Simulating Linear-optical Quantum Computers

Ana Filipa Gonçalves de Carvalho

UMinho | 2021

University of Minho
School of Engineering

Ana Filipa Gonçalves de Carvalho

Simulating Linear-optical Quantum Computers

December 2021

University of Minho
School of Engineering

Ana Filipa Gonçalves de Carvalho

Simulating Linear-optical Quantum Computers

Master dissertation
Integrated Master's in Engineering Physics
Physics of Information

Dissertation supervised by
Michael Scott Belsley
Ernesto Fagundes Galvão

December 2021

ACKNOWLEDGEMENTS

The dissertation would not be possible without the contribution of several people to whom I wish to express my gratitude.

First, I would like to thank my supervisor Ernesto Galvão for being available to answer my questions and for all the patience over these months of work. Also to professor Michael Belsley who supervised the work, your perspective from a theoretical physicist contributed both for the theory and to complete the computational part to other readers.

Moreover, to all the teachers along the course that were always available to help. In that note, a special thank you to my cousin Jorge who helped me more times than I can count in my questions particularly in my graduation years. In no particular order of importance, to my oldest trio, my colleague friends and all those who were present in any way they could and at all times that contributed in keeping my sanity throughout these five years and many more.

Finally, to all my family but specially to my parents who always supported me and motivated to continue working as well as my sister who in her peculiar way can trigger me into being and doing better.

STATEMENT OF INTEGRITY

ABSTRACT

Research in quantum computation has sharply increased in recent years, due to the promised computational advantage with respect to classical computers. Nowadays there are several proposals to encode quantum information. This dissertation discusses a particular type of quantum computer, based on linear optics.

To support this approach, in this work we will investigate in detail the computational cost and challenges of simulating Boson Sampling and Gaussian Boson Sampling models to try to show quantum supremacy. The complexity of classical simulation is mainly due to the calculation of a very particular function for each case but several aspects can be considered to help minimize these costs; here we will discuss some of these aspects along with the most efficient proposals in the literature. First we will review some of the basic theory about linear and non-linear optics, a mature research topic. Thereafter, this theory will be applied to the two sampling-based quantum computational models we will study, followed by verification of computational complexity of them with some numerical experiments with our simulator. We use Python code, as well as an implementation using the Strawberry Fields library made available by Canadian company Xanadu, and which allows to use that code to run in an actual device.

Besides demonstrations of quantum computational advantage, we also discuss useful applications of linear-optical quantum computation. These applications are diverse, ranging from graph theory to quantum chemistry, and use different encodings which we will discuss.

KEYWORDS Linear optics, computational complexity, quantum information, simulation, quantum computation

RESUMO

Investigação em computação quântica aumentou consideravelmente nos últimos anos devido à promessa de vantagem computacional relativamente a computadores clássicos. Atualmente há várias propostas para codificar informação quântica. Esta dissertação foca-se num tipo de computador quântico particular baseado em ótica linear.

Para suportar esta abordagem, neste trabalho será investigado em detalhe o custo computacional de simular os modelos de Amostragem Bosónica e Amostragem Bosónica Gaussiana e mostrar vantagem computacional quântica. A complexidade da simulação clássica deve-se principalmente ao cálculo de uma função muito particular para cada caso mas vários aspetos podem ser tomados em conta para ajudar a minimizar estes custos; serão discutidos alguns juntamente com as propostas mais eficientes da literatura. Primeiramente faremos uma revisão teórica sobre ótica linear e não linear, um tópico bastante desenvolvido. Posteriormente, será aplicada esta teoria aos dois modelos baseados em amostragem para computação, seguido da verificação da complexidade computacional dos mesmos juntamente com alguns testes numéricos com o nosso simulador. Nós usaremos código escrito em Python bem como uma implementação pela biblioteca Strawberry Fields disponibilizada pela empresa Xanadu que permite usar esse código para correr num dispositivo real.

Além de demonstrar vantagem computacional quântica, também discutiremos aplicações úteis para computação quântica de ótica linear. Estas aplicações são diversas, desde teoria de grafos a química quântica, que usam diferentes codificações que discutiremos.

PALAVRAS-CHAVE Ótica linear, complexidade computacional, informação quântica, simulação, computação quântica

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

1

INTRODUCTION

1.1 the context: quantum computation

Quantum computation is a topic that has been getting a lot of attention in the past decades and a few quantum computers are already available. This technology has applications in cryptography, as well as the efficient solution of otherwise intractable computational problems.become tractable with quantum programming and computation. The interest arose mostly with the Deutsch-Jozsa algorithm introduced in 1992, for which classical solution take at least $2^{n-1}+1$ iterations to know if the function is balanced or constant, while a quantum solution takes impressively only one Nielsen and Chuang (2010). The technology promises extreme advantage in runtime, as well as in memory use.

The applications for quantum computation are several: from database search, solving linear equations to quantum simulation (for chemistry, nanotechnology...) and quantum cryptography.

The basic unit in classical computation is the bit which can be either 0 or 1 but, in quantum computation, quantum bits (qubits) are used which can be 0, 1 or a linear combination of them. The physical implementation requires quantum systems with two (or more) distinguishable states. Examples of implementations are superconducting circuits, using Josephson junctions and nuclear magnetic resonance quantum computer (NMRQC) using nuclear spins.

All quantum computers must be able to do the next three fundamental processes:

1. Prepare input states: the computer must be capable of preparing the states - qubits - in a relatively efficient way;
2. Evolution: the states pass through a series of operations and the computer must be able implement this arbitrary temporal evolution - condition for universality;
3. Measurement: the only way to access the information after the process (temporal evolution) is by measuring the quantum states created, for read-out or possibly as an intermediate step in the computation.

More recently, in the article by S. Aaronson and A. Arkhipov Aaronson and Arkhipov (2013), they showed a quantum advantage for solving a problem they defined, which became known as the Boson Sampling problem. Later other variations were proposed, such as Gaussian Boson Sampling. In both models, special states of light are created over a certain number of optical modes, so the encoding is not usually made in terms of qubits. However, using only linear optical elements it is possible to create an universal quantum computer. But what is the interest in using linear optics instead of the already existing techniques? Here are a few reasons:

- Coherence and flexibility in designing error-correcting codes;
- Does not require extreme cryogenic temperatures;
- Scalability since the physical requirements are based on mature fabrication techniques;
- Photons are excellent information carriers.

Importantly, qubit computations can be incorporated into the optical quantum field picture, so when taking the approaches in this thesis (continuous variable and Fock states) there is no loss in computational power.

1.2 state of the art

Encoding quantum information in optical systems needs a strong theoretical background and books by Barnett and Radmore (1997) and Mandel and Wolf (1995) have the complete description of the theory and applications of quantum optics. Focused on mathematical construction for the encoding of CV operations, Adesso et al. (2014) gives us the basic notions to understand and introduce theory for the GBS.

The models reviewed in this dissertation were introduced by Aaronson and Arkhipov (2013) that discussed the complexity and advantage of Boson Sampling and by Hamilton et al. (2017) that discussed a different approach - Gaussian Boson Sampling - which offers an improvement in physical implementation according to current technologies. After both these two models were proposed, several teams presented projects implementing them. Current proposals that demonstrated quantum advantage are by Wang et al. (2019) and Zhong et al. (2021) for BS and GBS respectively; they executed tasks that are intractable to be calculated by a classical algorithm but they lack interferometer reconfiguration, that is, flexibility of programming the device for different applications.

The motivation to study them relies on demonstrating quantum supremacy so if a classical algorithm is discovered that efficiently solves these problems, they should no longer be useful. In some particular functioning regimes, such as when only distinguishable photons are used, there are algorithms that run in a polynomial time and some of them are by Glynn (2013), Aaronson and Hance (2014) and Valiant (1979) which makes it harder for the samplers since it implies a requirement for better hardware, to ensure the experiment is run in a regime that is not efficiently classically simulable. As for algorithms to simulate a BS device, Clifford and Clifford (2017) is the best algorithm at the time that does weak simulation without requiring strong simulation first and as for GBS devices, Björklund et al. (2019) and Quesada and Arrazola (2020) show algorithms for faster running time but which are still exponential-time in a classical computer.

With a nanophotonic chip capable of GBS or BS, we need a programming language to implement the computation. The Xanadu company and the Strawberry Fields library more particularly introduced by Killoran et al. (2019b) are dedicated to simulating these models and, more recently, it is possible to access quantum hardware to obtain real experimental samples from cloud-based devices. Quantum algorithms depend on quantum system initialization, quantum dynamics, followed by measurement and classical post-processing and each of these needs to be efficient. To program the hardware, a fundamental problem is that of finding specific decompositions of interferometer designs into small building blocks, in terms of beam-splitters and phase shifters, or their integrated chip counterparts (directional waveguide couplers). For interferometer decomposition, the best proposal at the moment runs in polynomial time is by Clements et al. (2016). Bromley et al. (2020) also discusses other decomposition methods for applications where Brádler et al. (2018) study it in detail. The programmable chip mentioned was presented by Arrazola et al. (2021); although it is a small device and easy to classically simulate, it can be scaled to a bigger size since it has all requirements of state preparation, evolution and measurement with efficient implementation and lower associated experimental error.

Applications are being studied for the two boson sampling models. Some are suggested by Bromley et al. (2020) with graph algorithms and many others, even though a few of them have efficient classical algorithms. The most promising applications focus on encoding qubits into optical modes, using specific error-correction schemes initially described by Knill et al. (2001) and Gottesman et al. (2001).

1.3 objectives and outline

In this thesis we will take quantum linear optics and study the quantum advantage of two models proposed using this strategy - Boson Sampling and Gaussian Boson Sampling. The goal will be to study state preparation, examine how to treat the evolution of these systems and how to measure in order to calculate this computational behaviour to show advantage using a quantum model.

We will show how to simulate a quantum system for both models mentioned which includes the mathematical part, the computational processing and generation of samples in a classical computer. To illustrate the quantum advantage, we proceed by showing the complexity of solving the corresponding problem with a quantum device. A major concern about quantum computers is experimental challenges and for that reason, it is relevant to consider not only computational complexity but implementation of quantum device with the characteristics the model requires.

When experiments demonstrating quantum computational advantage are performed, using these probabilistic models can be tricky because we have no way to verify the samples from the device since we are not capable of simulating the quantum process on a classical computer. We intend to demonstrate a few validation methods to indicate a way to gain confidence in the device and observe if it outputs as predicted.

