xQSim Manual: v2.0

Peter D. Drummond, Alexander S. Dellios

May 4, 2023

Contents

1 Introduction

Large, network based experiments in quantum optics are becoming increasingly common, pushing the development of networks to ever large sizes. These optical linear networks have a variety of applications such as quantum computers implementing a type of random-number generation [1, 2, 3, 4, 5, 6] and large scale interferometers producing output measurements below the shot noise limit [7, 8].

Many implementations use nonclassical inputs [9] such as squeezed states or number states. These are not mathematically representable as a classical, positive, non-singular Glauber-Sudarshan phase-space distribution [10, 11].

Recently, large networks employed as quantum computers have been used to claim quantum computational advantage [12, 13, 14, 15]. This leads to a theoretical challenge: how does one check hardware operation? Are the random numbers generated correctly? Do they follow the expected probability distribution?

Traditional number-state validation methods are only practical for small scale networks, since they use an exponentially large basis set. This rules out direct calculations, due to the finite size, computational time and precision of a digital computer. Although more feasible, even Monte-Carlo methods take an exponentially long time digitally [16].

Other methods utilize marginal probabilities of the expected distribution, which are calculable [17, 18]. Although scalable, such methods cannot verify higher-order correlations generated in the network. Therefore, these methods can only validate part of a larger picture.

For Gaussian boson sampling with linear networks, other methods using quantum phase-space exist. These do allow comparisons of theory and experiment for large, nonclassical networks, simulating all possible correlations generated by an experiment. A computational implementation is necessary, however. xQSim - meaning e**x**tensible **Q**uantum **Sim**ulations using phase-space representations - is a software package that does this.

While xQSim is able to compare theoretical predictions with experimental data, it cannot generate a direct random output of photon counts. This is the task implemented by experiments [13, 14, 15] and some validation methods [17]. However, there is extensive theory suggesting that such brute-force computations are #P hard [1, 2, 3, 19, 4, 5, 6], and therefore our code implements a completely different approach.

To do this, we employ three phase-space simulation methods: the positive P-distribution [20] ($method = 1$), the Wigner distribution [21] ($method = 2$), provided it has a positive distribution for the corresponding input state, and the Husimi Q-function [22] ($method = 3$).

The present code distribution can simulate multidimensional photon probabilities, photon number correlations and quadrature moments for a variety of inputs, networks and output measurements. It uses sampling and binning to obtain computable results, giving error-bars comparable to those of the experimental measurements. In addition to simulating Gaussian quantum inputs, admixtures of thermal noise can be included to model phase decoherence in the input states.

Saturating or click detectors require the use of $method = 1$. This is also preferred for

intensity correlations, due to the exponentially lower sampling error obtained. Quadrature measurements are best simulated with $method = 2$. However, intensity correlation and quadrature probabilities can be simulated using all methods.

The code also generates arbitrarily binned moments. These are vital for large-scale comparisons with data, due to the exponential sparsity of the full probability distributions. It can also extract experimental moments from available public datasets, which depends on the available data and the encoding employed.

The programs provided are written in Matlab, which is a proprietary language of The Mathworks Inc. It is fully compatible with Octave, which is a free public domain clone of Matlab. Parallel features are only available in Matlab currently, and require the parallel toolbox to give large speed improvements on multicore computers.

A GPU/Python version exists [23], for GPU enabled supercomputers. While this does not implement all the features of the Matlab version, it is faster, with up to $16,000$ qubits demonstrated. For user's familiar with the original xQSim version [24], four new features have been introduced:

- Random permutations: Users can now randomly permute output binary patterns, changing the order of the binning for multi-dimensional simulations. This allows an exponential number of possible experimental comparisons to occur, with each permutation simulating a different correlation.
- New data: Experimental data and extraction code corresponding to the larger experiment from Ref. [14].
- Z-score, or Z-statistic, tests: An additional statistical test which allows users to determine how randomly distributed each output pattern is.
- xGraph3 compatibility: This version is fully compatible with xGraph3, a companion batch graphics program that includes extensive graphical and statistical comparisons.

Sections 2 through 4 outline all the necessary background theory implemented in xQSim. Additional theoretical details can also be found in Refs. [25, 26, 24, 27]. A description of the main simulation code *xQSim* is given in section 5 whilst descriptions of observables, comparison functions used for testing and experimental data are presented in sections 5.5 and 6.

Finally, numerous examples of various input states and output distributions are presented in section. 7, with a general reference to xQSim functions and parameters provided in section. 8.

2 Theoretical background

This section provides a comprehensive theoretical background to the general theory of linear networks, including input inputs to the network, detection mechanisms and observable correlations.

2.1 Linear photonic networks

The numerical code solves for output observables obtained after a linear transformation of a multi-mode quantum input state $\hat{\rho}^{(\text{in})}$. The mode transformation is generated by a linear photonic network of generalized beam-splitters and phase delays. This acts as an M-mode interferometer such that output modes are linear combinations of each input mode.

Without losses, the network itself is defined by an $M \times M$ unitary matrix $\boldsymbol{U}$. Physically, this corresponds to the interference of input photons that generates large amounts of entanglement due to the exponential number of paths available to photons.

In the ideal, lossless case, one has:

$$\hat{a}_i^{(\text{out})} = \sum_{j=1}^{M} U_{ij} \hat{a}_j^{(\text{in})}, \tag{1}$$

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, losses in the network are commonplace, thus causing the experimental transmission matrix to be non-unitary. Therefore, lossy networks are denoted by the transmission matrix $\boldsymbol{T}$. Not every input channel needs to have an input. In these cases, $N \subset M$ represents the number of input modes, thus changing the unitary to an $N \times M$ transmission matrix.

These give a different transformation law, where:

$$\hat{a}_i^{(\text{out})} = \sum_{j=1}^{N} T_{ij} \hat{a}_j^{(\text{in})} + \sum_{j=1}^{M} B_{ij} \hat{b}_j^{(\text{in})}, \tag{2}$$

Here, the M operators $\hat{b}_i^{(\text{in})}$are noise operators. These are necessary to conserve the operator commutation relations. They comprise inputs from the reservoirs that cause losses, as well as the $M - N$ vacuum inputs at unused ports. They are independent commuting operators, whose reservoirs are all in a vacuum state.

Due to the need to conserve commutators, we know that for both inputs and outputs:

$$\begin{aligned} [\hat{a}_i, \hat{a}_j] &= 0 \\ \left[\hat{a}_i, \hat{a}_j^{\dagger}\right] &= \delta_{ij}. \end{aligned} \tag{3}$$

Applying this to the outputs, considering a vacuum state input, and taking expectation values, gives

$$\begin{aligned} \delta_{ij} &= \left\langle \left[\hat{a}_i^{(\text{out})}, \hat{a}_j^{\dagger(\text{out})}\right] \right\rangle \\ &= \sum_k \left(T_{ik} T_{jk}^* + B_{ik} B_{jk}^*\right). \end{aligned} \tag{4}$$

Next, we can define a new $M \times M$ matrix

$$\boldsymbol{D} = \boldsymbol{B}\boldsymbol{B}^\dagger = \boldsymbol{I} - \boldsymbol{T}\boldsymbol{T}^\dagger. \tag{5}$$

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.

2.2 Quantum input states

The input state is defined in a number of ways depending on the desired distribution one wishes to sample from. If each input mode is independent, $\hat{\rho}^{(\text{in})}$ is a product of input states.

To sample from the permanent, inputs are single photon Fock states such that

$$\hat{\rho}^{(\text{in})} = \prod_{j=1}^{M} |1\rangle_j \langle 1|_j \,, \tag{6}$$

where $|1\rangle_j = a_j^\dagger |0\rangle$.

To sample from the Hafnian or Torontonian, the input is a product of single-mode pure squeezed vacuum states

$$\hat{\rho}^{(\text{in})} = \prod_{j=1}^{M} |r_j\rangle \langle r_j| \,, \tag{7}$$

where $\boldsymbol{r} = [r_1, \ldots, r_M]$ is the squeezing vector and

$$\begin{aligned} |r_j\rangle &= \hat{S}(r_j)\,|0\rangle \\ &= \exp\left(r_j \frac{(a_j^\dagger)^2}{2} - r_j \frac{\hat{a}_j^2}{2}\right)|0\rangle \,, \end{aligned} \tag{8}$$

is the squeezed vacuum state where we have assumed the squeezed phase is zero.

Currently, xQSim 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.

2.2.1 Pure squeezed states

For a single-mode, squeezed vacuum states are generated by applying the squeezing operator $\hat{S}(r)$ onto a vacuum state. As is clear from the operator definition Eq.(8), where we drop the j subscript for the single-mode case, squeezed vacuum states always generate even numbers of photons.

The mean photon number $\bar{n} = \left\langle \hat{a}^\dagger \hat{a} \right\rangle$ and coherence $m = \left\langle (\hat{a})^2 \right\rangle$ can be derived using the relations

$$\begin{aligned} \hat{S}^\dagger(r)\hat{a}\hat{S}(r) &= \hat{a}\cosh(r) - \hat{a}^\dagger \sinh(r) \\ \hat{S}^\dagger(r)\hat{a}^\dagger\hat{S}(r) &= \hat{a}^\dagger\cosh(r) - \hat{a}\sinh(r), \end{aligned} \tag{9}$$

where the mean photon number is

$$\begin{aligned} \bar{n} &= \left\langle \hat{a}^\dagger \hat{a} \right\rangle \\ &= \langle r| \hat{a}^\dagger \hat{a} |r\rangle \\ &= \langle 0| \hat{S}^\dagger(r)\hat{a}^\dagger\hat{a}\hat{S}(r) |0\rangle \\ &= \sinh^2(r), \end{aligned} \tag{10}$$

whilst the coherence is

$$\begin{aligned} m &= \left\langle (\hat{a})^2 \right\rangle \\ &= \langle r| \hat{a}\hat{a} |r\rangle \\ &= \langle 0| \hat{S}^\dagger(r)\hat{a}\hat{a}\hat{S}(r) |0\rangle \\ &= \sinh(r)\cosh(r). \end{aligned} \tag{11}$$

For pure squeezed states, the coherence and photon number are related via $m^2 - \bar{n} = \bar{n}^2$.

The superposition of only even Fock states becomes clearer when expanding the squeezed state in terms of Fock states as

$$|r\rangle = \frac{1}{\sqrt{\cosh(r)}} \sum_{n=0}^{\infty} \frac{\sqrt{(2n)!}}{2^n n!} \tanh^n(r) \, |2n\rangle \,, \tag{12}$$

where $n = 0, 1, 2, \ldots$ is the number of photons. From the above Fock state expansion the photon number distribution for the squeezed vacuum state is

$$\begin{aligned} P_{2n} &= \frac{1}{\cosh(r)} \frac{(2n)!}{(n!)^{2n} 2^{2n}} (\tanh(r))^{2n}, \\ P_{2n+1} &= 0, \end{aligned} \tag{13}$$

with variance $\sigma_n^2 = 2\bar{n}(\bar{n}+1) = \bar{n}(1 + \cosh(2r_j))$.

