimental click probability - single partition
expk2	matrix	Experimental click probability - two-fold partition
expk4	matrix	Experimental click probability - four-fold partition

4.2 expk

In this comparison of experimental click probability per channel, experimental data is loaded from *expk.mat*.

4.3 expkm

In this comparison of experimental click probability per channel, experimental data is loaded from *expkm.mat*.

4.4 expk1

In this comparison of experimental click probability for a single partition of all channels, data is loaded from *expk1.mat*.

4.5 expk2

In this comparison of experimental click probability for an equal two-fold partition of all channels, data is loaded from *expk2.mat*.

4.6 expk4

In this comparison of experimental click probability for four-fold partition of all channels, data is loaded from *expk4.mat*.

4.7 Chi-square tests

The data generated and compared with experiment is in the form of probabilities of clicks or binned click patterns. This is tested using chi-square tests. If n observations in a random sample from a population are classified into classes with observed numbers x_i (for $i = 1, 2, \ldots, k$), and a null hypothesis gives the probability p_i that an observation falls into the i-th class, then the expected numbers are $m_i = np_i$.

The limiting distribution of the quantity given below is the χ^2 distribution:

$$\chi^2 = \sum_{i=1}^{k} \frac{(x_i - m_i)^2}{m_i}$$
$$= \sum_{i=1}^{k} \frac{(x_i/n - p_i)^2}{p_i/n}.$$

For our purposes, define an experimental probability estimate as $p_i^e = x_i/n$, then

$$\chi^2 = \sum_{i=1}^{k} \frac{(p_i^e - p_i)^2}{\sigma_i^2}$$

Here, $\sigma_i^2 = p_i/n$, is the estimated experimental variance. Typically, counts less that $x_i^{min} \sim 10$ are ignored, as the distribution is no longer Gaussian for small counts. The program produces χ^2 tests that include an estimate of simulation errors:

$$\sigma_i^2 = p_i/n + \sigma_i^{s2}$$

Note that this can also be used to compare simulations with other analytic comparison theories, not just with experiment.

5 Architecture and customization

5.1 xQSim data objects

The xQSim data objects include parameters and methods, but with an open, object-oriented architecture for extensibility. Most xQSim functions are modular and replaceable. This is as easy as just defining a new function handle to replace the default value. The xQSim code folder includes examples of how these are written. All output observables are divided up into a cell array of data graphs, generated in parallel.

5.2 Common functions

The user-definable functions, input and output field dimensions and calling arguments are:

Label	Arguments	Output dimensions
a=gen	*(a,p):* $a = (2M \times S_1)$, *p=[struct]*	$a = (2M \times S_1)$
o=observe{n}	*(a,p):* $a = (2M \times S_1)$, *p=[struct]*	Arbitrary: $m_l \times m_1 \times m_2 \ldots \times m_n$
c=compare{n}	*(p), p=[struct]*	Should match observe: $m_l \times m_1 \times m_2 \ldots \times m_n$

Notes

- ***gen* has a default, namely $qgen$, but this can be user modified**
- **gen input amplitudes are used for recycling. If this is not required, just use $(\sim, p)$:**
- **data outputs to xGraph include two extra dimensions, $m_l \times m_1 \times m_2 \ldots \times m_n \times m_e$**
- **The first graphics m_l dimension is reserved for plotting multiple lines on graphs, hence use $m_l = 1$ internally**
- **Obtaining multiple lines requires modifying the output data file before xGraph, or using the gfunction call in xGraph**
- **The m_e dimension is used for error bars. Here $m_e = 2$ is reserved, $m_e = 3$ stores the sampling error bars**

5.3 Customization guide

For example, to define your own stochastic variable generation function, include in the xQSim input the line:

```
p.gen=@Mygen;
```

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

```
function a = Mygen(a,p)
% a = Mygen(a,p) generates stochastic variables my way!
..
a = ...;
end
```

Similarly, to define your own observe function for data graph 6, include in the xQSim input the line:

```
p.observe{6}=@Myobserve;
```

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

```
function d = Myobserve(a,p)
% d = Myobserve(a,p) generates observables my way!
..
d = ...;
end
```

To define your own comparison function, include in the xQSim input the line:

```
p.compare{6} = @Mycompare;
```

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

```
function c = Mycompare(a,p)
% c = Mycompare(a,p) generates comparisons my way!
..
c = ...;
end
```

5.4 xQSim hints

- When using xQSim, it is a good idea to first run the batch test script, xQSimtest.m.
- xQSimtest.m tests your parallel toolbox installation. If you have no license for this, omit the third ensemble setting.
- To create a script, it is often easiest to start with an existing script with similar requirements: see Examples folder.
- Graphics parameters can also be included in the parameter structure **p**, either before or after calling *qcpsim*, to modify graphs.
- Comparison functions can be included to compare with analytic or experimental results.
- A full list of functionality is listed above.

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

Simulation metadata, including all preferred default values that were used in a particular simulation, is also stored in the xQSim output files. This is done in the Matlab *.mat* format, for simplicity so the entire simulation can be easily reconstructed or changed.

Some conventions are used to simplify inputs, as follows:

- **Most input data has default values**
- **Vector inputs of numbers are enclosed in square brackets, [...].**
- **Where multiple inputs of strings, functions or vectors are needed they should be enclosed in curly brackets, {...}, to create a cell array.**

- **Vector or cell array inputs with only one member don't require brackets.**
- **Incomplete vector or cell array inputs are filled in with the last applicable default or input value, where feasible.**
- **Default parameters can be checked by inspecting the output parameters, or setting print =2 in the inputs.**

In the output data arrays, the last index is $m_e = 1$ for the averages. The first error field, $m_e = 2$, is reserved for any systematic errors like step-sizes. The second error field, $m_e = 3$ is used for storing one standard deviation error-bar estimates due to computational sampling errors.