Last but not least, having devices capable of quantum supremacy, it is important to be able to apply them to real life applications. So, to conclude the thesis, we review some of the applications that have been proposed in the literature for these devices

This dissertation is structured as follows:

Chapter 2 - mainly divided in three parts: quantum states of light with respective preparation, operations/evolution of those states and measurement. It is easy to see a relation from this theory to application in quantum computing fundamental processes. To do so we begin by briefly explaining the quantum harmonic oscillator which has the central mathematical foundation to analyse the electromagnetic field of light. The evolution is given by particular physical elements that will be translated into a matrix formulation which facilitates calculating their action on quantum sates. Finally, we consider measurements and detection methods in different bases.

Chapter 3 focuses on the Boson Sampling model; having the theory from the previous chapter allows us to adapt it according to the specifications of the model, implement classical programs to simulate a Boson Sampling device, implement a quantum program (run for simulation and in actual devices) with Strawberry Fields library, calculate the complexity of classical implementations, test validation of samples and finally make a few numerical experiments.

In chapter 4 we review the second quantum computational model of this dissertation, known as Gaussian Boson Sampling. Here we will be focusing on the adjustments to previous programs to run this model; the code itself will be similar, adapting the necessary parts and we also present the quantum algorithm. We have several chips to run the GBS model, there we present some and discuss programmability and implementations. In this chapter we did not discuss validation because the theory behind it is the same as for Boson Sampling; similarly, the simulation functions are adapted from the previous chapter.

In chapter 5 we focus on applications of the GBS model. As many applications are in graph theory, we review some of the basic graph-theoretic notions. In section 5.2 we describe three different applications using graphs. The next section centers around the physical implementation and experiments for these problems. This is, according to the nanophotonic chips available, check if the requirements to solve these problems can be implemented. To finish this chapter, we discuss other applications and future perspectives; other problems are mentioned that can be solved with qumodes and how to use these linear optical elements talked throughout the thesis to encode the usual qubits. We conclude this dissertation in chapter 6 with some final remarks.

2

LINEAR OPTICS

Light consists of particles called photons which are elementary particles and as such they exhibit a wave-particle duality. The classical description, in terms of waves, is given by the electromagnetic field. These particles have zero mass and travel in vacuum at speed of light c_0. The orientation of the electric field characterizes the light's polarization (a property of these waves) that takes various forms like linear, circular and elliptical. Throughout the study of optical phenomena, this relation between the induced polarization of matter and electric field was thought to be linear ($P \propto E$) but only until 1960 where the invention of laser showed a behavior different of observed so far that required higher intensities. From that point forward, it is known that light has nonlinear properties depending on the medium it passes through. Explicitly, this means, for instance, that:

- Medium properties are not fixed (refractive index, absorption and others are affected by light);
- Superposition principle for E is no longer valid and there is the need to consider interaction between photons.

However, these phenomena is not easily detected because linear terms usually are stronger than non linear ones. In order to detect this it is required to have special conditions and/or higher intensity of light (there are cases where a few or even one photon can provoke a nonlinear effect) which not only can be very difficult to implement but, as said, hard to detect so in this work, we will be considering linear-optical phenomena.

From a mathematical point of view, a classical monochromatic mode of the electromagnetic field and a classical harmonic oscillator behave identically. Similarly, a quantum monochromatic electromagnetic mode and a one-dimensional quantum-mechanical harmonic oscillator have identical behavior. With this, before we start, let us look for a (time-independent) solution of the quantum harmonic oscillator where we want to find the stationary states of energy eigenvector Ohanian (1990) such that:

$$\left(\frac{p_{op}^2}{2m} + \frac{1}{2}m\omega^2 x_{op}^2\right)|E_n\rangle = E_n |E_n\rangle \tag{1}$$

For the calculation it is convenient to define the dimensionless operators a and $a^\dagger$ (creation and annihilation operators, respectively) with $x_0 = \sqrt{\hbar/m\omega}$ such that:

$$\hat{a} = \frac{1}{\sqrt{2}}\left(\frac{1}{x_0}x_{op} + \frac{ix_0}{\hbar}p_{op}\right) \tag{2}$$

$$\hat{a}^\dagger = \frac{1}{\sqrt{2}}\left(\frac{1}{x_0}x_{op} - \frac{ix_0}{\hbar}p_{op}\right) \tag{3}$$

and we can easily obtain the commutation relation $[\hat{a}, \hat{a}^\dagger] = \hat{a}\hat{a}^\dagger - \hat{a}^\dagger\hat{a} = 1$ from the canonical commutation relation of x_{op} and p_{op}. Rewriting the position and momentum operator in terms of $\hat{a}$ and $\hat{a}^\dagger$:

$$x_{op} = \frac{x_0}{\sqrt{2}}(\hat{a} + \hat{a}^\dagger) \qquad p_{op} = \frac{\hbar}{i\sqrt{2}x_0}(\hat{a} - \hat{a}^\dagger)$$

Given by equation 1, it is then the same as:

$$\hat{H} = \hbar\omega\left(\hat{a}^\dagger\hat{a} + \frac{1}{2}\right) \tag{4}$$

Having this formulation, one can prove that $\hat{a}$ and $\hat{a}^\dagger$ are operators where $\hat{a}^\dagger |E_n\rangle$ is eigenvector of $\hat{H}$ with eigenvalue $E_n + \hbar\omega$ and $\hat{a}|E_n\rangle$ is eigenvector of $\hat{H}$ with eigenvalue $E_n - \hbar\omega$. This should clarify why they are called creation and annihilation operators: when acting on an energy eigenvector, they produce another with increased or decreased eigenvalue.

Following the previous point, to know the eigenvalue to each eigenvector, we only need to know the eigenvalue of the ground state and apply $\hat{a}^\dagger$ as many times as necessary. Since the expectation value of the energy in an eigenstate equals the eigenvalue we have:

$$\langle E_0| \hat{H} |E_0\rangle = 0 + \frac{\hbar\omega}{2} \quad \rightarrow \quad E_0 = \frac{1}{2}\hbar\omega$$

Finally, we can conclude that:

$$E_n = \hbar\omega\left(\hat{n} + \frac{1}{2}\right) \tag{5}$$

With this last result we can find out the explicit relation between $|E_n\rangle$ and $(a^\dagger)^n |E_0\rangle$. We know that they are proportional so let us use c to denote the constant relating them. We also have that $\langle E_i|E_j\rangle = \delta_{i,j}$ so, doing the inner product and normalizing:

$$|E_{n+1}\rangle = c_{n+1}\hat{a}^\dagger |E_n\rangle \quad \rightarrow \quad c_{n+1} = \frac{1}{\sqrt{n+1}}$$

$$|E_{n-1}\rangle = c_{n-1}\hat{a} |E_n\rangle \quad \rightarrow \quad c_{n-1} = \frac{1}{\sqrt{n}}$$

Applying this result successively to $n = 0, 1, 2...$, the normalized eigenvector $|E_n\rangle$ is reduced to:

$$|E_n\rangle = \frac{(\hat{a}^\dagger)^n}{\sqrt{n!}} |E_0\rangle \tag{6}$$

2.1 phase space representation

These states can be represented in phase space, as we will soon see. Even though a quantum particle does not have simultaneously well-defined position and momentum, phase space representations help in the visualization of quantum phenomena, and in the discussion of the quantum/classical correspondence.

When deriving results from the Schrödinger picture, the states represented by a vector or density matrix evolve in time and the operators are constant but when using the phase space, the time evolution is given by transformations of a constant state and in the characterization associated to it. Specifically, an optical phase space is a phase space in which all quantum states of an optical system are described. For a classical state of an optical system, each point in the optical phase space is described with definite values of x and p. In the case of the harmonic oscillator, when position x is maximum, velocity p is minimum and over time it is translated to a sinusoidal function and in phase space to an ellipse. For a quantum state, the quadratures x and p are associated with a probability distribution for the uncertainty in measurements and since the phase space can represent them, this function also describes the state other than just the mean or a point value associated.

Contrary to classical systems, quantum systems have an uncertainty associated to the states that can not violate Heisenberg's uncertainty principle. The function for the probabilities in this case is a quasi-probability distribution according to the wave function or density matrix of the state that is called Wigner function or also a Characteristic Function that both fully characterize a quantum state in phase space but the former will be the most used. So drawing a parallel between the classical and quantum representation of the dynamics of a harmonic oscillator, the former can be represented by a shape as in figure 1a where it is a simple circle and the latter is in figure 1b where states have a minimum uncertainty (width), bounded below by the uncertainly principle.

The Wigner function for arbitrary states given the density matrix ρ is $W(x, p)$:

$$W(x,p) = \frac{1}{2\pi\hbar} \int_{-\infty}^{+\infty} \left\langle x + \frac{y}{2} \middle| \rho \middle| x - \frac{y}{2} \right\rangle \exp(-ipy/\hbar) \mathrm{d}y$$

With properties of quantum mechanics, we can have the expectation value of an operator $\hat{A}$ via $\langle \hat{A} \rangle = Tr(\rho \hat{A})$ and in this case, using $W(x, p)$ we have:

$$\langle x \rangle = \int \int \mathrm{d}x \mathrm{d}p W(x,p) x$$

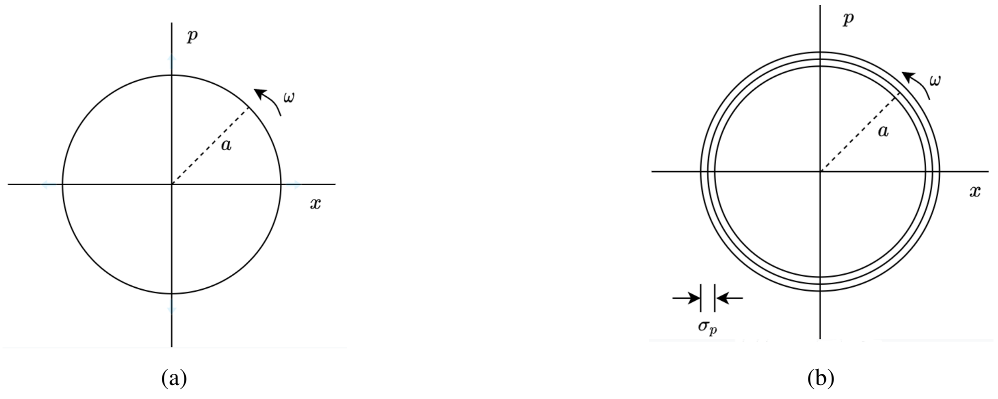


Figure 1: Representation of classical 1a and quantum 1b harmonic oscillator in phase space for arbitrary values of intensity a (proportional value to energy) and uncertainty $\sigma_p = \sigma_x$ in the case of a quantum HO.

and a similar expression for p. The probability distribution for each parameter is obtained via a single integral.