Squeezed states are minimum uncertainty states and are therefore defined entirely by their quadrature variances. Using the quadrature operators

$$\begin{aligned} \hat{x} &= \hat{a} + \hat{a}^\dagger \\ \hat{y} &= -i\left(\hat{a} - \hat{a}^\dagger\right), \end{aligned} \tag{14}$$

which obey the commutation relation $[\hat{x}_j, \hat{y}_k] = 2i\delta_{jk}$, the normally ordered x-quadrature variance is obtained as

$$\begin{aligned}\left\langle :(\Delta\hat{x})^2: \right\rangle &= \left\langle \hat{x}^2 \right\rangle \\ &= 2(n+m) \\ &= e^{2r} - 1,\end{aligned} \tag{15}$$

whilst the normally ordered y-quadrature variance is

$$\begin{aligned}\left\langle :(\Delta\hat{y})^2: \right\rangle &= \left\langle \hat{y}^2 \right\rangle \\ &= 2(n-m) \\ &= e^{-2r} - 1.\end{aligned} \tag{16}$$

All the results derived here are also valid for multiple modes, denoted by the subscript j, given each mode is independent as is the case in optical networks. Currently, xQSim can simulate pure and thermalized squeezed states as well as classical thermal state inputs into the network. This is achieved using a model for thermal squeezed states which alters the multi-mode input coherence $m(r_j)$ as $\tilde{m}(r_j) = (1-\epsilon)m(r_j)$ whilst keeping the input photon number $n(r_j) = \bar{n}_j$ unchanged. This allows users to easily interpolate between thermal, $\epsilon = 1$, and pure squeezed, $\epsilon = 0$, states.

2.2.2 Squeezed thermal states

Thermal states are classical states with fluctuations larger than the vacuum limit such that their quadrature variances are $\left\langle :(\Delta\hat{x})^2: \right\rangle = \left\langle :(\Delta\hat{y})^2: \right\rangle$.

In terms of Fock states, the thermal state density matrix is

$$\hat{\rho} = \frac{1}{1+\bar{n}} \sum_{n=0}^{\infty} \left(\frac{\bar{n}}{1+\bar{n}} \right)^n |n\rangle \langle n| , \tag{17}$$

which gives the well known photon number distribution

$$P(n) = \frac{\bar{n}^n}{(\bar{n}+1)^{n+1}}. \tag{18}$$

Thermal states can be used to generate squeezed thermal states with initial occupation n_{th}, which gives [28]:

$$\begin{aligned}\bar{n} &= n_{th} + (2n_{th}+1)\sinh^2(r) \\ \tilde{m} &= (2n_{th}+1)\sinh(r)\cosh(r).\end{aligned} \tag{19}$$

In the thermalized case, the relationship between coherence and photon number is modified, since to eliminate r one must use the relationship that

$$\begin{aligned}\frac{\tilde{m}^2}{(2n_{th}+1)^2} &= \sinh^2(r)\left(1+\sinh^2(r)\right)\\ &= \frac{\bar{n}-n_{th}}{(2n_{th}+1)}\left(1+\frac{\bar{n}-n_{th}}{(2n_{th}+1)}\right)\end{aligned}$$

Therefore:

$$\begin{aligned}\tilde{m}^2 &= (\bar{n}-n_{th})(1+\bar{n}+n_{th})\\ &= \bar{n}+\bar{n}^2-\left(n_{th}^2+n_{th}\right). \end{aligned} \tag{20}$$

Squeezed thermal states are used as a test case, since one can define the saturating, or click, detectors in terms of the photon number and coherence as explained below. An example of such a test is also given in subsection 7.1.

2.3 Photon counting

Photonic networks employed as quantum computers aim to sample from an output state whose distribution corresponds to the $\#P$-hard matrix permanent, Hafnian or Torontonian distributions. Which output probability is sampled depends on the input states to the network, with Fock states corresponding to the permanent and squeezed states corresponding to the Hafnian or Torontonian distributions, where the difference between these distributions comes from the detector used.

The sampled distribution not only depends on the input state but also the detector type. When photon-number resolving (PNR) detectors are used one samples either the permanent, given the input is a Fock state, or the Hafnian distribution for a squeezed state input. The Torontonian corresponds to the use of saturating, or click, detectors with squeezed state inputs.

Output samples from linear networks consist of photon count patterns. The j-th detector records $c_j = 0, 1, 2, \ldots$ photon counts, with a specific output pattern being denoted by the count vector $\boldsymbol{c}$.

From standard photon counting theory, the projection operator for observing $c_j = 0, 1, 2, \ldots$ counts is denoted by

$$\hat{p}_j(c_j) = \frac{1}{c_j!} : (\hat{n}'_j)^{c_j} e^{-\hat{n}'_j} :, \tag{21}$$

where $: \cdots :$ denotes normal ordering and $\hat{n}'_j = a_j^{\dagger(\text{out})} a_j^{(\text{out})}$ is the output photon number.

For PNR detectors, which can discriminate between photon numbers, each detector is defined by the above projector, with the projection operator for a specific output pattern given as

$$\hat{P}(\boldsymbol{c}) = \bigotimes_{j=i}^{M} \hat{p}_j(c_j). \tag{22}$$

The expectation value of the PNR pattern projection operator corresponds to the Hafnian

$$\text{Haf} = \left\langle \hat{P}(\boldsymbol{c}) \right\rangle, \tag{23}$$

which is $\#P$-hard to compute at large M.

Click detectors saturate for more than one count at a detector. Therefore, outputs are binary with $c_j = 1$ denoting a detection event, even if multiple photons hit the same detector, and $c_j = 0$ is no detection event. From Eq.(21), the click projection operator is obtained by summing over all $c_j > 0$ counts such that

$$\begin{aligned} \hat{\pi}(1) =: & \sum_{c_j>0} \frac{(\hat{n}'_j)^{c_j}}{c_j!} e^{-\hat{n}'_j} : \\ & = 1 - e^{-\hat{n}'_j}, \end{aligned} \tag{24}$$

which gives the standard saturating detector projection operator

$$\hat{\pi}_j(c_j) =: e^{-\hat{n}'_j} \left(e^{\hat{n}'_j} - 1 \right)^{c_j} : . \tag{25}$$

The projection operator for an pattern output is then similarly defined as

$$\hat{\Pi}(\boldsymbol{c}) = \bigotimes_{j=i}^{M} \hat{\pi}_j(c_j), \tag{26}$$

where the expectation value corresponds to the Torontonian distribution

$$\text{Tor} = \left\langle \hat{\Pi}(\boldsymbol{c}) \right\rangle. \tag{27}$$

2.3.1 Exact outputs

To test photon count distributions, xQSim uses a modified version of the click probabilities which can be computed exactly in the limit of a unit transmission matrix.

Using the thermal state photon number distribution Eq.(18), one obtains the click distributions:

$$\begin{aligned} \pi(0) = P(0) &= \frac{1}{\bar{n}+1} \\ \pi(1) = 1 - \pi(0) &= \frac{\bar{n}}{\bar{n}+1}. \end{aligned} \tag{28}$$

Substituting the thermal squeezed state modified photon number the probability of a

vacuum state is [28]:

$$\begin{aligned}\pi(0) &= \frac{1}{n_{th}+1}\left(1+\frac{2n_{th}+1}{(n_{th}+1)^2}\sinh^2(r)\right)^{-(1/2)} \\ &= \left(\bar{n}-n_{th}+(n_{th}+1)^2\right)^{-(1/2)} \\ &= \left(1+\bar{n}+n_{th}+n_{th}^2\right)^{-(1/2)} \\ &= \left((1+\bar{n})^2-\tilde{m}^2\right)^{-(1/2)}.\end{aligned}$$

Hence, by using known results for a photon number distribution of a squeezed thermal state [28], combined with the definitions of $\bar{n}$ and $\tilde{m}$, the probability of a vacuum state and a non-vacuum or click state are:

$$\begin{aligned}\langle\hat{\pi}(0)\rangle &= \frac{1}{\sqrt{(1+\bar{n})^2-\tilde{m}^2}} \\ \langle\hat{\pi}(1)\rangle &= 1-\frac{1}{\sqrt{(1+\bar{n})^2-\tilde{m}^2}}.\end{aligned} \tag{29}$$

Example applications of this result is presented in subsection 7.1.

2.4 Intensity correlations

xQSim generates comparisons of two types of correlations: Glauber intensity correlations and grouped correlations, 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 only valid for click detectors. Although methods are available which convert PNR detectors to click detectors, direct binning methods for PNR outputs will be included in a sequential updates.

Glauber's n-th order intensity correlation is defined as

$$G^{(n)}(c_j) = \left\langle : (\hat{n}'_j)^{c_j} \dots (\hat{n}'_M)^{c_M} : \right\rangle, \tag{30}$$

where $n = \sum c_j$ is the correlation order. Multi-mode Glauber correlations determine the probability of detecting n photons at n 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

$$G^{(2)} = \left\langle a_1^{\dagger(\text{out})} a_2^{\dagger(\text{out})} a_2^{(\text{out})} a_1^{(\text{out})} \right\rangle, \tag{31}$$

corresponds to detecting one photon at mode 1 and one at mode 2.

Upon reordering using the standard Bose commutation relations Eq.(3), one obtains

$$G^{(2)} = \left\langle a_1^{\dagger(\text{out})} a_2^{(\text{out})} \right\rangle \left\langle a_2^{\dagger(\text{out})} a_1^{(\text{out})} \right\rangle + \left\langle a_1^{\dagger(\text{out})} a_1^{(\text{out})} \right\rangle \left\langle a_2^{\dagger(\text{out})} a_2^{(\text{out})} \right\rangle . \tag{32}$$

The first term describes non-local correlations which is the interference of photons between detectors, whilst the second term describes the photon intensity at a detector or 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

$$P_N = \left\langle \prod_j \hat{n}'_j \right\rangle , \tag{33}$$

as we assume photons do not interfere at detectors, removing non-local correlations.

2.5 Grouped correlations

Grouped count probabilities (GCPs) are the main observable correlation implemented by xQSim and are only valid for click detectors.

GCPs are defined as

$$\mathcal{G}_{\boldsymbol{S}}^{(n)}(\boldsymbol{m}) = \left\langle \prod_{j=1}^{d} \left[\sum_{\sum c_i = m_j} \hat{\Pi}_{S_j}(\boldsymbol{c}) \right] \right\rangle , \tag{34}$$

where $\boldsymbol{m} = (m_1, \dots, m_d)$ are the observed grouped counts in d-dimensions 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.

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

$$\begin{aligned} S_1 &= \left\{ 1, \dots, \frac{M}{2} \right\} \\ S_2 &= \left\{ \frac{M+2}{2}, \dots, M \right\} . \end{aligned} \tag{35}$$

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

$$\frac{\binom{M}{M/d}}{d} = \frac{M!}{d(M/d)!(M-M/d)!}, \tag{36}$$

possible ways of generating $m_1, \ldots, m_d$ 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

$$\begin{aligned} \boldsymbol{S} &= (S_1, S_2) = (\{1,2\},\{3,4\}), \\ \boldsymbol{S} &= (S_1, S_2) = (\{1,3\},\{2,4\}), \\ \boldsymbol{S} &= (S_1, S_2) = (\{1,4\},\{2,3\}). \end{aligned} \tag{37}$$

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 $S = \{1, \ldots, M\}$. This is also the case when simulating marginal probabilities, which are obtained by setting $n < M$ such that $M - n$ inputs are ignored. The first-order marginal, $n = 1$, is called the click correlation, $\langle \hat{\pi}_j(1) \rangle$, and determines the probability of observing a click at the j-th detector.

3 Phase-space sampling

To simulate output probabilities, xQSim uses phase-space representations. As already explained, xQSim *does not* generate photon counts, which is the $\#P$-hard task implemented by experiments. Instead, phase-space representations allow users to compute *distributions and moments* of the resulting output distributions of linear networks.

The outputs in phase-space are continuous real or complex variables whose stochastic moments are equal to moments of the experimental distributions, apart from sampling errors due to finite numbers of experimental and theoretical counts. This assumes that the parameters are precisely known, and do not have noise or fluctuations.

3.1 Glauber P-representation

The M-mode Glauber diagonal P-representation expands the density matrix as a sum of diagonal coherent state projectors

$$\hat{\rho} = \int P(\boldsymbol{\alpha}) \left|\boldsymbol{\alpha}\right\rangle \left\langle\boldsymbol{\alpha}\right| d^{2M}\boldsymbol{\alpha}, \tag{38}$$

where the distribution $P(\boldsymbol{\alpha})$ is a positive and non-singular distribution over multimode coherent state amplitudes $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_M]$ for classical states.