6 Public API

6.1 xQSim functions and objects

6.1.1 function xQSim (input)

This is the xQSim simulation function. It takes an input sequence of parameter structures that define a sequence of networks. Each network in the sequence can be repeated, with recycling of the amplitudes, which are then combined with new squeezed or coherent inputs.

6.2 xQSim main parameters

The following parameters can be applied to each member of the sequence. Some important xGraph parameters are included here for reference. These these are marked with a '*' if they are *only* used by xGraph. They can be omitted for simulation purposes, and added later if required.

6.2.1 M

Default: 1

Total number of input and/or output modes to the network.

- p.M >0

6.2.2 N

Default: 1

Number of input modes excited, less tyhan or equal M

- p.N >0
- p.M >0

6.2.3 cyc

Default: 1

Number of cycles of the network simulated.

- p.cyc >0

6.2.4 matrix

Default: @Identity

Function handle to generate the linear network matrix. Both Identity and random Unitary matrices are supplied. It can also be just an $M \times M$ complex matrix, for small test matrices entered inline.

- p.matrix = @function

6.2.5 ensembles

Default: [1,1,1]

Number of ensembles used in simulations. The first is the local vector ensemble. The second is the number of serial repeats. The third is the number or parallel repeats. Note that serial and parallel repeats are used for sampling error estimation.

- p.name = [ens1,ens2,ens3]

6.2.6 method

Default: 1

The phase-space method, where 1 = generalized P-representation, 2 = Wigner representation, 3 = Husimi (Q) representation

- p.method > 0

6.2.7 r

Default: 0

This gives a real squeezing vector that is input. It is expected to be of length at least $p.N$. If smaller, it will be expanded to length $p.N$. The entries with index greater than $p.N$ are set to zero. Can be replaced by a function call for large quantities of data.

- p.r

6.2.8 eps

Default: 0

This gives a real decoherence vector that is input. It is expected to be of length at least $p.N$. If smaller, it will be expanded to length $p.N$. The values should such that $0 \leq p.eps \leq 1$.

- p.eps

6.2.9 t

Default: 1

This gives a real transmission vector that is input. It is expected to be of length $p.M$. If smaller, it will be expanded to length $p.M$. The entries with index greater than $p.M$ are set to zero. This allows corrections to be made to the measured transmission matrix.

- $p.t \leq 1$

6.2.10 re

Default: 0

This gives a real recycling amplitude. It is expected to be of length $p.M$. If smaller, it will be expanded to length $p.M$. The entries with index greater than $p.M$ are set to zero. This allows previous amplitudes to be recycled.

- $p.re \leq 1$

6.2.11 observe

Default: {@(a,p) mean(real(a),2)}

Cell array of function handles to generate the observable phase-space averages. A large number of functions equivalent to commonly used hermitian operators are available.

- p.observe = {@function,..}

6.2.12 compare

Default: []

Cell array of function handles to generate the comparison data. A large number of functions equivalent to common hermitian operators are available for analytically known cases..

- p.compare = {@function,..}

6.3 xQSim secondary parameters

The following parameters are available also. These less frequently used xQSim input parameters are listed here. Also, an xGraph label parameter is included for reference. This is marked with a '*', and is *only* used by xGraph. These can all be omitted for simulation purposes, unless they are needed for special purposes like doing the χ^2 fitting. Note that they have defaults in all cases.

6.3.1 name

Default: ' '

Name used to label simulation, usually corresponding to the equation or problem solved, and will be passed to the xGraph program if it is used.

- p.name = 'your project name'

6.3.2 print

Default: 1

Print flag for output information while running xQSim. If "print = 0", most output is suppressed, while "print = 1" displays a progress report, and "print = 2" also generates a readable summary of the parameters as a record.

- p.print >= 0

6.3.3 pnames

Default: {'+P ','W ','Q '}

Cell vector of names used to label the phase-space representation, to allow future expansion or changes. Normally is not changed from default and can be omitted.

- p.pnames = {'your favorite method'}

6.3.4 *olabels

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.

- p.olabels{n} = 'string'

6.3.5 Tname

Default: matrix(0)

Name used to label type of network matrix. Usually it is returned by the matrix function to give the default name. However, other matrix names can be entered here.

- p.Tname = 'your transmission matrix name'

6.3.6 gen

Default: @qgen

Name of network quantum noise function. The default qgen function is used for GBS, and in this case, just omit *gen*.

- p.qgen = @yourgenerator

6.3.7 graphs

Default: number of observables

Number of distinct output averages calculated, normally set to the default value of the number of observe functions. This allows a reduced data output if required. This can be completely omitted if not required.

- $p.graphs > 0$

6.3.8 counts

Default: 0

Used to define the total counts for calculating count numbers and minimum count cutoffs when there is experimental data on experimental count probabilities.

- $p.counts > 0$

6.3.9 mincount

Default: 0

Used to set a lower bound on calculations for chi-square fits, when there is experimental data on experimental count probabilities. A typical value here is 10.

- $p.mincount > 0$

6.3.10 cutoff

Default: -1E-100

Used to define a lower cutoff, especially for probability data, where a small positive value, eg 10^{-7}, should be used. This is useful for log graphs and χ^2 fits.

- $p.cutoff$

6.3.11 mincount

Default: 0

Used to set a lower bound on calculations for χ^2 fits, when there is experimental data on experimental count probabilities.

- $p.mincount > 0$

References

[1] P. D. Drummond, B. Opanchuk, A. Dellios, M. D. Reid, Simulating complex networks in phase space: Gaussian boson sampling, arXiv:2102.10341.

[2] 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., Science 370, 1460 (2020).