The Wigner function representation may help in identifying non-classical characteristics of a quantum state. The Wigner function is not a simple probability distribution because it only behaves like a probability: the sum or integral over the parameters is one, i.e. the function is normalized but this probabilities have a peculiar aspect, they are not restricted to the interval $[0, 1]$ also allowing negative probabilities. Note that this doesn't mean the probability to find the state in a given value of x for instance is below zero! Only the combination of the probability for value x and p might be unconventional, the probability over a single variable is in the ordinary range and we will see a brief example in the next section. This is, the marginal distributions for x, p and other quadratures (linear combinations of x and p) are all the correct quantum-mechanical distributions. The feature of negative quasi-probabilities are a quantum mechanical phenomenon, arising from fitting quantum mechanics into the classical phase-space picture.

2.2 fock states

Using the association to the quantum harmonic oscillator, denoting states $|E_n\rangle$ as simply $|n\rangle$, these are called number states of the harmonic oscillator or Fock states and are simultaneously eigenstates of $\hat{H}$ and $\hat{n} = \hat{a}^\dagger \hat{a}$ where $\hat{n}\,|n\rangle = n\,|n\rangle$ since $\hat{H} = \hbar\omega\left(\hat{a}^\dagger\hat{a} + \frac{1}{2}\right)$. They have a well defined number of particles (or quanta) and are characterized by a set of occupation numbers for all modes.

Using equation 6 we have the next equation where m is the number of modes and $\sum_i n_i = n$:

$$|n\rangle = |n_1, n_2, \dots, n_m\rangle = \prod_{i=1}^{m} \frac{(\hat{a}_i{}^\dagger)^{n_i}}{\sqrt{n_i!}}\,|0\rangle \tag{7}$$

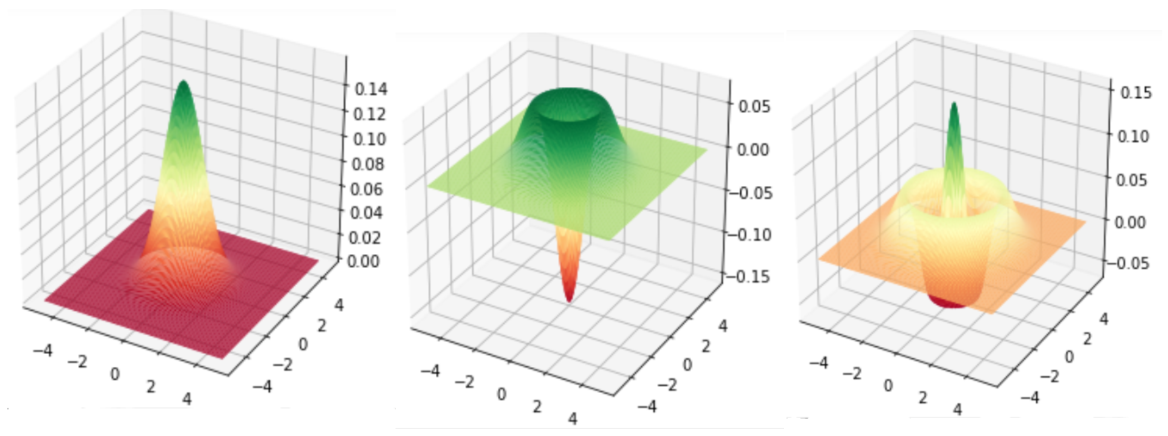
In the radiation field, these quanta or discrete excitations, are the photons so the creation and annihilation operators increase or decrease this number by one unit. The photon number has no upper bounds and the degeneracy of the eigenvalues is infinite. Fock states form a complete set which can serve as a basis for the representation of arbitrary states which will come in handy for further analysis.

The Wigner function for Fock states is found to be:

$$W_{|n\rangle}(x,p) = \frac{(-1)^n}{\pi} \exp\left(-(x^2+p^2)\right) L_n(2(x^2+p^2))$$

where $L_n(x)$ denotes the Laguerre polynomials Kenfack and Zyczkowski (2004). Here we only wish to show an example of the distinctive aspect of quasi-probability distribution in phase space. For better illustration, we used the Strawberry Fields library to represent selected states (see Appendix B.1). In sub-figure 2a we see the Wigner function representation of the vacuum state representing the initial state for any quantum optical mode. In figure 2 we can see different Wigner functions for Fock states and conclude two things mainly:

- The vacuum state in this term is similar to classical states in that it is always positive in the interval $[0, 1]$;
- Fock states allow to observe nonclassicality; for $x = p = 0$ the amplitude is maximum and it is somewhat intuitive since it is unlikely for the particle to be found with that condition. Also, the more excited the state is, the less well behaved is the function: according to Kenfack and Zyczkowski (2004), the number of zeros of Laguerre polynomials increases monotonically with n hence the larger the number n, the more W can be interpreted as a non-classical distribution.



(a) (b) (c)

Figure 2: Wigner function representations of several Fock states; colour is added only to aid visualization. The vacuum state in 2a is for simple comparison with behavior of Fock states where in figure 2b is drawn the state $|1\rangle$ and in figure 2c is the state $|2\rangle$.

2.3 gaussian states

The annihilation and creation operators $\hat{a}$ and $\hat{a}^\dagger$ are not Hermitian but observable quantities can be represented by Hermitian combinations of them.

An example was already used in last chapter - the number operator $\hat{n} = \hat{a}^\dagger\hat{a}$. Another example are the quadrature operators which are linear combinations of position $\hat{x}$ and momentum $\hat{p}$. Adopting natural units, they combine the previous operators in this way:

$$\hat{x} = \frac{(\hat{a}+\hat{a}^\dagger)}{\sqrt{2}} \quad ; \quad \hat{p} = \frac{(\hat{a}-\hat{a}^\dagger)}{i\sqrt{2}} \tag{8}$$

They satisfy the bosonic commutation relations $[\hat{x}_i, \hat{p}_j] = i\delta_{ij}$. We can group these in one vector $\hat{R} = (\hat{x}_1, \hat{p}_1, ..., \hat{x}_N, \hat{p}_N)^\intercal$ so the previous commutation relation becomes

$$[\hat{R}_k, \hat{R}_l] = i\Omega_{kl} \qquad \text{where } \Omega = \bigoplus_{k=1}^{N} \omega; \quad \omega = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{9}$$

The quadrature operators ($\hat{X} = \hat{x}/\sqrt{2}$ and $\hat{P} = \hat{p}/\sqrt{2}$) represent the real and imaginary parts of the complex amplitude of the electromagnetic field (the phasor $\alpha = |\alpha|\exp(i\theta)$) and the reason for their name is because they are a quarter cycle (90°) out of phase with each other, as we will see.

2.3.1 Coherent states

Suppose we have an eigenstate of annihilation operator $\hat{a}$ such that $\hat{a}\,|\alpha\rangle = \alpha\,|\alpha\rangle$. Since the set of Fock states form a basis for this Hilbert space, we can rewrite the previous equation as a combination of number states

$$|\alpha\rangle = \sum_{n=0}^{\infty} c_n\,|n\rangle$$

where $c_n \in \mathbb{C}$ is the coefficient for occupation number n. Applying $\hat{a}$ on this expression (with the aid of Eq. 6), we get:

$$\hat{a}\,|\alpha\rangle = \sum_{n=1}^{\infty} c_n\sqrt{n}\,|n-1\rangle = \alpha\sum_{n=0}^{\infty} c_n\,|n\rangle$$

Since the Fock states are orthogonal ($\langle n_i|n_j\rangle = \delta_{i,j}$):

$$c_n\sqrt{n} = \alpha\cdot c_{n-1} \Leftrightarrow c_n = \frac{\alpha}{\sqrt{n}}c_{n-1} \quad \rightarrow \quad c_n = \frac{\alpha^n}{\sqrt{n!}}c_0$$

Substituting the result c_n,

$$|\alpha\rangle = c_0 \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle$$

Similar to above, knowing that $|\alpha\rangle$ must be normalized ($\langle\alpha|\alpha\rangle = 1$), we can use the orthogonality of $|n\rangle$ and Taylor series expansion to determine c_0:

$$\langle\alpha|\alpha\rangle = 1 \Leftrightarrow |c_0|^2 \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} \frac{\alpha^{*n}\alpha^m}{\sqrt{n!}\sqrt{m!}} \langle n|m\rangle = |c_0|^2 \sum_{n=0}^{\infty} \frac{|\alpha|^{2n}}{n!} = |c_0|^2 \cdot e^{|\alpha|^2}$$

$$1 = |c_0|^2 \cdot \exp\left(|\alpha|^2\right) \quad \rightarrow \quad c_0 = \exp({}^{-|\alpha|^2}/_2)$$

To sum up, we write the state $|\alpha\rangle$ denominated coherent state as:

$$|\alpha\rangle = \exp\left(-|\alpha|^2/2\right) \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle \tag{10}$$

which is equivalent to

$$|\alpha\rangle = \exp\left(\alpha\hat{a}^\dagger - \alpha^*\hat{a}\right) |0\rangle = \hat{\mathcal{D}}(\alpha) |0\rangle \tag{11}$$

where $\mathcal{D}(\alpha)$ is the coherent state Glauber displacement operator. In analogy to the harmonic oscillator, the effect is similar to pulling the particle out of the equilibrium position by an initial displacement $\mathrm{Re}(\alpha)$ and initial velocity $\mathrm{Im}(\alpha)$ leaving it then oscillating.