However, the diagonal P-representation famously breaks down for certain quantum states, generating non-positive and singular distributions. This is due to the lack of off-diagonal coherent state amplitudes needed to represent such nonclassical superpositions.

3.2 Positive P-representation

Part of a family of generalized P-representations developed to extend Glauber's diagonal P-representation to quantum states [20], the normally ordered positive-P representation always generates a non-singular and positive distribution for any quantum state.

The density matrix is defined as an expansion over a multidimensional subspace of the complex plane:

$$\hat{\rho} = \iint P(\boldsymbol{\alpha}, \boldsymbol{\beta}) \hat{\Lambda}(\boldsymbol{\alpha}, \boldsymbol{\beta}) \mathrm{d}^{2M}\boldsymbol{\alpha} \mathrm{d}^{2M}\boldsymbol{\beta}, \tag{39}$$

where $P(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is the positive-P distribution over coherent state amplitudes $\boldsymbol{\alpha}, \boldsymbol{\beta}$. The off-diagonal coherent state projector

$$\hat{\Lambda}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{|\boldsymbol{\alpha}\rangle \langle \boldsymbol{\beta}^*|}{\langle \boldsymbol{\beta}^* | \boldsymbol{\alpha} \rangle}, \tag{40}$$

doubles the classical phase-space dimension, allowing off-diagonal amplitudes $\boldsymbol{\beta} \neq \boldsymbol{\alpha}^*$ to exist.

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 $P(\boldsymbol{\alpha}, \boldsymbol{\beta}) = P(\boldsymbol{\alpha})\delta(\boldsymbol{\alpha}^* - \boldsymbol{\beta})$. However, for squeezed states, this will lead to the singular behavior that is already known.

Moments of the positive-P distribution are equivalent to normally ordered operator moments

$$\begin{aligned} \left\langle \hat{a}_{j_1}^\dagger, \ldots, \hat{a}_{j_n} \right\rangle &= \langle \beta_{j_1}, \ldots, \alpha_{j_n} \rangle_P \\ &= \iint P(\boldsymbol{\alpha}, \boldsymbol{\beta}) [\beta_{j_1}, \ldots, \alpha_{j_n}] \mathrm{d}^{2M}\boldsymbol{\alpha} \mathrm{d}^{2M}\boldsymbol{\beta}, \end{aligned} \tag{41}$$

where $\langle \ldots \rangle$ denotes a quantum expectation value and $\langle \ldots \rangle_P$ is the positive-P probability average.

Although the diagonal P-representation is unsuitable to simulate squeezed or entangled states, other classical phase-space distributions exist with positive distributions for quantum states. These are the symmetrically orders Wigner representation and anti-normally ordered Q-function.

3.3 Wigner representation

The M-mode Wigner representation is defined as the Fourier transform of the symmetrically ordered characteristic function such that

$$W(\boldsymbol{\alpha}) = \frac{1}{\pi^{2M}} \int \mathrm{d}^2\boldsymbol{z} \mathrm{Tr}\left\{\hat{\rho} e^{i\boldsymbol{z}(\hat{a}-\boldsymbol{\alpha})+i\boldsymbol{z}^*(\hat{a}^\dagger-\boldsymbol{\alpha}^*)}\right\}, \tag{42}$$

where Tr $\{\dots\}$ is the trace and $\boldsymbol{z}$ is a complex vector.

Although the Wigner function always exists as a real-valued function on phase-space for the density operator, or any other hermitian operator, the resulting probability distribution need not be positive. This is why the Wigner distribution is referred to as a quasi-probability. However for thermal and squeezed states, the Wigner distribution is positive.

Despite the positive distribution, the Wigner function is only valid for symmetrically ordered operator products. Symmetric ordering, denoted $\{\dots\}_{sym}$, is the average over all possible combinations of creation and annihilation operators, for example

$$\{\hat{a}^\dagger a\}_{sym} = \frac{1}{2}(\hat{a}\hat{a}^\dagger + \hat{a}^\dagger\hat{a}) \tag{43}$$

$$\{\hat{a}^\dagger\hat{a}^2\}_{sym} = \frac{1}{3}(\hat{a}^2\hat{a}^\dagger + \hat{a}\hat{a}^\dagger\hat{a} + \hat{a}^\dagger\hat{a}^2). \tag{44}$$

This ordering requirement makes applications to normally ordered detectors both cumbersome, as one must reorder all operators to normal order, and inaccurate, as seen from Eq.(43), where the expectation value of the symmetrically ordered number operator becomes

$$\left\langle \{\hat{a}^\dagger a\}_{sym} \right\rangle = |\alpha|^2 + \frac{1}{2}. \tag{45}$$

Therefore, the Wigner function adds half a quantum of vacuum noise per mode causing a rapid increase sampling errors, making the Wigner function unsuitable for simulations of network output probabilities.

However, the Wigner function is ideal for simulating squeezed state quadrature operators, Eqs.(), as these are measured via homodyne detectors which are symmetrically ordered.

3.4 Q-function

The standard form of the anti-normally ordered M-mode Q-function is

$$Q(\boldsymbol{\alpha}) = \frac{1}{\pi^M} \langle \boldsymbol{\alpha}|\hat{\rho}|\boldsymbol{\alpha}\rangle, \tag{46}$$

and, like the Wigner function, can be expressed as the Fourier transform of the anti-normally ordered characteristic function.

The Q-function distribution is always positive but is only defined for anti-normally ordered operator products with moments being obtained as

$$\left\langle \hat{a}_{j_1}, \ldots, \hat{a}^{\dagger}_{j_n} \right\rangle = \left\langle \alpha_{j_1}, \ldots, \alpha^{*}_{j_n} \right\rangle_{Q}$$
$$= \int Q(\boldsymbol{\alpha})[\alpha_{j_1}, \ldots, \alpha^{*}_{j_n}]\mathrm{d}^{2M}\boldsymbol{\alpha}, \tag{47}$$

where $\langle \ldots \rangle_Q$ denotes a Q-distribution average.

However, like the Wigner function, operators must be reordered for applications to normally ordered detectors. Using the standard bosonic commutation relations, Eqs.(3), the expectation value of the number operator is

$$\left\langle \{a\hat{a}^{\dagger}\}_{anti} \right\rangle = |\alpha|^2 + 1. \tag{48}$$

The Q-function adds a quantum of vacuum noise per mode, generating the largest increase in sampling errors of any phase-space representation when used to simulate normally ordered detectors. This accumulation of vacuum noise for multimode networks rapidly causes the Q-function to become inaccurate for simulating linear networks.

3.5 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.

The ability to define a common ordering scheme arises from writing the Wigner and Q-function distributions as convolutions of the positive P-representation

$$P_{\sigma}(\boldsymbol{\alpha}) = \frac{1}{(\pi\sigma)^M} \int P(\boldsymbol{\alpha}_0, \boldsymbol{\beta}_0) e^{-(\boldsymbol{\alpha}-\boldsymbol{\alpha}_0)(\boldsymbol{\alpha}^*-\boldsymbol{\beta}_0)/\sigma} \mathrm{d}^{2M}\boldsymbol{\alpha}\mathrm{d}^{2M}\boldsymbol{\beta}, \tag{49}$$

where $P_\sigma(\boldsymbol{\alpha})$ is a σ-ordered representation, $P(\boldsymbol{\alpha}_0, \boldsymbol{\beta}_0)$ is the positive-P distribution and $\boldsymbol{\alpha}_0, \boldsymbol{\beta}_0$ are used to denote the normal ordered nonclassical phase-space variables whilst $\boldsymbol{\alpha}, \boldsymbol{\alpha}^*$ denotes a classical phase-space which is valid for $\sigma = 1/2, 1$.

Operator moments for any ordering can now be obtained via

$$\left\langle \left\{ \hat{a}^{\dagger}_{j_1}, \ldots, \hat{a}_{j_n} \right\}_{\sigma} \right\rangle = \left\langle \alpha^{*}_{j_1}, \ldots, \alpha_{j_n} \right\rangle_{\sigma}$$
$$= \int P_{\sigma}(\boldsymbol{\alpha})[\alpha^{*}_{j_1}, \ldots, \alpha_{j_n}]\mathrm{d}^{2M}\boldsymbol{\alpha}. \tag{50}$$

4 Numerical methods

The section outlines the numerical methods and statistical tests implemented by xQSim to simulate linear network output distributions and perform comparisons with either experimental data, or exact tests.

4.1 Input-output samples

To simulate linear networks in phase-space, one must first generate initial stochastic samples. This is achieved using the σ-ordering scheme as stochastic samples for any Gaussian input state in any representation are generated following:

$$\begin{aligned}\alpha_j &= \frac{1}{2}\left(\Delta_{\sigma x_j} w_j + i\Delta_{\sigma y_j} w_{j+M}\right)\\ \beta_j &= \frac{1}{2}\left(\Delta_{\sigma x_j} w_j - i\Delta_{\sigma y_j} w_{j+M}\right),\end{aligned} \tag{51}$$

where $\langle w_j w_k\rangle = \delta_{jk}$ are real Gaussian noises and

$$\begin{aligned}\Delta^2_{\sigma x_j} &= 2(n_j + \sigma + \tilde{m}_j)\\ \Delta^2_{\sigma y_j} &= 2(n_j + \sigma - \tilde{m}_j),\end{aligned} \tag{52}$$

are thermal squeezed state quadrature variances which are altered from the pure squeezed state definitions Eqs.(15) and (16).

For normally ordering, the input amplitudes $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are converted to outputs as

$$\begin{aligned}\boldsymbol{\alpha}' &= \boldsymbol{T}\boldsymbol{\alpha}\\ \boldsymbol{\beta}' &= \boldsymbol{T}^*\boldsymbol{\beta},\end{aligned} \tag{53}$$

which follows from Eq.(1). 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

$$\boldsymbol{D} = \boldsymbol{I} - \boldsymbol{T}^\dagger\boldsymbol{T}, \tag{54}$$

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

$$\boldsymbol{\alpha}' = \boldsymbol{T}\boldsymbol{\alpha} + \sqrt{\frac{\sigma}{2}}\boldsymbol{B}(\boldsymbol{u} + i\boldsymbol{v}), \tag{55}$$

where $\boldsymbol{\beta}' = \boldsymbol{\alpha}'^*$ as we are in a classical phase-space.

4.2 Grouped correlations computation

GCPs are readily simulated in phase-space using the positive-P representation by replacing the normally ordered projection operator Eq.(25) with the positive-P observable

$$\pi_i(c_i) =: e^{-n'_i}\left(e^{n'_i} - 1\right)^{c_i}, \tag{56}$$

where $n'_i = \alpha'_i\beta'_i$ is the output photon number.

The summation over exponentially many patterns implemented by GCPs (see Eq.(34) is simulated using a multidimensional inverse discrete Fourier transform

$$\tilde{\mathcal{G}}^{(n)}_{\boldsymbol{S}}(\boldsymbol{k}) = \left\langle \prod_{j=1}^{d} \bigotimes_{i\in S_j} \left(\pi_i(0) + \pi_i(1)e^{-ik_j\theta_j}\right) \right\rangle_P ,$$
$$\mathcal{G}^{(n)}_{\boldsymbol{S}}(\boldsymbol{m}) = \frac{1}{\prod_j (M_j+1)} \sum_{\boldsymbol{k}} \tilde{\mathcal{G}}^{(n)}_{\boldsymbol{S}}(\boldsymbol{k}) e^{i\sum k_j\theta_j m_j}, \tag{57}$$

where $\theta_j = 2\pi/(M_j+1)$ and $k_j = 0,\ldots,M_j$.

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.

4.3 Statistical tests

The data generated and compared with experiment is in the form of probabilities of clicks or binned click patterns. xQSim implements two statistical tests: chi-square and Z-statistic tests.

4.3.1 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 $p^e_i = x_i/n$, then

$$\chi^2 = \sum_{i=1}^{k} \frac{(p^e_i - p_i)^2}{\sigma^2_i}$$

Here, $\sigma^2_i = p_i/n$, is the estimated experimental variance. This is only valid for click detectors, or PNR detectors when photon flux is small, as binary outputs can be modeled as Poissonian. For larger photon numbers this is not the case as multiple photons can be detected simultaneously.

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.

4.3.2 Z-statistic tests

The Z-statistic, or Z-score, test determines how many standard deviations a test statistic, X, is from its expected, normally distributed mean and is defined as

$$Z = \frac{X - \mu}{\sigma},$$

where μ and σ are the mean and standard deviation of a normal distribution, respectively.

In the limit $k \to \infty$, the chi-square distribution of the test statistic approaches a normal distribution via the central limit theorem. Typically this convergence is slow, however using the Wilson-Hilferty (WH) transformation, an accurate convergence can be achieved when $k \geq 10$.

Using the WH transformed chi-square statistic $\left(\chi^2/k\right)^{1/3}$, the Z-statistic of the chi-square distribution is

$$Z = \frac{\left(\chi^2/k\right)^{1/3} - (1 - 2/(9k))}{\sqrt{2/(9k)}},$$

for comparisons with $k \geq 10$ distinct classes.

The Z-statistic is used to determine the randomness of output patterns, as large Z values could indicate experimental output patterns are not occurring randomly. A threshold of $Z > 6$ is used to define outputs with extremely small probabilities of occurring.

The Z-statistic becomes particularly powerful when used in conjunction with random permutations of output patterns, as repeatedly observing large Z values indicates outputs with very small probabilities are being continuously observed. This could indicate systematic errors in an experimental network are causing non-random behavior, such as more $0's$ than $1's$ in binary patterns, to occur.

5 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.

5.1 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 section. 7 and 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
```

5.2 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 there is only one simulation in a sequence. Otherwise, the cell array defines a sequence of network transformations.

In general, 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. Full input parameter lists are in Sec (8).

The most used parameter inputs are:

Label	Type	Default	Description
deco	string	”	Decoherence factors used for optimization
CO	integer	None	Correlation order
compare	function handles	[]	Optional comparison functions
ensembles	vector	[1,1,1]	Size of vector, serial, parallel ensemble
eps	vector	0	An n-dimensional decoherence factor
M	integer	1	Total number of modes
matrix	matrix function	Identity	An $M \times M$ transfer matrix function
method	integer	1	Phase-space method: $m = 1, 2, 3$
N	integer	1	Number of excited input modes
name	character	”	Name of simulation
O	vector	[]	Vector of various correlation orders
observe	function handles	$\{@(a, p)\, a\}$	Observable functions
permute	vector	$[1, 2, 3, \ldots, M]$	Random permutation of outputs
r	vector or function	0	An n-dimensional squeezing vector
t	vector	1	An n-dimensional transmission factor

- 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. One dataset is generated per observe function, but this can produce more than one plot, depending on the graphics options specified.

5.3 Simulation notes

5.3.1 General

- 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.

- The *gen* code generates the input, checks if random permutations are present and simulates the network.

5.3.2 Optimizing decoherence & transmission

When using xQSim to compare theoretical and experimental quantum networks, it is recommended that the first simulation use $eps = 0$ and $t = 1$. This corresponds to pure squeezed state inputs as $eps = \epsilon$ (see subsection. 2.2) whilst t corresponds to measurement errors in the transmission matrix.

Once completed, users may wish to simulate decoherence effects. Although exact values of *eps* and t are up to users discretion, it is recommended to use Matlab's *fminsearch* function to find optimal fitting values. These correspond to the lowest obtainable χ^2 output. An example of how to implement *fminsearch* in xQSim is given below for the script *xQSim_165W_experiment.m* contained in the folder *xQSimExamples*.

To start *fminsearch* the following prompt is inserted into the Command Window:

```
%Inital conditions:
%x0(1) = t, x0(2) = eps

x0 = [1,0]
x = fminsearch(@xQSim_165W_experiment, x0)%Call fminsearch
```

This will run xQSim multiple times for different parameters of *eps* and t until it reaches a minimum at which point it will output the optimal values of *eps* and t.

Some general considerations when using *fminsearch* is that it will likely take multiple hours to complete, assuming the user has access to Matlab's parallel computing toolbox. Without the parallel toolbox computation time will increase significantly. If this is the case, a manual optimizing route may be more efficient.

To reduce computation time, *fminsearch* can be stopped before it has found the absolute minimum. This is due to two reasons:

1. Since xQSim simulates stochastic samples, output probabilities will vary slightly between runs causing *fminsearch* to run longer than may be necessary.

2. χ^2 output differences between simulation runs becomes negligible once *eps* and t are accurate to $\approx 4/5$ decimal places. Since *fminsearch* will continue repeating simulations until a minimum is found, the increase in computation for greater decimal accuracy may outweigh the negligible improvement in χ^2 results.

If the user wishes to have the option of stopping *fminsearch* early, it is recommended to use the parameter *p.deco*. *fminsearch* doesn't output the values of *eps* and t it uses in each simulation run, only outputting them once it's fully completed. Therefore, *p.deco* prints the values of *eps* and t before each simulation run, allowing the user to know what decoherence factors the most recent simulation run has used.

Caution - When running *fminsearch* be sure to either comment out or remove the graphics program *xGraph* from the script as once *fminsearch* is complete, Matlab will generate plots for every simulation run. These comparison plots are not necessary when optimizing decoherence and transmission and may cause Matlab to crash.

```
function e = xQSim_165W_experiment(x)

p.name = '144-mode: Waist=65um, power=1.65W';
p.matrix     = @expmatrix;
p.M          = 144;
p.N          = 50;
p.r          = @expsqueeze;
p.ensembles = [5000,20,12];
p.t                = x(1);
p.eps       = x(2);
p.deco    = fprintf('\n t =%d, epsilon=%d\n',p.t,p.eps);
p.counts     = 40183178;
p.mincount   = 10;
p.cutoff     = 1e-7;
p.observe    = {@k,@k1};
p.compare    = {@expk,@expk1};
p.logs{2}    = [0,1];
p.diffplot   = {2,2};
p.glabels    = {{'j'},{'m'}};
p.olabels    = {'<\pi_j(1)>','G^{(M)}(m)'};
p.xk{2}      = {0:p.M};
[e,d,cp]     = xQSim(p);
```

5.3.3 Data extraction

Data from both 100-mode and 144-mode experiments of Zhong et al [13, 14] have been extracted and are contained within the folder *xQSimGBSExperiments*. Due to their size, raw data files are not provided but can be obtained from [29, 30].

If the user is required to extract additional observable data, xQSim contains three data extraction algorithms:

- *qcounts*: Extracts data from binary files with big endian encoding. Corresponds to experiment [13]. Algorithm is run separately from the input m-file simulation script and is only valid when $M = 100$.

- *qcountsbe*: Extracts data from the same binary encoding as *qcounts* but is integrated into the input m-file and is valid for any mode number.

- *qcountsle*: Extracts data from binary files with little endian encoding. Corresponds to experiment [14]. Like *qcountsbe*, *qcountsle* is integrated into the input m-file and is valid for any mode number.

An example implementation of *qcountsle* is supplied below. To allow phase-space simulations to be performed immediately after extraction, the experimental counts integer *Count* is output. When using *qcountsbe* or *qcountsle*, the integer variable *bits* is required to input the bit-type of the encoded data file. Although *qcountsbe* and *qcountsle* have been generalized, this is limited to data from experiments [13, 14] since every experiment may encode data differently.

The only time user's may wish to extract additional data is to perform random permutation tests, which require permutations to be applied before binning. In this case, one can use *qcounts_permute* and *qcountsle_permute* for 100-mode and 144-mode data, respectively. The only addition in these scripts compared to the standard extraction code is the use of Matlab's *randperm* function. The $1 \times M$ random permutation vector applied to each pattern is then saved as *randp.mat*, which is called from the main Matlab directory using the function *xqpermutation* and set as the par mater *p.permute*. This permutation vector is then applied to the rows of the transmission matrix for use in phase-space simulations.

More details on random permutations can be found in the appropriate example located in section 7.

```
function e = xQSim_165W_experiment()