Because it has to be unitary, this operator must satisfy the following

$$\mathcal{D}^\dagger(\alpha)\mathcal{D}(\alpha) = \mathcal{D}(\alpha)\mathcal{D}^\dagger(\alpha) = 1$$

The fact that, from Barnett and Radmore (1997), for all operators $\hat{A}$ we have $\exp(\hat{A})\exp(-\hat{A}) = 1$ and along with its definition, we see $\mathcal{D}(\alpha)$ is unitary:

$$\mathcal{D}^\dagger(\alpha) = \exp\left(-\alpha\hat{a}^\dagger + \alpha^*\hat{a}\right) = \mathcal{D}(-\alpha) = \mathcal{D}(\alpha)^{-1} \tag{12}$$

Now we can prove that this operator indeed displaces a state and prove that equation 11 is valid, this is, we obtain a state $|\alpha\rangle$ from applying $\mathcal{D}(\alpha)$ to the vacuum. It is convenient to have the same exponential relating $|\alpha\rangle$ to $|0\rangle$ so we use a detail from Mandel and Wolf (1995) - Given that not all operators commute, $\exp(\hat{A}+\hat{B})$ is not always $\exp\hat{A}\exp\hat{B}$; The correct equality given by the Baker–Campbell–Hausdorff formula where $\exp(\hat{A}+\hat{B})$ is translated into a formula with infinite terms. Nevertheless, if $[\hat{A},\hat{B}] = c$ for c some constant, this is, those operators commute with their commutator, i.e. $[\hat{A},[\hat{A},\hat{B}]] = [[\hat{A},\hat{B}],\hat{B}]$ then we only need the first three terms of the series:

$$\exp(\hat{A}+\hat{B}) = \exp\hat{A}\exp\hat{B}\exp(-[\hat{A},\hat{B}]/2)$$

Using $\hat{A} = \alpha\hat{a}^\dagger$ and $[\hat{A},\hat{B}] = |\alpha|^2$, $\hat{B}$ is then $-\alpha^*\hat{a}$. Then, we can rewrite Eq. 11 as:

$$\exp\left(\alpha\hat{a}^\dagger - \alpha^*\hat{a}\right)|0\rangle = \exp(-|\alpha|^2/2)\exp\left(\alpha\hat{a}^\dagger\right)\exp(-\alpha^*\hat{a})|0\rangle$$

Remembering again the Taylor series and $\exp(-\alpha^*\hat{a})|0\rangle = |0\rangle$ we can write $|\alpha\rangle$ as a sum over the vacuum state $|0\rangle$ where we use Eq.7, which leads to:

$$\exp(-|\alpha|^2/2)\exp\left(\alpha\hat{a}^\dagger\right)|0\rangle = \exp(-|\alpha|^2/2)\sum_{n=0}^{\infty}\frac{\alpha^n}{\sqrt{n!}}\frac{(\hat{a}^\dagger)^n}{\sqrt{n!}}|0\rangle = |\alpha\rangle$$

These states form a over-complete basis for the harmonic oscillator and, as a consequence, they are not mutually orthogonal, i.e. perfectly distinguishable. The overlap between them can be determined by their Wigner functions or with two states $|\alpha\rangle$ and $|\alpha'\rangle$ where $\langle\alpha|\alpha'\rangle = \exp\left[-(|\alpha|^2 + |\alpha'|^2 - 2\alpha'\alpha^*)/2\right]$.

Properties of the Displacement operator

The unitary transformation that represents the displacement operator acting on the vacuum generates a coherent state. Now let us first see the transformation for the creation and annihilation operators which can be verified with the following: consider the annihilation operator acting on a state $\hat{a}|\alpha\rangle$. Using equations 11 and 12, then

$$\hat{a}|\alpha\rangle = \hat{a}\mathcal{D}(\alpha)|0\rangle = \mathcal{D}(\alpha)\mathcal{D}(-\alpha)\hat{a}\mathcal{D}(\alpha)|0\rangle$$

The operators $\mathcal{D}(-\alpha)\hat{a}\mathcal{D}(\alpha)$ can be simplified using the Hadamard lemma Barnett and Radmore (1997):

$$\exp(\hat{A})\hat{B}\exp(-\hat{A}) = \hat{B} + [\hat{A},\hat{B}] + \frac{1}{2!}[\hat{A},[\hat{A},\hat{B}]] + \frac{1}{3!}[\hat{A},[\hat{A},[\hat{A},\hat{B}]]] + ... \tag{13}$$

Considering $\hat{B} = \hat{a}$ and $\hat{A} = -\alpha\hat{a}^\dagger + \alpha^*\hat{a}$, the mode evolution is:

$$\begin{aligned}\mathcal{D}(-\alpha)\hat{a}\mathcal{D}(\alpha) &= \hat{a} + [-\alpha\hat{a}^\dagger + \alpha^*\hat{a}, \hat{a}] + ...\\ &= \hat{a} + \alpha(\hat{a}\hat{a}^\dagger - \hat{a}^\dagger\hat{a}) + \alpha^*(\hat{a}\hat{a} - \hat{a}\hat{a})\\ &= \hat{a} + \alpha\end{aligned}$$

Substituting in the state considered above, we get:

$$\mathcal{D}(\alpha)\mathcal{D}(-\alpha)\hat{a}\mathcal{D}(\alpha)|0\rangle = \mathcal{D}(\alpha)(\hat{a}+\alpha)|0\rangle = \alpha\mathcal{D}(\alpha)|0\rangle = \alpha|\alpha\rangle$$

The same can be done for creation operator obtaining: $\mathcal{D}(-\alpha)\hat{a}^\dagger\mathcal{D}(\alpha) = \hat{a}^\dagger + \alpha^*$. This can be generalized to any function f of the operators $\hat{a}$ and $\hat{a}^\dagger$:

$$\mathcal{D}(-\alpha)f(\hat{a},\hat{a}^\dagger)\mathcal{D}(\alpha) = f(\hat{a}+\alpha, \hat{a}^\dagger + \alpha^*)$$

This confirms the analogy made earlier between QHO and the displacement operator for construction of coherent states: acting on a coherent state (the vacuum or any other), it creates another coherent state represented by the original coherence state parameter, added to α. This is a complex value so $\mathcal{D}(\alpha)$ dislocates the position by $\mathrm{Re}(\alpha)$ and velocity by $\mathrm{Im}(\alpha)$ so in phase space it will be possible to observe this for different values such as for values with same module of α but different angles as we will shortly see.

Before that, let us make two more steps into concluding analysis of coherent states. The first is by generalizing this formulation for N modes.

The definition of Displacement operator in eq. 11 can be generalized for coherent states for N different modes. In order to do so, we use the tensor product of $\hat{\mathcal{D}}(\alpha)$ for a N-mode of vacuum state $|0\rangle$. This general operator can have the following form:

$$\hat{\mathcal{D}}(\xi) = e^{i\hat{R}^{\intercal}\Omega\xi} \text{ such that } |\xi\rangle = \hat{\mathcal{D}}(\xi)\,|0\rangle \tag{14}$$

For one mode we know that $\xi = \sqrt{2}\begin{pmatrix}\mathrm{Re}(\alpha)\\ \mathrm{Im}(\alpha)\end{pmatrix}$, in which case it is easy to demonstrate that it corresponds to the displacement operator:

$$\begin{aligned}\hat{D}(\xi) &= \exp\left(i(\xi_2\hat{q}_k - \xi_1\hat{p}_k)\right)\\ &= \exp\left(i\sqrt{2}\,\mathrm{Im}(\alpha)\frac{\hat{a}_k + \hat{a}_k^\dagger}{\sqrt{2}} - \sqrt{2}\,\mathrm{Re}(\alpha)\frac{\hat{a}_k - \hat{a}_k^\dagger}{\sqrt{2}}\right)\\ &= \exp\left(\hat{a}_k[i\,\mathrm{Im}(\alpha) - \mathrm{Re}(\alpha)] - \hat{a}_k^\dagger[-\mathrm{Re}(\alpha) - i\,\mathrm{Im}(\alpha)]\right)\\ &= \exp\left(\hat{a}_k^\dagger\alpha - \hat{a}_k\alpha^*\right)\end{aligned}$$

The second step is the expectation value and uncertainty for these states. For a general observable, they are given respectively by:

$$\langle A\rangle = \langle\Psi|A|\Psi\rangle \qquad (\Delta A)^2 = \left\langle\hat{A}^2\right\rangle - \left\langle\hat{A}\right\rangle^2 \tag{15}$$

Calculating the uncertainty of $\hat{x}$ knowing it is defined by Eq. 8, we have $\Delta x^2 = \langle\alpha|\,\hat{x}^2\,|\alpha\rangle - (\langle\alpha|\,\hat{x}\,|\alpha\rangle)^2$ where:

$$\langle\alpha|\,\hat{x}\,|\alpha\rangle = \frac{\alpha+\alpha^*}{\sqrt{2}} \quad\rightarrow\quad (\langle\alpha|\,\hat{x}\,|\alpha\rangle)^2 = \frac{\alpha^2 + (\alpha^*)^2 + 2\alpha\alpha^*}{2}$$

$$\begin{aligned}\langle\alpha|\,\hat{x}^2\,|\alpha\rangle &= \langle\alpha|\,(\hat{a}\hat{a} + \hat{a}^\dagger\hat{a}^\dagger + \hat{a}\hat{a}^\dagger + \hat{a}^\dagger\hat{a})\,|\alpha\rangle/2\\ &= \langle\alpha|\,(\hat{a}\hat{a} + \hat{a}^\dagger\hat{a}^\dagger + 1 + 2\hat{a}^\dagger\hat{a})\,|\alpha\rangle/2\\ &= (\alpha^2 + (\alpha^*)^2 + 1 + 2\alpha^*\alpha)/2.\end{aligned}$$

Combining the results into Δx^2,

$$\Delta x^2 = \frac{1}{2} \tag{16}$$

The same can be done to Δp^2 to also obtain $\Delta p^2 = 1/2$. According to Heisenberg's uncertainty principle, $\Delta x \Delta p \geq \hbar/2$. As we see, coherent states are states of minimum uncertainty, as they saturate the Heisenberg uncertainty bound for the variances of x and p.

As we have seen, Fock states can have arbitrarily high mode occupation numbers n so the Hilbert Space has infinite dimension. In addition, coherent states are a continuous variables system. So once more, we use the phase space formalism. In the book Barnett and Radmore (1997) (section 4.4) is defined s-ordered characteristic function $\chi(\xi, s)$ relying on the operator $\hat{\mathcal{D}}(\alpha)$:

$$\chi(\xi, s) = Tr[\rho \hat{\mathcal{D}}(\xi)] \exp\left(s|\xi|^2/2\right)$$

where, for N modes, ρ is the density operator of the quantum state with dimension $2N$, s the order ($s = 1$, 0 or -1 for normal, symmetric or antinormal order) and $\xi \in \mathbb{R}^{2N}$ was already defined but only for $N = 1$ for equation 14.

We obtain the expectation value of an operator doing the differentiation of $\chi(\xi, s)$ to a given order s over ξ as opposed to the double integral when using $W(x, p)$. The mentioned quasi-probability distribution is a function obtained by the Fourier Transform of the characteristic function:

$$W(\alpha, s) = \frac{1}{\pi^2} \int_{-\infty}^{\infty} \chi(\xi, s) \exp(\alpha \xi^* - \alpha^* \xi) \, \mathrm{d}^2\xi$$

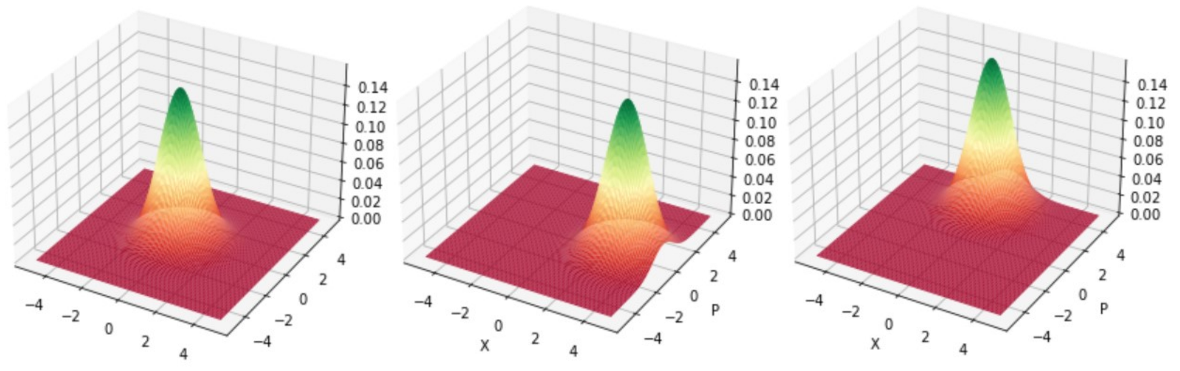
The density matrix is obtained then by:

$$\rho = \int_{-\infty}^{+\infty} W(\alpha, s) \left|\alpha\right\rangle \left\langle\alpha\right| \mathrm{d}^2\alpha$$

Combining the displacement properties, the uncertainties and the Wigner function, these states are displaced by a value α from the origin of the phase space, has uncertainties $\sigma_x = \sigma_p = 1/2$ and the probability distribution for quadratures is given by marginals of $W(\alpha, s)$. In figure 3 we can observe this for two different coherent states and compare with the vacuum once more. Indeed the operation displaces the Gaussian corresponding to the vacuum state from the center and the bigger the module of α, the bigger the displacement. It is also possible to observe the action of a state with same module but different imaginary part.

2.3.2 Squeezed states

Coherent states are not the only states with the possible minimum uncertainty states and here are introduced the squeezed states that can also behave at the minimum value of uncertainty Saleh and Teich (1991). Although the product $\Delta x \Delta p$ can not be lower than $1/2$,



(a) (b) (c)

Figure 3: Representation of coherent states in phase space. In figure 3a we have the vacuum state for contrast. In figure 3b we visualize the coherent state $|\alpha = 1.5 + 0j\rangle$ and in figure 3c $|\alpha = 1.32 + 0.72j\rangle$ (state with same module but distinct angle).

the uncertainty of one of the quadrature components can be reduced at the cost of increased uncertainty of other so that the product remains constant, i.e.

$$\Delta\hat{X}^2 = \frac{1}{4}\gamma \; ; \; \Delta\hat{P}^2 = \frac{1}{4\gamma} \tag{17}$$

Constructing once more the theory analogous for the coherent states, suppose we have a new set of annihilation and creation operators in terms of x and p given by:

$$b = \sqrt{\frac{m\omega'}{2\hbar}}\left(x + i\frac{p}{m\omega'}\right)$$

$$b^\dagger = \sqrt{\frac{m\omega'}{2\hbar}}\left(x - i\frac{p}{m\omega'}\right)$$

such that the state $|\beta\rangle$ is a right eigenstate of operator b, this is:

$$b\,|\beta\rangle = \beta\,|\beta\rangle$$

One can easily rewrite them as $b = \lambda a + \nu a^\dagger$ and $b^\dagger = \lambda a^\dagger + \nu a$ where

$$\lambda = \frac{1}{2}\left(\sqrt{\frac{m\omega'}{\hbar}}x_0 + \frac{1}{x_0}\sqrt{\frac{\hbar}{m\omega'}}\right) \quad ; \quad \nu = \frac{1}{2}\left(\sqrt{\frac{m\omega'}{\hbar}}x_0 - \frac{1}{x_0}\sqrt{\frac{\hbar}{m\omega'}}\right)$$

Similar to the annihilation and creation operators, these operators obey the commutation relation $[b, b^\dagger] = 1$. Calculating the uncertainty for these states using again equations in 15 it results in:

$$\Delta x^2 = \frac{\hbar}{2m\omega'} \quad \text{and} \quad \Delta p^2 = \frac{\hbar m\omega'}{2}$$

So we indeed have uncertainties like equation 17 (up to reformulation of quadrature operators to position and momentum) and their product is constant has predicted in the limit uncertainty.

The squeezed states can the obtained from an unsqueezed one with the squeeze operator $S(\zeta)$:

$$S(\zeta) = \exp\left(\frac{1}{2}[\zeta^* \hat{a}^2 - \zeta(\hat{a}^\dagger)^2]\right) \quad \rightarrow \quad |\zeta\rangle = S(\zeta)\,|0\rangle \tag{18}$$

where $\zeta \in \mathbb{C}$ such that $\zeta = re^{i\phi}$ and r is the squeezing parameter and θ the angle representing the orientation of squeezing as should become clearer soon when we see some examples. It is a unitary operator since:

$$S^\dagger(\zeta) = S^{-1}(\zeta)$$

This state obtained by $S(\zeta)\,|0\rangle$ can too be written in the Fock basis which will come a handy in the quantifying measurements. The state obtained when we apply the squeezing operator on the vacuum state is Barnett and Radmore (1997):

$$|\zeta\rangle = \sqrt{\sinh r} \sum_{n=0}^{\infty} \frac{\sqrt{(2n)!}}{2^n n!} \left(-\exp(i\phi)\tanh r\right)^n |2n\rangle \tag{19}$$

In terms of the phase space, the coherent states have a distribution that takes a circular form as in figure 3, whereas the unbalance in uncertainties for squeezed states translates as an elliptical shape, as illustrated in figure 4.

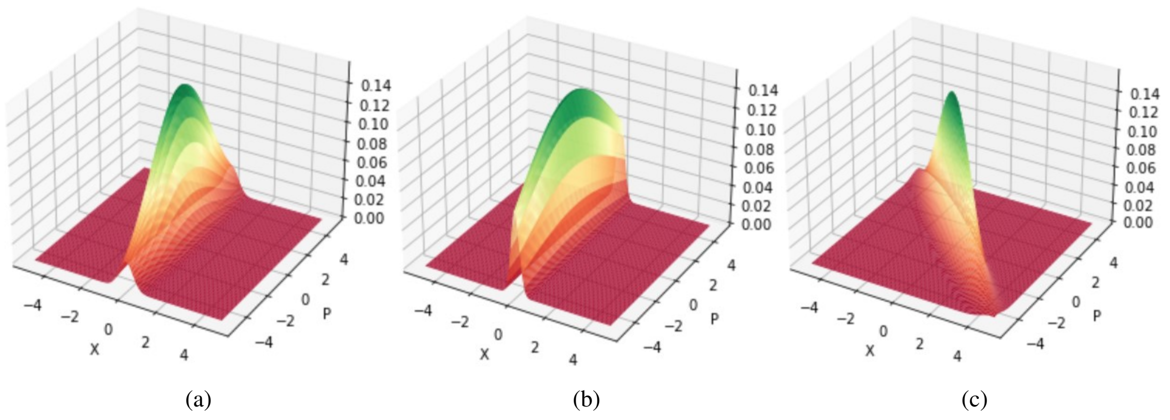


Figure 4: Representation of squeezed states in phase space. The three states illustrated have in common that they are placed in the origin of plane xp. In figure 4a the state is $|\zeta = 1\rangle$ so it took a vacuum state and squeezed the position quadrature by 1. In figure 4b is the same state with increased squeezing in the same quadrature. We can observe the function being compressed in x while in p had the opposite effect. Finally in figure 4c is the state $|\zeta = 0 + 1j\rangle$, is a state with squeezing $r = 1$ but $\theta = \pi/2$ where the rotation is quite visible.

Decibel (dB) is a common unit of squeezing in experiments. The amount of squeezing in dB corresponds to $10\log_{10}(2\langle \Delta X^2 \rangle)$ if squeezed along the X quadrature.

Properties of the Squeezing Operator

Using the lemma in Eq. 13 and definition in Eq. 18, the annihilation operator is transformed by the squeezing operation as follows:

$$\begin{aligned} S(\zeta)^{\dagger}\hat{a}S(\zeta) &= \hat{a} + \zeta\hat{a}^{\dagger} + \frac{|\zeta|^2}{2!}\hat{a} + \frac{\zeta|\zeta|^2}{3!}\hat{a}^{\dagger} + ... \\ &= \hat{a}\cosh r + \hat{a}^{\dagger}e^{i\phi}\sinh r \\ &= \mu\hat{a} + \nu\hat{a}^{\dagger}. \end{aligned} \tag{20}$$

Once more, the transformation can be also for $\hat{a}^{\dagger}$ resulting in:

$$\begin{aligned} S^{\dagger}(\zeta)\hat{a}^{\dagger}S(\zeta) &= \hat{a}^{\dagger}\cosh r + \hat{a}e^{-i\phi}\sinh r \\ &= \mu\hat{a}^{\dagger} + \nu^{*}\hat{a}. \end{aligned} \tag{21}$$

These operators obtained for transformation of $\hat{a}$ and $\hat{a}^{\dagger}$ are quite similar to the operators $\hat{b}$ and $\hat{b}^{\dagger}$ and in fact both work as ladder operators. The difference that leads to distinct results is that before ν and λ were real numbers and now they may be complex-valued. To sum up, squeezed state are eigenstates of the operator $S(\zeta)\hat{a}S^{\dagger}(\zeta)$.

Now the calculations for the uncertainty can be simplified using the previous operators. First, the expectation value of $\hat{x}$ is zero which will be an important characteristic for later in the vacuum squeezed state.

$$\begin{aligned} \langle\zeta|\hat{x}|\zeta\rangle = \langle\zeta|(\hat{a}+\hat{a}^{\dagger})|\zeta\rangle &= \langle 0|S^{\dagger}(\zeta)(\hat{a}+\hat{a}^{\dagger})S(\zeta)|0\rangle \\ &= \cosh r\,\langle 0|\,\hat{a}\,|0\rangle + e^{i\phi}\sinh r\,\langle 0|\,\hat{a}^{\dagger}\,|0\rangle + \\ &\qquad + \cosh r\,\langle 0|\,\hat{a}^{\dagger}\,|0\rangle + e^{-i\phi}\sinh r\,\langle 0|\,\hat{a}\,|0\rangle \\ &= 0. \end{aligned}$$

Second, the uncertainty itself is then:

$$\langle\zeta|\Delta\hat{x}^2|\zeta\rangle = \frac{1}{2}\exp(-2r) \text{ and for momentum } \langle\zeta|\Delta\hat{p}^2|\zeta\rangle = \frac{1}{2}\exp(2r).$$

2.3.3 Generalizing and representation

The states analysed so far (except Fock states) obey to a general form: Gaussian States. The Wigner functions that represent these states in phase space have a Gaussian form as will be shown here.