p.name = '144-mode: Waist=65um, power=1.65W';
p.matrix     = @expmatrix;
p.M          = 144;
p.N          = 50;
p.bits     = 32;
p.r          = @expsqueeze;
p.ensembles = [5000,20,12];
p.t          = 1.00;
p.eps      = 0;
[Count]      = qcountsle(p);
p.counts     = Count;
p.mincount   = 10;
p.cutoff     = 1e-7;
p.observe    = {@k,@k1};
p.compare    = {@expk,@expk1};
p.logs{2}    = [0,1];
p.diffplot   = {2,2};
p.glabels    = {{'j'},{'m'}};
p.olabels    = {'<\pi_j(1)>','G^{(M)}(m)'};
p.xk{2}      = {0:p.M};
[e,d,cp]     = xQSim(p);
xGraph(d,cp);
```

5.4 Data structures in xQSim and xGraph

5.4.1 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 be used directly, without an enclosing cell array.

5.4.2 xQSim internal phase-space

Although this is transparent to the user, each network in the sequence transforms a matrix of quantum phase-space 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.

User's can vary between phase-space representations using *method* which corresponds to the σ-ordering notation via $method = 1$ is equal to $\sigma = 0$, $method = 2$ to $\sigma = 1/2$ and $method = 3$ to $\sigma = 1$.

5.4.3 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 generated 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.

5.4.4 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.

5.5 Observations

The *observe* functions are user-specified functions that calculate an 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 initializes 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
km2	vector	Mean click moments over two sequential channels
km3	vector	Mean click moments over three sequential channels
kmsub	vector	Mean subset of click moments per CO channels
k1	vector	Click probability - single partition
kn	array	Click probability - n-fold partition

5.6 List of observe functions

5.6.1 x2

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

5.6.2 p2

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

5.6.3 n

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

5.6.4 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, $p.On = [k_1, k_2, ..]$.

5.6.5 k

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

5.6.6 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, $p.On = [k_1, k_2, ..]$.

5.6.7 km2

This computes the mean click correlation over two channels, $\langle \hat{\pi}_j(1)\,\hat{\pi}_k(1) \rangle$, where $j < k$. The number of possible combinations follows the binomial coefficient $\binom{M}{n}$, where M is the number of modes and n is the correlation order which is input as the integer $p.CO$.

5.6.8 km3

This computes the mean click correlation over three channels $\langle \hat{\pi}_j(1)\,\hat{\pi}_k(1)\,\hat{\pi}_h(1) \rangle$, where $j < k < h$. The number of possible combinations follows the binomial coefficient $\binom{M}{n}$, where M is the number of modes and n is the correlation order which is input as the integer $p.CO$.

5.6.9 kmsub

This computes the first $M - CO$ mean click correlations from a possible $\binom{M}{n}$. This gives a more graphically appealing output than $km2$ and $km3$ observables.

5.6.10 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 $1 \ldots p.Part\{n\}$.

5.6.11 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.

5.7 Comparisons

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
km2c	vector	Mean clicks per two channels
km3c	vector	Mean clicks per three channels
kmsubc	vector	Mean subset of clicks per CO channels
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 thermalized ($eps = 1$), uniform input.

5.7.1 x2c

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

5.7.2 p2c

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

5.7.3 nc

This gives a comparison of photon numbers per channel.

5.7.4 nmc

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

5.7.5 kc

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

5.7.6 kmc

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

5.7.7 km2c

This comparison of all combinations of click probabilities over two channels uses the same input as **km2**.

5.7.8 km3c

This comparison of all combinations of click probabilities over three channels uses the same input as **km3**.

5.7.9 kmsubc

This comparison of a subset of all combinations of click probabilities over sequential channels uses the same input as **kmsub**.

5.7.10 k1c

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

5.7.11 knc

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

6 Experimental data

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 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
expk2m.mat	file	All experimental second order click correlations
expk2ms.mat	file	Subset of experimental second order click correlations
expk3m.mat	file	All experimental third order click correlations
expk3ms.mat	file	Subset of experimental third order click correlations
expk4ms.mat	file	Subset of experimental fourth order click correlations
expk1.mat	file	Experimental single partition clicks
expk2.mat	file	Experimental two-fold partition clicks
expk4.mat	file	Experimental four-fold partition clicks

6.1 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
expk2m	vector	Experimental click probability - All combinations of clicks per two channels
expk2ms	vector	Experimental click probability - Subset of clicks per two channels
expk3m	vector	Experimental click probability - All combinations of clicks per three channels
expk3ms	vector	Experimental click probability - Subset of clicks per three channels
expk4ms	vector	Experimental click probability - Subset of clicks per four channels
expk1	vector	Experimental click probability - single partition
expk2	matrix	Experimental click probability - two-fold partition
expk4	matrix	Experimental click probability - four-fold partition

6.2 expk

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

6.3 expkm

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

6.4 expk2m

In this comparison of experimental click probabilities for all combinations of two output channels, experimental data is loaded from *expk2m.mat*.

6.5 expk2ms

In this comparison of experimental click probabilities for a subset of all two channel combinations, experimental data is loaded from *expk2ms.mat*.

6.6 expk3m

In this comparison of experimental click probabilities for all combinations of three output channels, experimental data is loaded from *expk3m.mat*.

6.7 expk3ms

In this comparison of experimental click probabilities for a subset of all three channel combinations, experimental data is loaded from *expk3ms.mat*.

6.8 expk4ms

In this comparison of experimental click probabilities for a subset of all four channel combinations, experimental data is loaded from *expk4ms.mat*.

6.9 expk1

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

6.10 expk2

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

6.11 expk4

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

7 Examples

The folder *xQSimExamples* contains a variety of examples for different mode numbers, input states and observables. Individual tests can be performed or all can be run using batch test scripts *xQSim_ GBStestM.m*, where M varies between $M = 20, 40, 100$ for different batch tests.

The examples presented in this manual are slightly modified versions of the examples given in *xQSimExamples*.

7.1 Squeezed thermalized state examples

Using the exact derivation of click projectors outlined in subsection. 2.3.1 for the comparison functions, special multimode test cases are used, in which all output events are independent. This is obtainable either from having arbitrary independent thermalized squeezed inputs and a diagonal transmission matrix, or else by having a uniform, fully thermalized input together with an arbitrary unitary transmission matrix.

Either of these cases lead to independent outputs in which each mode is in a Gaussian state. The simplest case is for a uniform output in which $\langle \hat{\pi}(1) \rangle = p$ is constant. In this case, the probability of m total clicks in M channels has a probability of:

$$P(m) = \frac{M! p^m (1-p)^{M-m}}{m!(M-m)!}. \tag{58}$$

Two examples of this are given below.

7.1.1 Uniform squeezing

This example generates total count probabilities for the case of pure squeezed state inputs with uniform squeezing $\boldsymbol{r} = [1, \ldots, 1]$. The transmission matrix is an $N \times M$ identity matrix with $M = N = 20$ and a transmission coefficient of $t = 0.5$.

In this case, the χ^2 statistical test outputs give $\chi^2/k \approx 1 \pm 0.3$ and the Z-statistic gives $Z \approx 1 \pm 2$. This indicates that the statistical test is satisfied.

```
function e1 = xQSim_uni_sq_test20( )
p.matrix      = @Identity;
p.M           = 20;
p.N           = 20;
p.name        = sprintf('+P uniform squeezed, M=%d',p.M);
p.t           = 0.5;
p.r           = ones(1,p.M);
p.eps         = zeros(1,p.M);
p.Part{1}     = p.M;
p.ensembles   = [1000,100,12];
p.cutoff      = 1.e-7;
p.observe     = {@k1};
p.compare     = {@k1c};
p.glabels     = {{'Clicks m'}};
p.olabels     = {'G_1(m)'};
p.diffplot    = {2};
p.xk{1}       = {0:p.M};
[e1,d,cp]     = xqsim(p);
xgraph(d,cp);
```

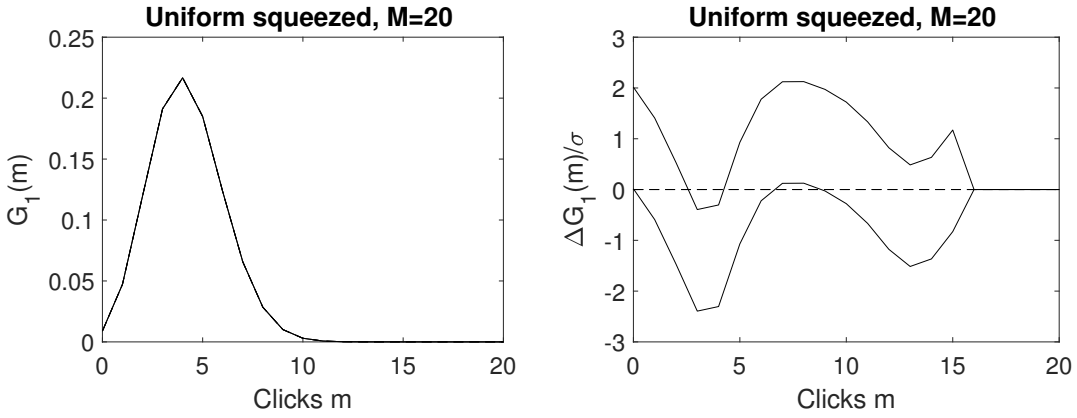

Figure 1: Example: $M = 20$ mode linear network with uniform squeezed state inputs. Left graph is simulation versus exact total count probabilities. Right plot is the normalized difference between simulated (solid line) and exact (dotted line) distributions. Simulations obtained for 1.2×10^6 ensembles.

7.1.2 Thermal state

This example generates total count probabilities for the case of thermal state inputs with squeezing parameter $\boldsymbol{r} = [1, \ldots, 1]$ and $\epsilon = 1$. The transmission matrix is an $N \times M$

Haar random unitary with $M = N = 100$ and a transmission coefficient of $t = 0.5$.

χ^2 statistical test outputs give $\chi^2/k \approx 1 \pm 0.4$ for $k = 45$ valid bins whilst the Z-statistic gives $Z \approx 1.5 \pm 2$.

```
function e1 = xQSim_thermal_test100( )
p.matrix      = @Unitary;
p.M           = 100;
p.N           = 100;
p.name        = sprintf('+P  unitary thermal, M=%d',p.M);
p.t           = 0.5;
p.r           = ones(1,p.M);
p.eps         = ones(1,p.M);
p.Part{1}     = p.M;
p.ensembles   = [1000,100,12];
p.cutoff      = 1.e-7;
p.observe     = {@k1};
p.compare     = {@k1c};
p.glabels     = {{'Clicks m'}};
p.olabels     = {'G_1(m)'};
p.diffplot    = {2};
p.xk{1}       = {0:p.M};
[e1,d,cp]     = xqsim(p);
xgraph(d,cp);
```

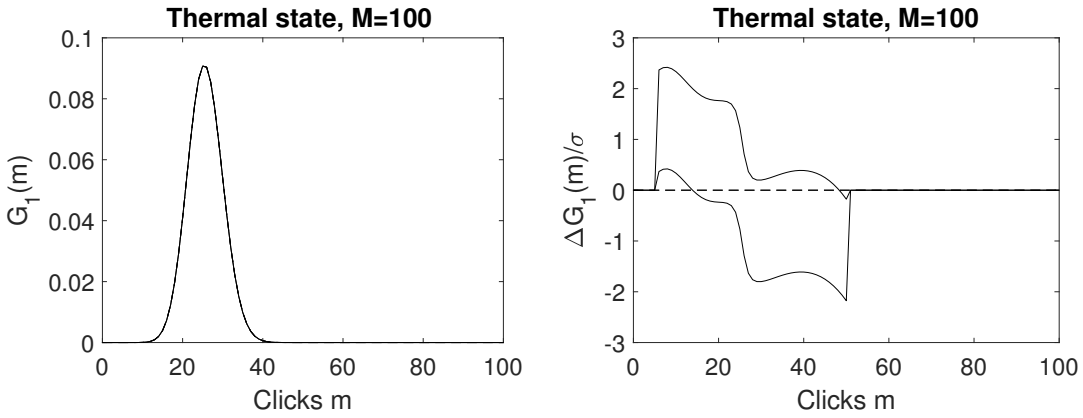

Figure 2: Example: $M = 100$ mode linear network with thermal state inputs. Left graph is simulation versus exact total count probabilities. Right plot is the normalized difference between simulated (solid line) and exact (dotted line) distributions. Simulations obtained for 1.2×10^6 ensembles.

7.2 Multidimensional binning

This example generates output GCPs in two and four dimensions for the case of non-uniform squeezed inputs. The transmission matrix is an $N \times M$ identity matrix with $M = N = 40$ and a transmission coefficient of $t = 0.5$.

For simulations with $d > 1$, xGraph generates four output plots; two 3D surface plots and two 2D plots. When $d = 2$, one surface plot is the full two-dimensional distribution, whilst the other is the difference comparison plot. For $d > 2$ the surface plots are then two-dimensional planar slices of, by default, grouped counts m_1, m_2 although this can be changed. Standard 2D plot outputs are always one-dimensional slices of surface plots.

Two-dimensional comparisons give χ^2 statistical test outputs of $\chi^2/k \approx 1 \pm 0.5$ for $k = 169$ valid bins whilst the Z-statistic gives $Z \approx 0.7 \pm 2$. Four-dimensional comparisons give $\chi^2/k \approx 1 \pm 0.5$ for $k = 2083$ valid bins and $Z \approx 1.4 \pm 2$.

```
function e1 = xQSim_non_uni_sq_test40( )
p.matrix      = @Identity;
p.M           = 40;
p.N           = 40;
p.name        = sprintf('+P non-uniform squeezed, M=%d',p.M);
p.t           = 0.5;
I             = ones(1,p.M/5);
p.r           = [I/4,2*I,4*I,I/2,I];
p.eps         = zeros(1,p.M);
p.Part{1}     = [p.M/2, p.M/2];
p.Part{2}     = [p.M/4, p.M/4,p.M/4,p.M/4];
p.ensembles   = [1000,100,12];
p.cutoff      = 1.e-7;
p.observe     = {@kn,@kn};
p.compare     = {@knc,@knc};
p.glabels     = {{'Clicks m_1','Clicks m_2'},...
     {'Clicks m_1','Clicks m_2','Clicks m_3','Clicks m_4'}};
p.olabels     = {'G_2(m)','G_4(m)'};
p.diffplot    = {2,2};
p.xk{1}       = {0:p.M/2,0:p.M/2};
p.xk{2}       = {0:p.M/4,0:p.M/4,0:p.M/4,0:p.M/4};
[e1,d,cp]     = xqsim(p);
xgraph(d,cp);
```

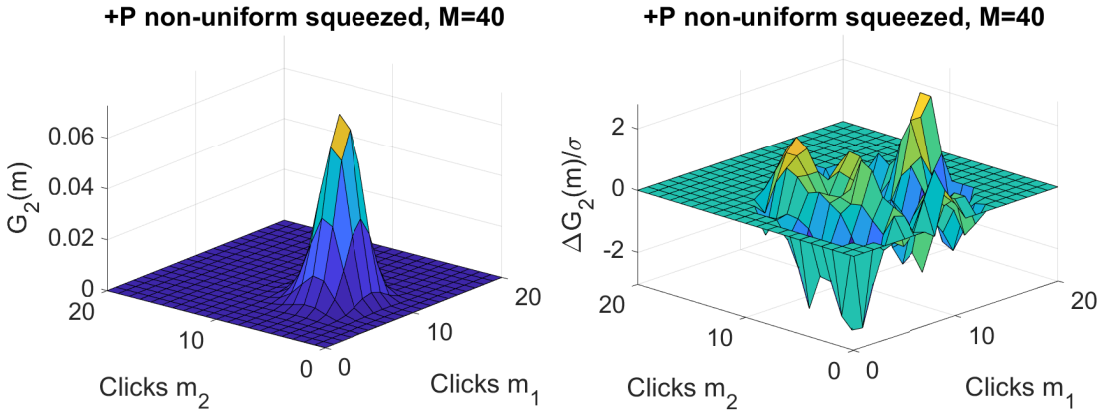

Figure 3: Example: $M = 40$ mode linear network with non-uniform squeezed state inputs. Left surface plot is simulation versus exact two-dimensional output probability for all 21^2 data points. Right plot is the normalized difference between simulated and exact distributions. Simulations obtained for 1.2×10^6 ensembles.

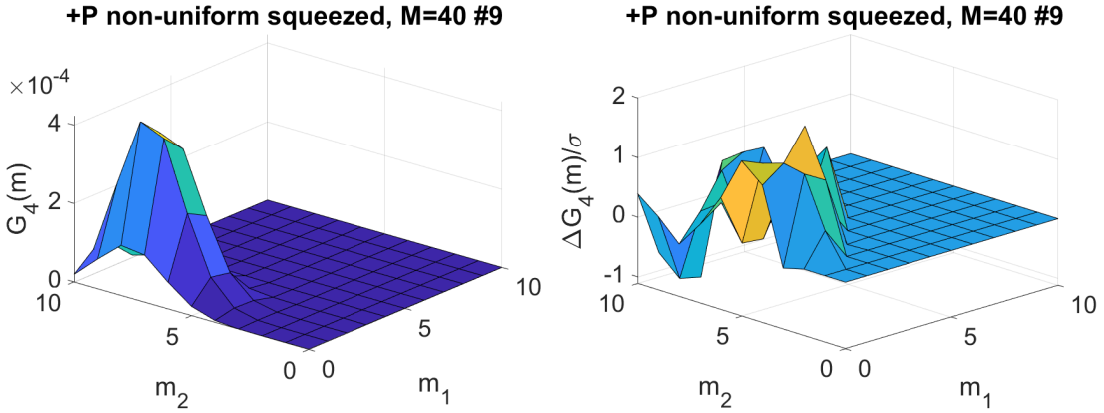

Figure 4: Example: $M = 40$ mode linear network with non-uniform squeezed state inputs. Left surface plot is a two-dimensional planar slice of simulation versus exact four-dimensional output probability for 11^2 data points. Right plot is the normalized difference between simulated and exact distributions. Simulations obtained for 1.2×10^6 ensembles.

7.3 Phase-space comparisons

This example generates intensity correlation moments, $\left\langle \hat{n}'_1 \dots \hat{n}'_j \right\rangle$, of orders $1 \to 5$ for non-uniform thermalized squeezed state inputs in the positive-P, Wigner and Q phase-space representations with $\epsilon = 0.5$. The transmission matrix is an $N \times M$ identity matrix with $M = N = 20$ and a transmission coefficient of $t = 0.5$.

Changing between representations is simple and can be performed on one script as

shown below. However, Wigner and Q representations can only be used to compute observables with photon number outputs or quadrature operator moments.

```
function e1 = xQSim_ps_test20( )
p.matrix      = @Identity;
p.M           = 20;
p.N           = 20;
p.name        = sprintf('+P non-uniform thermalized, M=%d',p.M);
p.t           = 0.5;
I             = ones(1,p.M/5);
p.r           = [I/4,2*I,4*I,I/2,I];
p.eps         = 0.5*ones(1,p.M);
p.O{1}        = 1:5;
p.ensembles   = [1000,100,12];
p.cutoff      = 1.e-7;
p.observe     = {@nm};
p.compare     = {@nmc};
p.glabels     = {{'Order'}};
p.olabels     = {'<n_1...n_j>'};
p.diffplot    = {2};
p.xk{1}       = p.O(4);

pW            = p;
pW.method     = 2;
pW.name       = sprintf('W non-uniform thermalized, M=%d',p.M);

pQ            = p;
pQ.method     = 3;
pQ.name       = sprintf('Q non-uniform thermalized, M=%d',p.M);

[e1,d,cp]     = xqsim({p,pW,pQ});
xgraph(d,cp);
```

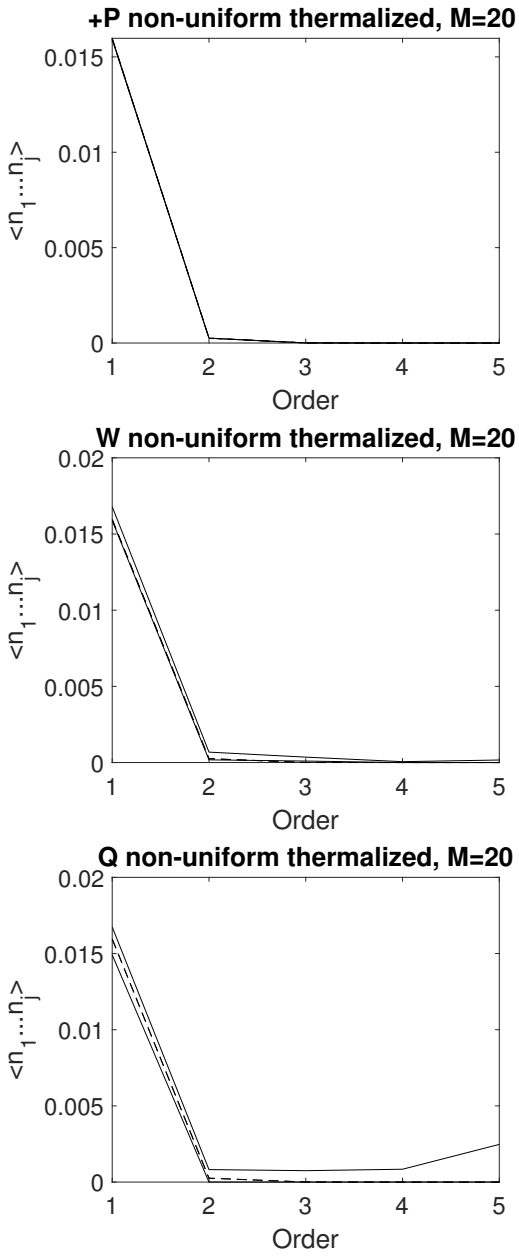

Figure 5: Example: Intensity correlation moments versus correlation order. Simulations are denoted by the solid line and use the positive-P (top), Wigner (middle) and Q (bottom) representations whilst the dashed lines denotes exactly computed correlation moments. While the normally ordered positive-P probabilities agree with the exact calculations, the Wigner and Q representations deviate significantly from exact calculations which increase with correlation order. This deviation is due to vacuum fluctuations causing sampling errors increasing with correlation order.

7.4 Experimental examples

This subsection contains examples for simulations using experimental data from the 100-mode implementation of Zhong et al [29, 13]. Although multiple examples are given in the *xQSimExamples* folder, we focus on generating comparisons of higher-order click correlations and random permutations of binary patterns in this manual.

7.4.1 High-order correlations

This example generates comparisons of click correlation moments of first and second order which are defined as

$$\langle \hat{\pi}_j(1) \rangle$$

$$\langle \hat{\pi}_j(1) \hat{\pi}_k(1) \rangle \, .$$

Comparisons can be obtained for all possible combinations of output modes, which follows the binomial coefficient $\binom{M}{n}$, or for a subset of these combinations. The subset comparisons are preferable for graphing, whilst the full comparison of output modes is preferable for statistical tests to ensure all possible correlations are observed. Not observing all correlations will alter the statistical test outputs as there are fewer valid bins.

Inputs are thermalized squeezed states with decoherence coefficient $eps = 0.0932$ and transmission $t = 1.0235$. Simulations were performed for 1.44×10^7 ensembles. First-order correlations have statistical test outputs of $\chi^2/k \approx 5.25 \times 10^3$ and $Z \approx 347$, for $k = 100$, whilst second-order marginals give $\chi^2/k \approx 4.99 \times 10^3$ and $Z \approx 2.4 \times 10^3$, for $k = 4950$.

```
function e = xQSim_experiment_HOC( )
p.name       = 'Click correlation ,M=100'
p.matrix     = @expmatrix;
p.M          = 100;
p.N          = 50;
p.r          = @expsqueeze;
p.CO{2}      = 2;
p.CO{3}      = 2;
p.ensembles  = [6000,200,12];
p.t          = 1.0235;
p.eps        = 0.0932;
p.counts     = 51392341;
p.mincount   = 10;
p.cutoff     = 1e-7;
p.observe    = {@k,@kmsub,@km2};
p.compare    = {@expk,@expk2ms,@expk2m};
p.diffplot   = {2,2,2};
p.glabels    = {{'j'},{'j,k'}{'j<k'}};
p.olabels    = {'<\pi_j(1)>','G^{(2)}','<\pi_j(1)\pi_k(1)>'};
p.xk{2}      = {0:p.M-1};
p.xk{3}      = {0:4950};
[e,d,cp]     = xqsim(p);
xgraph(d,cp);
```

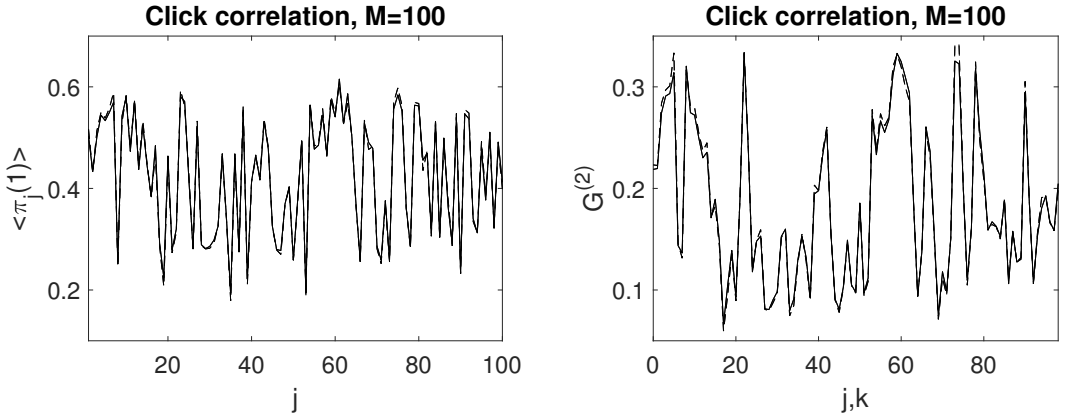

Figure 6: Example: Comparisons of click correlations of experiment (dashed line) versus theory (solid line). The left graph plots correlations of first-order for all possible combinations of output modes. Right graph contains only the first 99 second-order correlations for graphical simplicity.

7.4.2 Random permutations

This example generates comparisons of two-dimensional GCPs for binary patterns that have been randomly permuted. Inputs are thermalized squeezed states with decoherence coefficient $eps = 0.0932$ and transmission $t = 1.0235$. Data for three random permutations from this experiment are available in the folder *xQSimGBSExperiments*, however more can be readily obtained.

Permutations are input through the parameter *p.permute*, which is assumed to be a $1 \times M$ row vector, and are applied to the transmission matrix, changing the row order according to the mode ordering of the permutation. User's can either manually permute output modes, e.g. $p.permute = [15, 8, 44, \dots]$, or use permutations generated from data extraction algorithms which are loaded from relevant .mat files by the function *xqpermutation* (see subsection. 5.3.3). Note, manual permutation vectors are only applied to the transmission matrix rows, not the binary data which must be permuted before being binned.

Simulations can then be performed, with total count and click correlations unaffected by the permutation. Since each permutation produces comparisons of different correlations, each test will output different χ^2 and Z-statistic results as well as normalized difference graphs. An example of this is given in the below figure.