Let us prove that coherent states have Wigner functions that are Gaussian. Starting from the characteristic function, we have:

$$\begin{aligned}\chi(\xi) &= Tr[\rho\hat{D}(\xi)] \\ &= \sum \langle n|\, \rho\hat{D}(\xi)\, |n\rangle \\ &= \sum_n \langle n|\alpha_0\rangle \langle\alpha_0|\, \hat{D}(\xi)\, |n\rangle \\ &= \sum_n \langle\alpha_0|\, \hat{D}(\xi)\, |n\rangle \langle n|\alpha_0\rangle \\ &= \langle\alpha_0|\, \hat{D}(\xi)\, |\alpha_0\rangle\,.\end{aligned}$$

First we used the cyclic property of trace and then used the identity over the Fock basis, this is:

$$Tr(ABC) = Tr(BCA) \qquad \sum_n |n\rangle\langle n| = \mathbb{1}$$

Putting the result in the Wigner function we obtain

$$W(\alpha) = \frac{1}{\pi^2}\int \mathrm{d}^2\xi\, e^{\xi^*\alpha - \xi\alpha^*} \langle\alpha_0|\, \hat{D}(\xi)\, |\alpha_0\rangle$$

Unfolding the definition in the integral, we finally obtain that for a coherent state the Wigner function is:

$$W(\alpha) = \frac{2}{\pi}e^{-2|\alpha-\alpha_0|^2} \tag{22}$$

We can take a general form of the function in this last equation 22 for N modes with a similar approach as used in Eq. 14. It results in the next two equations:

$$\chi(\xi) = \exp\left[-\frac{1}{2}\xi^\intercal(\Omega\sigma\Omega^\intercal)\xi - i(\Omega d)^\intercal\xi\right]$$

$$W(X) = \frac{\exp\left[-\frac{1}{2}(X-d)^\intercal\sigma^{-1}(X-d)\right]}{(2\pi)^N\sqrt{\det\sigma}}$$

The functions have a particular general form: a constant multiplied by an exponential composed with a quadratic function, i.e. Gaussian. Another interesting property in the description of Gaussian states is that although for general states W can be negative, Gaussian states are always positive as we confirm some examples in previous figures. We have already seen Fock states with $n \geq 1$ as examples of non-classical states where $W < 0$ for some region in phase space.

On quantum phase space Γ Gaussian states are fully described by the first and second moments which are respectively the mean values (denominated displacement vector in form of a $2N$ vector d) and the variance (in form of a $2N \times 2N$ matrix σ called covariance matrix). Intuitively, the fist moment shows the displacement in the phase space that can be given by the displacement operator previously introduced $\mathcal{D}(\alpha)$ and the second moment encodes the uncertainty of the states giving not only the shape (circular, elliptical..) but the probability distribution too. The explicit formulas for each element of d and σ using R as in equation 9 are given by:

$$d_i = \langle \hat{R}_i \rangle_\rho \quad ; \quad \sigma_{i,j} = \frac{1}{2} \langle \hat{R}_i \hat{R}_j + \hat{R}_j \hat{R}_i \rangle_\rho - \langle \hat{R}_i \rangle_\rho \langle \hat{R}_j \rangle_\rho \tag{23}$$

The covariance matrix is a real and symmetric matrix that, in correspondence to density operator ρ, must satisfy two conditions: the commutation relations of x_{op} and p_{op} if using the basis for $\hat{R}$ and be positive semidefinite. Both are summarized in the following:

$$\sigma + i\Omega \geq 0 \tag{24}$$

2.4 linear-optical, continuous-variable transformations

Now suppose we have a linear transformation over the N modes described by a unitary matrix $U = \exp(-i\hat{H})$ where $\hat{H}$ is the Hamiltonian describing the dynamics.

The linear transformations adequate to this picture are also called Bogoliubov transformations and they act on a bosonic mode as the following:

$$U \begin{pmatrix} \hat{a} \\ \hat{a}^\dagger \end{pmatrix} U^\dagger = \begin{pmatrix} \alpha & \beta \\ \overline{\beta} & \overline{\alpha} \end{pmatrix} \begin{pmatrix} \hat{a} \\ \hat{a}^\dagger \end{pmatrix} \tag{25}$$

where the coefficients in the matrix are complex and $\overline{x}$ is the Hermitian conjugate of x. Since these transformations must preserve the commutation relations, they are canonical if $|\alpha|^2 + |\beta|^2 = 1$ so instead of one bosonic mode, if we deal with N then α and β represent a matrix that must obey the following equations:

$$\alpha\alpha^\dagger + \beta\beta^\dagger = \mathbb{1} \text{ and } \alpha\beta^\intercal = (\alpha\beta^\intercal)^\intercal \tag{26}$$

2.4.1 Beam splitters and phase shifters

Phase Shifter

The simplest linear-optical operation is the one obtained by a so-called phase shifter $P_\phi = e^{i\phi}$. It can be any material that slows down the wave in relation to others resulting in a phase shift of $e^{i\phi}$. Figure 5 shows a schematic of the representation.

We can see that it acts on a single mode. This transformation simply returns a phase-shifted state $|\Psi'\rangle$ depicted as:

$$|\Psi'\rangle = P_\phi |\Psi\rangle = e^{i\phi} |\Psi\rangle$$

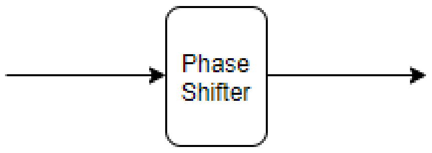


Figure 5: Representation of a phase shifter.

The phase shifter adds a phase to the state and its action on the annihilation operator is $e^{i\phi}\hat{a}$. The unitary is then $U_P = \exp(-i\phi\hat{a}^\dagger\hat{a})$ where the Hamiltonian is $H_P = \phi\hat{a}^\dagger\hat{a}$. Given the transformation, the matrix in form of Eq. 25 is:

$$\begin{pmatrix} e^{i\phi} & 0 \\ 0 & e^{-i\phi} \end{pmatrix} \begin{pmatrix} \hat{a} \\ \hat{a}^\dagger \end{pmatrix} \tag{27}$$

In a different basis (more helpful/direct for phase space representation), this gate rotates the position and momentum quadratures according to the next matrix which is called the symplectic matrix (see following sections).

$$\begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \hat{x} \\ \hat{p} \end{pmatrix}$$

Beamsplitter

A beamsplitter is a central component in interferometers so it is useful to understand its basic properties. Figure 6 shows a schematic representation of a beamsplitter. The role of the device is to split a beam of light according to their reflection and transmission coefficients. As a result, if r is real, an incident beam in input 1 will have a certain probability r^2 of being reflected, with a probability of transmission given by $1 - r^2$.

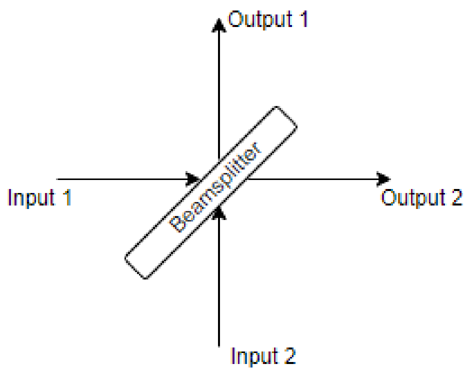


Figure 6: Representation of a beamsplitter.

Knowing light is defined by electric fields, one can translate the transformation of the beamsplitter has into a matrix of dimension 2×2 turning the input fields (Input 1 and 2 as E_{I1} and E_{I2}) into output fields (Outputs 1 and 2 as E_{O1} and E_{O2}). From figure 6, E_{I1} can either be reflected via E_{O1} or transmitted via E_{02} and the same for E_{I2} that can be reflected to E_{O2} or transmitted to E_{O1}. The matrix describing this is as follows:

$$\begin{pmatrix} E_{O1} \\ E_{O2} \end{pmatrix} = \begin{pmatrix} R_1 & T_1 \\ T_2 & R_2 \end{pmatrix} \begin{pmatrix} E_{I1} \\ E_{I2} \end{pmatrix}$$

As a result, each output is a linear combination of input. For a lossless beamsplitter, the total intensity at the outputs is the same as in the inputs. The device might change the phase of the beam as it passes through it and this is translated into a $\exp(i\phi)$ factor in transmitted arguments T_1 and T_2. We can write the reflection and transmission coefficients as the sine and cosine of an angle parameter θ that determines this role so $\cos^2\theta + \sin^2\theta = 1$.

All the above information combined leads to the final general matrix of the beamsplitter in Eq. 28. Note that there is no standard way to describe this transformation; the arguments are still θ and ϕ but they can be arranged differently in the matrix since the description of phase is arbitrary.

$$BS(\theta, \phi) = \begin{pmatrix} \cos\theta & e^{-i\phi}\sin\theta \\ -e^{i\phi}\sin\theta & \cos\theta \end{pmatrix} \tag{28}$$

With the above elements (phase shifter and beam splitter), it is possible to implement an arbitrary linear-optical evolution over two modes by choosing parameters θ and ϕ in equations 28 and 27 where the combination of them describes an interferometer represented by a matrix for an arbitrary number of modes that we will also study soon.

Since the photon number is preserved using this operation, the device is said to be passive.