```
function e = xQSim_experiment_permute( )
p.name       = 'Random permutation 1'
p.matrix     = @expmatrix;
p.M          = 100;
p.N          = 50;
p.r          = @expsqueeze;
p.Part{3}    = [50, 50];
p.Part{4}    = [25,25,25,25];
p.ensembles  = [5000,20,12];
p.t          = 1.0235;
p.eps        = 0.0932;
p.counts     = 51392341;
p.mincount   = 10;
p.cutoff     = 1e-7;
p.permute    = xqpermutation;
p.observe    = {@k,@k1,@kn};
p.compare    = {@expk,@expk1,@expk2};
p.logs{2}    = [0,1];
p.logs{3}    = [0,0,1];
p.diffplot   = {2,2,2};
p.glabels    = {{'j'},{'m'},{'Clicks m_1','Clicks m_2'}};
p.olabels    = {'<\pi_j(1)>','G_1(m)','G_2(m)'};
p.xk{2}      = {0:p.M};
p.xk{3}      = {0:50,0:50};
[e,d,cp]     = xqsim(p);
xgraph(d,cp);
```

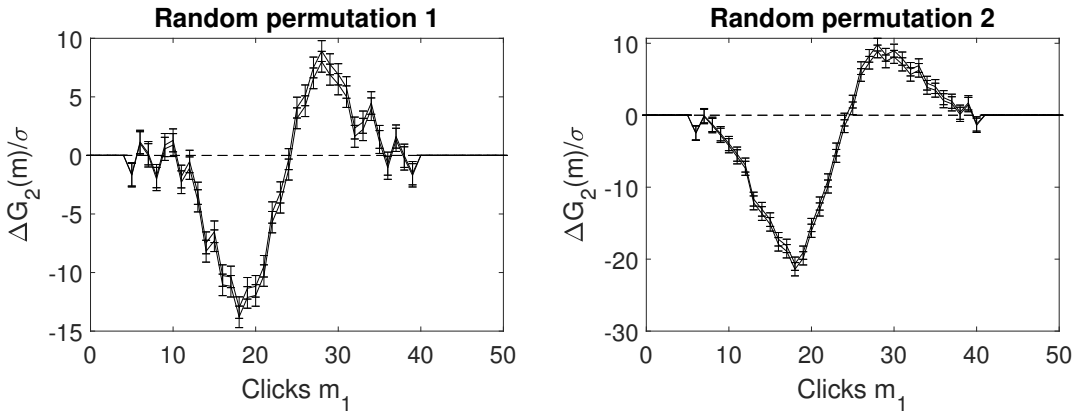

Figure 7: Example: Normalized difference of theory versus experiment of two-dimensional binning from two separate random permutations. Left graph is a one-dimensional slice of the full distribution with statistical test outputs of $\chi^2/k \approx 28$ for $k = 973$ and $Z \approx 134$. Right graph is also a one-dimensional slice of the full distribution with a different random permutation applied. Statistical test outputs are $\chi^2/k \approx 47$ for $k = 976$ and $Z \approx 173$ for this permutation. Both simulations were performed for 1.2×10^6 ensembles.

8 xQSim reference

This section gives a reference guide to the xQSim parameters and functions.

8.1 Included programs

The package provided has six parts:

1. **xQSimCode.** This contains all the main quantum simulation codes, from observable and comparison functions to quantum phase-space sample generating functions.

2. **xQSimExamples.** This has the scripts used to simulate experiments for comparisons, as well as scripts used for testing.

3. **xQSimData.** This has scripts used for simulating data from recent Gaussian boson sampling experiments.

4. **xQSimDocumentation.** This provides the documentation, including this report and a separate manual for the graphics program *xGraph*.

5. **xGraph.** This contains a multidimensional graphics program which is also used in the stochastic differential equation software package *xSPDE*.

6. **xQSimGBSExperiments.** This has recent public experimental data from Refs. [13, 14], and data extraction codes for reference purposes.

8.2 xQSim function call

The simulation function takes an input 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. Any input parameters or functions that are omitted are given appropriate default values.

Simulations carried out by the code are performed by other specialized internal functions. Input parameters come from an **input** cell array of structures, while output is saved in a **data** array, and optionally in a file. During the simulation, global averages and error-bars are calculated for sampling errors. When completed, timing and errors are printed.

The *xQSim* function call syntax is: *[error,data,output] = xQSim(input);*

The input data includes parameters and methods, but with an open architecture for extensibility. Most functions are modular and replaceable. This is as easy as defining a new function handle to replace the default value. The 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.

To explain this in full detail,

- Simulation parameters are stored in the ***input*** cell array.
- 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 ***xQSim(input).***
- The errors and integration time are returned in the ***error*** vector
- Averages are recorded sequentially in the ***data*** cell array.
- Parameters including defaults are returned in the ***output*** 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, xQSim can generates graphs with a companion graphics program, xGraph.

8.3 Parameters and functions

The xQSim input objects include parameters and functions. Many xQSim functions are modular and replaceable, through defining a new function handle to replace the default value.

There are standard conventions used throughout:

- All arguments in square brackets are optional, but may be needed only in specific cases.

- The last argument, *p,* is the parameter structure.

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

Label	Arguments	Output dimensions
a=qgen	*(a,p):* $a = (2M \times S_1)$, *p=[struct]*	$a = (2M \times S_1)$
permute=xqpermutation	~	Arbitrary: $1 \times M$
o=observe{n}	*(a,p):* $a = (2M \times S_1)$, *p=[struct]*	Arbitrary: $m_1 \times m_2 \ldots \times m_n$
c=compare{n}	(p), *p=[struct]*	Match observe: $m_1 \times m_2 \ldots \times m_n$

Notes

- ***qgen* has a default, namely *xqgen*, which can be user modified**
- **qgen 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**
- **Obtaining multiple lines requires modifying the output data file before xGraph**
- **The** m_e **dimension is used for error bars. Here** $m_e = 3$ **stores the sampling error bars**

8.4 Customization guide

To define your own stochastic generation function, include in the 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 (say) data graph 6, include in the 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 for the same graph, include in the 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
```

8.5 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 below.

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 array inputs are completed with the last default or input value.**
- **Default parameters are checked by inspecting the outputs, or setting verbose = 2.**

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

8.6 Parameter table

Simulation parameters are stored in a parameter structure which is passed to the *xsim* program. Inputs have default values, which are user-modifiable through the *xpreferences* function. Defaults can be checked by including the input *verbose* = 2. All the inputs are part of a structure passed to xQSim. 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.

Label	Type	Default	Description
deco	string	”	Decoherence factors used for optimization
bits	integer	None	Number of bits used for data extraction
CO	integer	None	Correlation order
compare	function handles	[]	Optional comparison functions
counts	integer	0	Total experimental counts for χ^2 tests
cutoff	real	-1.e100	Lower cutoff for graphs and χ^2
cyc	integer	1	Number of cycles of the network
ensembles	vector	[1,1,1]	Size of vector, serial, parallel ensemble
eps	vector	0	An n-dimensional decoherence factor
M	integer	1	Total matrix size
matrix	matrix function	Identity	An $M \times M$ transfer matrix function
method	integer	1	Phase-space method: $m = 1, 2, 3$
mincount	integer	0	Minimum count for χ^2 tests
N	integer	1	Number of excited input modes
name	character	”	Name of simulation
O	vector	None	Vector of various correlation orders
observe	function handles	$\{@(a,p)\, a\}$	Observable functions
permute	integer vector	$[1, 2, 3, \ldots, M]$	Random permutation of outputs
pnames	array of characters	{’+P ’,’W ’,’Q ’}	Names of phase-space methods
qgen	function handle	$@xqgen$	Stochastic generation code
r	vector or function	0	An n-dimensional squeezing vector
re	vector	1	An n-dimensional recycling factor
t	vector	1	An n-dimensional transmission factor
Tname	unitary name	matrix(0)	Name of transmission matrix
verbose	integer	0	Flag for printing verbosity level: 0,1,2

8.7 Parameter reference

The following parameters can be applied to each member of the simulation sequence. For single-input, single-pass GBS experiments, they are only specified once.

8.7.1 bits

Default: None

Integer used to define the bit-type encoding of experimental binary data files. Only used for integrated data extraction codes.

- p.bits

8.7.2 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,..}

8.7.3 CO

Default: None

Used to define the correlation order for calculating click correlation moments or marginal probabilities of output distribution.

- $p.CO$

8.7.4 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.

- $p.counts > 0$

8.7.5 cutoff

Default: -1E-100

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

- $p.cutoff$

8.7.6 cyc

Default: 1

Number of cycles of the network simulated.

- p.cyc >0

8.7.7 deco

Default: ' '

String used to output deocherence factors *eps* and t during optimization. Only needs to be used if Matlabs *fminsearch* is performing optimization.

- p.deco

8.7.8 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]

8.7.9 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

8.7.10 M

Default: 1

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

- p.M >0

8.7.11 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

8.7.12 method

Default: 1

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

- p.method > 0

8.7.13 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.

- $p.mincount > 0$

8.7.14 N

Default: 1

Number of input modes excited, less than or equal M

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

8.7.15 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'

8.7.16 O

Default: None

Used to define a vector of correlation orders to simulate mean click or photon number correlations over adjacent channels.

- $p.O$

8.7.17 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,..}

8.7.18 permute

Default: $[1, 2, 3, \ldots, M]$

Random permutation vector of size $1 \times M$ obtained either manually or via a data extraction output which is applied to rows of the transmission matrix for phase-space simulations.

- p.permute = xqpermutation

8.7.19 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'}

8.7.20 qgen

Default: @qgen

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

- p.qgen = @yourgenerator

8.7.21 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

8.7.22 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.

- $p.re \leq 1$

8.7.23 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.

- $p.t \leq 1$

8.7.24 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'

8.7.25 verbose

Default: 0

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

- p.verbose >= 0

9 xGraph reference

The graphics function provided is a general purpose multidimensional batch graphics code, xGraph, which can be called when xQSim 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 xQSim at a later time.

The program is described elsewhere, but the documentation is replicated here for users who wish to use it. The *xGraph* function call syntax is:

- ***xgraph (data [,input])***

This takes simulation *data* and *input* cell arrays with graphics parameters, then plots graphs. The *data* should have as many cells as there are *input* cells, for sequences. The data can include comparisons, either analytic or experimental, and error-bars for both the simulations and the comparisons, since either may have sampling or other errors.

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* parameters in the file are 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.

9.1 Parameter and data structures

This is a batch graphics function, intended to process quantities of graphics data in sequence, 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. For a sequence of one member, the enclosing cell array can be omitted.

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 *x*QSim. 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* is a cell array of parameter structures, which can be collapsed to one structure
2. The input *data* is a cell array of *datasets,* which can be collapsed to a single dataset
3. Each *dataset* is a cell array of multidimensional *graphs*, 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.

9.1.1 Comparisons

For every type of observation, 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.

9.2 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 an extended graphics data structure. Graphics parameters have default values which are user-modifiable by editing the *xgpreferences* function.

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
axes{n}	{0,..}	Points plotted for each axis
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
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
olabels{n}	*'a_1'*	Observable labels
pdimension{n}	3	Maximum plot dimensions
saveeps	0	Switch, set to 1 to save eps files
savefig	0	Switch, set to 1 to save figure files
scale{n}	1	Scaling: Counts/ probability density
transverse{n}	0	Number of transverse plots
xfunctions{n}	*{@(t,~) t,@(x,~) x,..}*	Axis transformations
verbose	0	0 for brief, 1 for informative, 2 for full output
xlabels	*{'t' ,'x' ,'y' ,'z'...}*	Global axis labels
octave	0	0 for Matlab, 1 for octave environment

- There are up to 6 types of input data, with 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 data has up to two error bars (I and II), and optional comparisons 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*,* the 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 have type I error bars while total simulation errors have type II errors with parallel lines, in order to distinguish them.

A detailed description of each parameter is listed in Sec (9.8).

9.3 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_1/100)','cosine(m_1/100)'};
data = {sin([1:100*pi]/100),cos([1:100*pi]/100)};
xgraph(data,p);
```

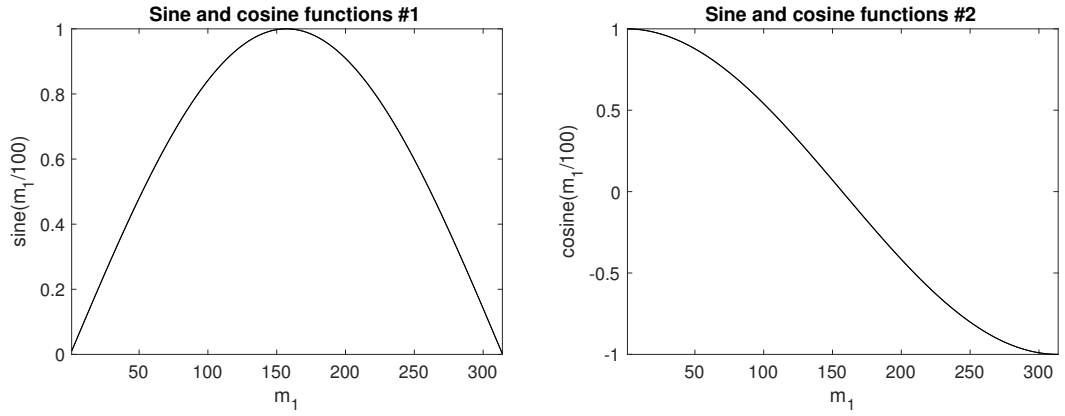

Figure 8: Example: xGraph output of two plots

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.

9.4 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.

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.

9.5 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.

9.6 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.

9.6.1 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.

9.6.2 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-bars, but they are still recorded in the data files.

9.7 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.

9.7.1 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:

$$\chi^2 = \sum_{j=1}^{m} \frac{(N_j - E_j)^2}{E_j}. \tag{59}$$

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.

9.7.2 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,

$$\chi^2 = \sum_{j=1}^{m} \frac{(p_j - e_j)^2}{\sigma_j^2 + \sigma_{e,j}^2}. \tag{60}$$

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 $p_j, e_j > p.cutoffs\{n\}$. This only has a χ^2 distribution if points are independent.

9.7.3 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

$$G^2 = 2 \sum_{j=1}^{m} N_j \ln\left(N_j / E_j\right). \tag{61}$$

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.

9.7.4 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:

$$p.parametric\{n\} = [n1, p2] \tag{62}$$

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.

9.8 Parameter reference

9.8.1 *axes{n}*

Default: *{0,0,0,..}*

Gives the axis points plotted for the n-th plotted function, in each dimension. Each entry value is a vector range for a particular plot and dimension. Thus, *p = 5* gives the fifth point only, and a vector input *p = 1:4:41* plots every fourth point. Single points generate graphics projections, allowing the other dimensions to be plotted. Zero or negative values are shorthand. For example, *p = -1* generates a default point at the midpoint, *p = -2* the endpoint, and *p = 0* is the default value that gives the vector for the every axis point. For each graph type, i.e. *n=1,..graphs* the axes can be individually specified in each dimension, *d=1,..dimension*s. If more than three axes are specified to be vectors, only the first three are used, and others are set to default values in the plots.

Example: *p.axes{4} = {1:2:10,0,0,-1}*

9.8.2 *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*

9.8.3 *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*

9.8.4 *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*

9.8.5 *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*

9.8.6 *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*

9.8.7 *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 *dimension*s. 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} = 'x^2'*

9.8.8 *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*

9.8.9 *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]*

9.8.10 *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'*

9.8.11 *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*

9.8.12 *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*

9.8.13 *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 *dimension*s. 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',}*

9.8.14 *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)}*

9.8.15 *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] ...,}*

9.8.16 *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',}*

9.8.17 *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*

9.8.18 *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*

9.8.19 *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'*

9.8.20 *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'*

9.8.21 *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*

9.8.22 *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*

9.8.23 *saveeps*

Default: 0

If set to 1, all plots are saved to the current folder as .eps files, numbered consecutively. It is best to use the *close all* command first to remove unwanted displayed xFIGURES, before running *xGraph* with this option.

Example: *p.saveeps =1*

9.8.24 *savefig*

Default: 0

If set to 1, all plots are saved to the current folder as .fig files, numbered consecutively. It is best to use the *close all* command first to remove unwanted displayed xFIGURES, before running *xGraph* with this option.

Example: *p.savefig =1*

9.8.25 *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*

9.8.26 *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*

9.8.27 *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 *dimension*s. 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'}*

9.9 User function reference

It is possible to simply run *xGraph* as is, without much intervention. However, there are customization options, including user defined functions. These are as follows:

9.9.1 *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.

9.9.2 *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.

References

[1] Scott Aaronson. A linear-optical proof that the permanent is #p-hard. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 467(2136):3393–3405, 2011.

[2] Scott Aaronson and Alex Arkhipov. The Computational Complexity of Linear Optics. *Theory of Computing*, 9:143–252, 2013.

[3] Scott Aaronson and Alex Arkhipov. Bosonsampling is far from uniform. *Quantum Info. Comput.*, 14:1383–1423, 2014.

[4] Craig S. Hamilton, Regina Kruse, Linda Sansoni, Sonja Barkhofen, Christine Silberhorn, and Igor Jex. Gaussian boson sampling. *Phys. Rev. Lett.*, 119:170501, Oct 2017.

[5] Nicolás Quesada, Juan Miguel Arrazola, and Nathan Killoran. Gaussian boson sampling using threshold detectors. *Physical Review A*, 98(6):062322, 2018.

[6] Regina Kruse, Craig S Hamilton, Linda Sansoni, Sonja Barkhofen, Christine Silberhorn, and Igor Jex. Detailed study of gaussian boson sampling. *Physical Review A*, 100(3):032326, 2019.

[7] Keith R. Motes, J. P. Olson, E. J. Rabeaux, J. P. Dowling, S. J. Olson, and P. P. Rohde. Linear optical quantum metrology with single photons: Exploiting spontaneously generated entanglement to beat the shot-noise limit. *Phys. Rev. Lett.*, 114:170802, 2015.

[8] Zu-En Su, Yuan Li, Peter P. Rohde, He-Liang Huang, Xi-Lin Wang, Li Li, Nai-Le Liu, Jonathan P. Dowling, Chao-Yang Lu, and Jian-Wei Pan. Multiphoton interference in quantum fourier transform circuits and applications to quantum metrology. *Phys. Rev. Lett.*, 119:080502, Aug 2017.

[9] M. D. Reid and D. F. Walls. Violations of classical inequalities in quantum optics. *Phys. Rev. A*, 34:1260–1276, Aug 1986.

[10] Roy J. Glauber. Coherent and Incoherent States of the Radiation Field. *Phys. Rev.*, 131:2766–2788, 1963.

[11] E. C. G. Sudarshan. Equivalence of Semiclassical and Quantum Mechanical Descriptions of Statistical Light Beams. *Phys. Rev. Lett.*, 10:277–279, 1963.

[12] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[13] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, et al. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

[14] Han-Sen Zhong, Yu-Hao Deng, Jian Qin, Hui Wang, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Dian Wu, Si-Qiu Gong, Hao Su, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Jelmer J. Renema, Chao-Yang Lu, and Jian-Wei Pan. Phase-programmable gaussian boson sampling using stimulated squeezed light. *Phys. Rev. Lett.*, 127:180502, Oct 2021.

[15] Lars S. Madsen, Fabian Laudenbach, Mohsen Falamarzi. Askarani, Fabien Rortais, Trevor Vincent, Jacob F. F. Bulmer, Filippo M. Miatto, Leonhard Neuhaus, Lukas G. Helt, Matthew J. Collins, Adriana E. Lita, Thomas Gerrits, Sae Woo Nam, Varun D. Vaidya, Matteo Menotti, Ish Dhand, Zachary Vernon, Nicolás Quesada, and Jonathan Lavoie. Quantum computational advantage with a programmable photonic processor. *Nature*, 606(7912):75–81, June 2022.

[16] Jacob F. F. Bulmer, Bryn A. Bell, Rachel S. Chadwick, Alex E. Jones, Diana Moise, Alessandro Rigazzi, Jan Thorbecke, Utz-Uwe Haus, Thomas Van Vaerenbergh, Raj B. Patel, Ian A. Walmsley, and Anthony Laing. The boundary for quantum advantage in Gaussian boson sampling. *Sci. Adv.*, 8(4):eabl9236, January 2022.

[17] Benjamin Villalonga, Murphy Yuezhen Niu, Li Li, Hartmut Neven, John C Platt, Vadim N Smelyanskiy, and Sergio Boixo. Efficient approximation of experimental gaussian boson sampling. *arXiv preprint arXiv:2109.11525*, 2021.

[18] Changhun Oh, Liang Jiang, and Bill Fefferman. Spoofing cross entropy measure in boson sampling, October 2022.

[19] A P Lund, A Laing, S Rahimi-Keshari, T Rudolph, J L O'Brien, and T C Ralph. Boson sampling from a gaussian state. *Phys. Rev. Lett.*, 113(10):100502, 2014.

[20] P D Drummond and C W Gardiner. Generalised p-representations in quantum optics. *Journal of Physics A: Mathematical and General*, 13(7):2353–2368, jul 1980.

[21] E. Wigner. On the Quantum Correction For Thermodynamic Equilibrium. *Phys. Rev.*, 40:749–759, 1932.

[22] K. Husimi. Some formal properties of the density matrix. *Proc. Phys. Math. Soc. Jpn.*, 22:264–314, 1940.

[23] Bogdan Opanchuk. squeezed-sim, 2021.

[24] Peter D. Drummond, Bogdan Opanchuk, A. Dellios, and M. D. Reid. Simulating complex networks in phase space: Gaussian boson sampling. *Phys. Rev. A*, 105:012427, Jan 2022.

[25] Peter D Drummond and Bogdan Opanchuk. Initial states for quantum field simulations in phase space. *Physical Review Research*, 2(3):033304, 2020.

[26] A. Dellios, Peter D. Drummond, Bogdan Opanchuk, Run Yan Teh, and Margaret D. Reid. Simulating macroscopic quantum correlations in networks. 2021.

[27] Alexander S Dellios, Margaret D Reid, Bogdan Opanchuk, and Peter D Drummond. Validation tests for gbs quantum computers using grouped count probabilities. *arXiv preprint arXiv:2211.03480*, 2022.

[28] Paulina Marian. Higher-order squeezing and photon statistics for squeezed thermal states. *Physical Review A*, 45(3):2044, 1992.

[29] Raw data from 100-mode experiment. `https://quantum.ustc.edu.cn/web/en/node/915`, 2020.

[30] Raw data from 144-mode experiment. `https://quantum.ustc.edu.cn/web/en/node/951`, 2021.