We now find the beam splitter dynamics on creation and annihilation operators, as done previously with the phase shifter. Equation 28 is almost in the matrix form required in 25. The calculations for two-mode associated with input operators $a, a^\dagger$ and output operators $b, b^\dagger$ and unitary for beamsplitter U_{BS} must result in:

$$U_{BS}\hat{a}^\dagger U_{BS}^\dagger = \cos\theta\hat{a}^\dagger + e^{-i\phi}\sin\theta\hat{b}^\dagger$$
$$U_{BS}\hat{b}^\dagger U_{BS}^\dagger = -e^{i\phi}\sin\theta\hat{a}^\dagger + \cos\theta\hat{b}^\dagger$$

Similar equations can be found for operators $\hat{a}$ and $\hat{b}$. The operator that leads to this action of creation and annihilation operator and respective Hamiltonian is then described by:

$$U_{BS} = \exp\left[\theta(e^{i\phi}a^\dagger b - e^{-i\phi}ab^\dagger)\right] \quad ; \quad H = \theta(e^{i\phi}a^\dagger b - e^{-i\phi}ab^\dagger)) \tag{29}$$

The beamsplitter matrix is subsequently given from Eq. 28 by a 4×4 matrix:

$$\begin{pmatrix} \cos\theta & -e^{-i\phi}\sin\theta & 0 & 0 \\ e^{i\phi}\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \cos\theta & e^{-i\phi}\sin\theta \\ 0 & 0 & -e^{i\phi}\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} a \\ b \\ a^\dagger \\ b^\dagger \end{pmatrix} \tag{30}$$

2.4.2 Interferometer Decomposition

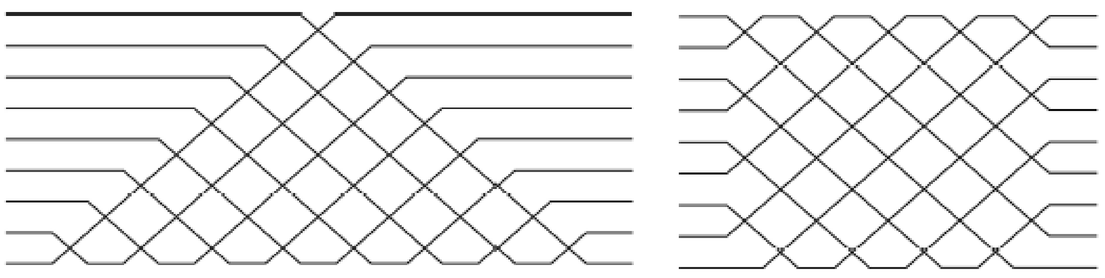
As seen in the last subsection, a linear-optical network can be described by phase shifters and beamsplitters. The combination of them is the interferometer and in this section we would like to discuss how to compose the action of many phase shifters and beam splitters to represent the action of an arbitrary linear-optical transformation on any number of modes of the electromagnetic field.

A beamsplitter is a 2×2 matrix so given N modes to process, a beamsplitter operator acts on two of them (n and m for example). Therefore, the matrix translating this operation is formed by the identity matrix except on rows and columns n and m where we substitute the diagonal over the beamsplitter matrix in Eq.28. For example, it has the following general form acting on two consecutive modes n and $m = n + 1$:

$$T_{n,m}(\theta,\phi) = \begin{bmatrix} 1 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & 1 & & & & & 0 \\ \vdots & & \ddots & & & & \vdots \\ \vdots & & & \cos\theta & e^{-i\phi}\sin\theta & & \vdots \\ \vdots & & & -e^{i\phi}\sin\theta & \cos\theta & & \vdots \\ \vdots & & & & & \ddots & \vdots \\ 0 & \dots & \dots & \dots & \dots & \dots & 1 \end{bmatrix}$$

Arranging these variables (θ and ϕ), any unitary matrix can be described by a sequence of beamsplitters in a specific order acting on the different modes as the matrix multiplication of every $T_{n,m}(\theta,\phi)$ for n and m in this arrangement. The mathematical description allowing this decomposition can take more than one form. The two best known are a triangular shape by Reck et al. (1994) and a rectangular shape by Clements et al. (2016) and both are represented in the figure 7.

The choice of design is informed by the error that can be expected. On one hand, for a photon entering the first qumode (upper mode) using for instance 9 modes, the Reck design only has 1 beamsplitter meanwhile Clements design has 5 beamsplitters. On the other hand, for the last mode (here mode number 9), Reck design has 8 beamsplitters but the number in



(a) Reck Design (b) Clements Design

Figure 7: Two possible designs for interferometer decomposition.

Clements design remains constant (5). Both methods use $N(N-1)/2$ beamsplitters to implement a $N \times N$ interferometer so the difference is exactly on how they are applied.

With this analysis the models work in different ways and depending on the application one might be better than the other. However, Clements design has been the one most used because although it induces more error in the first modes, the total error induced for all modes is the same. For the Reck design it might work better for the first modes but overall it induces much more errors in the last modes and they became disproportional in two senses: in asymmetric losses and more error overall.

The approach chosen her will be the one by Clements for the reasons above. It is also the one used in Strawberry Fields by default only using another if explicitly specified.

The rectangular description follows the next decomposition method which relies on two properties:

1. Process nulling of an element of U: given any unitary matrix U, there are specific values of θ and ϕ that turn any target element of in row n or m of matrix $T_{n,m}U$ to become zero;

2. Any element in column n or m of U can be also nulled by multiplying U on the right by matrix $T_{n,m}^{-1}$.

Then, by combining $T_{n,m}$ and $T_{n,m}^{-1}$ sequentially to null elements of diagonals of U, it is progressively turned into a lower triangular matrix. This is the basic point we need to reach because of a simple detail about diagonal matrices and unitaries. About this point a lot can be discussed however here we only need one proof: if a matrix A is both triangular and unitary, then it is diagonal. Let us prove this. If the inverse A^{-1} of an lower triangular matrix A exists, then it is lower triangular and, moreover, since A is unitary $A^{*} = A^{-1}$, so the transpose of A is upper triangular too. The only way for this to work is if A is diagonal. Now this diagonal matrix we found is a unitary D that can represent phase shifters in each mode of our circuit.

So, after all this we are left with a diagonal unitary matrix as a result of multiplication of U and respective $T_{m,n}$ and $T_{m,n}^{-1}$ for diagonal nulling:

$$\left(\prod_{(m,n)\in S_L} T_{m,n}\right) U \left(\prod_{(m,n)\in S_R} T_{m,n}^{-1}\right) = D \Rightarrow U = \left(\prod_{(m,n)\in S_L^T} T_{m,n}^{-1}\right) D \left(\prod_{(m,n)\in S_R^T} T_{m,n}\right)$$

S_L and S_R are the orderings for matrices $T_{m,n}$ and $T_{m,n}^{-1}$ respectively. D can be rearranged since it represents phase shifters in order to work only with direct matrices instead of inverses such that $T_{m,n}^{-1}D = D'T_{m,n}$ leading to the final equation of this construction:

$$U = D' \left(\prod_{(m,n)\in S} T_{m,n}\right)$$

Using Strawberry Fields library to get results from this algorithm, we first generated an arbitrary matrix and created the program (see in Appendices B.2, code B.5) and obtained the following circuit in figure 8.

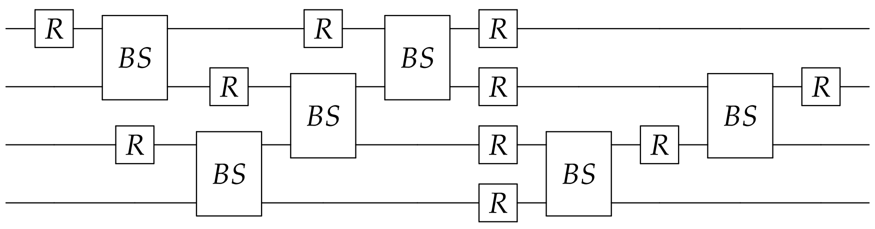


Figure 8: Decomposition of matrix U with function provided by Strawberry Fields based on Clements design.

The Clements decomposition is the default in Strawberry Fields, however, it is possible to add an argument and change the method. Now we can compare the physical implementation a quantum computer would make for our arbitrary matrix U with the decomposition method by Clements. Let us begin with the number of elements. Above we said it uses $N(N-1)/2$ beam splitters so, for this case, $N = 4$ and $4(4-1)/2 = 6$ which corresponds to the circuit. Then we have a series of rotation gates. Each mode passes at least through one rotation gate and there are 10 for 4 modes: 4 of them is related to our diagonal matrix D in decomposition and the remaining 6 are associated to each beam splitter (the returned parameters for BS is only over θ and the phase is implemented separately).

2.4.3 Symplectic transformations

Back to the evolution of the states, in the Gaussian form the states where described differently so since we are using another approach, how do elements of d and σ change with

Gaussian operations? In the Hilbert space notation, a linear optic network can be described by a unitary matrix U applied to the states:

$$\rho \to U\rho U^\dagger \quad ; \quad |\Psi\rangle \to U|\Psi\rangle$$

In phase space i.e. the quadrature operators, this unitary U must be translated to operations in d and σ that preserve the uncertainty principle and commutation relations in eqs. 9 and 24. The matrices capable of such transformations for these conditions are symplectic: a $2N \times 2N$ matrix S with real entries performs this if:

$$S\Omega S^\intercal = \Omega$$

Note that these matrices only hold for unitaries corresponding to Hamiltonians whose exponents are, at most, quadratic in creation and annihilation operators. The symplectic matrices can be for different bases; nevertheless they must satisfy $S\Omega S^\intercal = \Omega$ for the appropriate Ω for each specific basis as we'll soon see examples. Besides matrices, we can add an arbitrary constant vector b to the mean that is related to the action of the displacement operators $\mathcal{D}(\alpha)$ performing $d \to d + b$. Representing this operation, it does not change the uncertainties (either product for quadratures or each separately) so it has no effect in σ.

Summing up, the unitary corresponding to a symplectic transformation $U_{S,b} = \mathcal{D}(b)U_S$ in d and σ is:

$$d = Sd + b \quad ; \quad \sigma = S\sigma S^\intercal \tag{31}$$

Any symplectic transformation like equation 31 is generated by unitary transformation induced by bosonic Hamiltonians of the form of equation 32 where $h.c.$ stands for Hermitian conjugate.

$$H = \sum_n c_n^{(1)} a_n^\dagger + \sum_{n>k} c_{n,k}^{(2)} a_n^\dagger a_k + \sum_{n,k} c_{n,k}^{(3)} a_n^\dagger a_k^\dagger + h.c. \tag{32}$$

Corresponding to what was seen so far, the first term is linear to the field modes and the unitary transformation seen like this is the displacement operator 11. The second term involves two field modes so it mixes different modes and a operation in this context is beamsplitter 28. The last term corresponds to the squeezing operator (either for one mode when $n = k$ which is the only case already seen or two modes when $n \neq k$).

Let us recall the properties in the end of Gaussian States section, the operations on Gaussian states we are looking for here have the following characteristics:

- They are unitary operators. They are associated to an Hamiltonian H in the unitary matrix $U := \exp(-itH)$. Usually the time dependency is already built in the Hamiltonian or not explicitly considered because the interest in the operation works as a step, this is, a discrete evolution obtained for example by a time-independent Hamiltonian acting for a specified time step.

- In order to maintain working with Gaussian states, the operations must preserve the Gaussian nature of them. This type of generators that lead to Gaussian operations are functions of second-degree or lower to the quadrature components of the system exactly of the form 32.

Now let us then review operations mentioned so far and find the appropriate symplectic form for each.

Let us start with the rotation gate. Consider the Hamiltonian operator $\hat{H} = \hbar\omega\hat{a}^\dagger\hat{a}$, corresponding to the unitary operator:

$$U(t) = \exp\left(-it\hat{H}/\hbar\right) \to U(\phi) = \exp(i\phi\hat{n})$$

Using equation 10, the application of U to a coherent state α is then:

$$|\alpha\rangle = \exp\left(-|\alpha|^2/2\right)\sum_{n=0}^{\infty}\frac{\exp(i\phi n)\alpha^n}{\sqrt{n!}}|n\rangle$$

which leads to

$$U(\phi)|\alpha\rangle = \left|\alpha e^{i\phi}\right\rangle \tag{33}$$

So the effect of this gate in the state $|\alpha\rangle$ is a rotation in phase space also known as a phase shifter (the change of names here is due to the nomenclature in library Strawberry Fields). With this we have directly the symplectic form:

$$S_R(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \tag{34}$$

Our next element is beam splitter. The action on the annihilation operators with Eq. 13 confirms the results already from the first analysis:

$$U^\dagger_{BS}(\theta,\phi)\hat{a}_1 U_{BS}(\theta,\phi) = \hat{a}_1\cos\theta - \hat{a}_2 e^{-i\phi}\sin\theta$$

$$U^\dagger_{BS}(\theta,\phi)\hat{a}_2 U_{BS}(\theta,\phi) = \hat{a}_2\cos\theta + \hat{a}_1 e^{+i\phi}\sin\theta$$

However, the matrices we are looking for is in a different basis so the action on quadrature operators is:

$$U^\dagger_{BS}(\theta,\phi)\hat{x}_1 U_{BS}(\theta,\phi) = \hat{x}_1\cos\theta - \sin\theta(\hat{x}_2\cos\phi + \hat{p}_2\sin\phi)$$

$$U^\dagger_{BS}(\theta,\phi)\hat{p}_1 U_{BS}(\theta,\phi) = \hat{p}_1\cos\theta - \sin\theta(\hat{p}_2\cos\phi - \hat{x}_2\sin\phi)$$

$$U_{BS}^{\dagger}(\theta,\phi)\hat{x}_2 U_{BS}(\theta,\phi) = \hat{x}_2\cos\theta + \sin\theta(\hat{x}_1\cos\phi - \hat{p}_1\sin\phi)$$
$$U_{BS}^{\dagger}(\theta,\phi)\hat{p}_2 U_{BS}(\theta,\phi) = \hat{p}_2\cos\theta + \sin\theta(\hat{p}_1\cos\phi + \hat{x}_1\sin\phi)$$

which leads to the symplectic matrix of the form of Eq. 35 for the basis we are looking for in Eq. 9.

$$\begin{aligned} S_{BS}(\theta,\phi) &= \begin{pmatrix} \cos\theta & 0 & -\sin\theta\cos\phi & -\sin\theta\sin\phi \\ 0 & \cos\theta & +\sin\theta\sin\phi & -\sin\theta\cos\phi \\ \sin\theta\cos\phi & -\sin\theta\sin\phi & \cos\theta & 0 \\ \sin\theta\sin\phi & \sin\theta\cos\phi & 0 & \cos\theta \end{pmatrix} \\ &= \begin{pmatrix} \cos\theta\mathbb{1}_2 & -\sin\theta S_R(-\phi) \\ \sin\theta S_R(\phi) & \cos\theta\mathbb{1}_2 \end{pmatrix}. \end{aligned} \tag{35}$$

To rearrange to the usual arrangement, we change to the same basis but another organization: $\hat{T} = (\hat{x}_1, ..., \hat{x}_N, \hat{p}_1, ..., \hat{p}_N)^{\intercal}$ and, as such, the symplectic condition uses

$$\Omega = \begin{pmatrix} 0 & \mathbb{1} \\ -\mathbb{1} & 0 \end{pmatrix}$$

in this case, the symplectic matrix is:

$$S_{BS}(\theta,\phi) = \begin{pmatrix} \cos\theta & -\sin\theta\cos\phi & 0 & -\sin\theta\sin\phi \\ \sin\theta\cos\phi & \cos\theta & -\sin\theta\sin\phi & 0 \\ 0 & +\sin\theta\sin\phi & \cos\theta & -\sin\theta\cos\phi \\ \sin\theta\sin\phi & 0 & \sin\theta\cos\phi & \cos\theta \end{pmatrix} \tag{36}$$

Now let us take a look at squeezing which was introduced as an operator for state preparation but it can be used in the middle of a circuit. A simpler way to determine the symplectic matrix is to calculate the action on the creation and annihilation operators (let us call it Heisenberg basis to distinguish) and make a change of basis from this to the quadrature operators. Let us take the vector $\hat{R}$ represented in equation 9 and make the necessary transformations to obtain in the basis we intend to get. To do so, let us use two basis changing matrices T and L that do the next two operations:

$$\begin{pmatrix}\hat{x}_1\\ \vdots\\ \hat{x}_N\\ \hat{p}_1\\ \vdots\\ \hat{p}_N\end{pmatrix} = T\begin{pmatrix}\hat{x}_1\\ \hat{p}_1\\ \vdots\\ \vdots\\ \hat{x}_N\\ \hat{p}_N\end{pmatrix} = T\hat{R} \quad ; \quad \begin{pmatrix}\hat{a}_1\\ \vdots\\ \hat{a}_N\\ \hat{a}_1^\dagger\\ \vdots\\ \hat{a}_N^\dagger\end{pmatrix} = L\begin{pmatrix}\hat{x}_1\\ \vdots\\ \hat{x}_N\\ \hat{p}_1\\ \vdots\\ \hat{p}_N\end{pmatrix} = L\cdot T\hat{R}$$

where each element of T is $T_{i,j} = \delta_{j,2i-1} + \delta_{j+2N,2i}$ and matrix L is defined as $L = \frac{1}{\sqrt{2}}\begin{pmatrix}\mathbb{1} & i\mathbb{1}\\ \mathbb{1} & -i\mathbb{1}\end{pmatrix}$.

The conditions for the matrix to be symplectic depends on the basis considered. In the mathematical description for Fock states it was already shown the matrices and conditions on the basis for the operators $\hat{a}$ and $\hat{a}^\dagger$ in equations 25 and 26 such that the matrix in this basis $S_{\hat{a}}$ satisfies

$$S_{\hat{a}}KS_{\hat{a}}^\dagger = K \text{ where } K = \begin{pmatrix}\mathbb{1} & 0\\ 0 & -\mathbb{1}\end{pmatrix}.$$

This comes a handy because sometimes it is easier to calculate the effect on $\hat{a}$ and $\hat{a}^\dagger$ and just do a simple matrix multiplication on this to get the conditions over $\hat{x}$ and $\hat{p}$.

For squeezing it is usually implemented one and two mode physically and here we consider these situations for the purpose of our work:

1. Single-mode squeezing

   Here we have the squeezing operator in Eq.18 $S(\zeta) = \exp\left(\frac{1}{2}[\zeta^*\hat{a}^2 - \zeta(\hat{a}^\dagger)^2]\right)$ and using Hadamard lemma of Eq. 13, it has already been calculated in equations 20 and 21 the action on creation and annihilation operators respectively which leads to the matrix:

$$S_{\hat{a}} = \begin{pmatrix}\cosh r & e^{i\phi}\sinh r\\ e^{-i\phi}\sinh r & \cosh r\end{pmatrix} \tag{37}$$

   To obtain the symplectic matrix in the basis of $\hat{R}$ we then use T and L:

$$T = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}; \quad L = \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{i}{\sqrt{2}}\end{pmatrix}$$

$$\begin{aligned} S_S(r,\phi) &= T^{\intercal} L^{\dagger} S_{\hat{a}} L T \\ &= \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{i}{\sqrt{2}} & \frac{i}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \cosh(r) & e^{i\phi}\sinh(r) \\ e^{-i\phi}\sinh(r) & \cosh(r) \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \end{pmatrix} \\ &= \begin{pmatrix} \cosh(r)+\sinh(r)\cos\phi & \sinh(r)\sin\phi \\ \sinh(r)\sin\phi & \cosh(r)-\sinh(r)\cos\phi \end{pmatrix} \end{aligned} \tag{38}$$

If we take a look at what this operator does in phase space, it squeezes a quadrature by an amount defined by $r > 0$ and an orientation given by $\phi \in [0, 2\pi[$. So if we take the particular case when this angle is $\phi = 0$ this simplifies to

$$\begin{aligned} S_S(r,0) &= \begin{pmatrix} \cosh(r)+\sinh(r) & 0 \\ 0 & \cosh(r)-\sinh(r) \end{pmatrix} \\ &= \begin{pmatrix} e^{r} & 0 \\ 0 & e^{-r} \end{pmatrix}. \end{aligned}$$

Applying this simple matrix followed by a rotation gate we don't have a loss of generality and we simplify the matrix representation. $\phi = 0$ corresponds to squeezing the variance of the position operator, while $\phi = \pi/2$ corresponds to the squeezing of momentum operator.

2. Two-mode squeezing

   The two-mode squeezing operator acts on the vacuum like $S_2(\zeta)\left|0\right\rangle$ and is defined as

$$S_2(\zeta) = \exp\left(\zeta^{*}\hat{a}_2\hat{a}_1 - \zeta\hat{a}_1{}^{\dagger}\hat{a}_2^{\dagger}\right) \tag{39}$$

   This operator can not be decomposed into only two single-mode squeezing operators; both operators in equations 18 and 39 arise from the third term of the Hamiltonian in equation 32 but in the first the photons are generated by the action of $\hat{a}^{\dagger 2}$ and the second by $\hat{a}_1^{\dagger}\hat{a}_2^{\dagger}$. However, if we add beam splitters to the product of two $S(\zeta)$ then the joint operation can be written as:

$$U_{BS}^{\dagger}(\pi/4,0)[S(\zeta)\otimes S(-\zeta)]U_{BS}(\pi/4,0)$$

   Using either way, calculating once more the action on the usual basis so far we get:

$$S_2^{\dagger}(\zeta)\hat{a}_1 S_2(\zeta) = \hat{a}_1\cosh r - \hat{a}_2^{\dagger}e^{i\phi}\sinh r$$

$$S_2^{\dagger}(\zeta)\hat{a}_2 S_2(\zeta) = \hat{a}_2\cosh r - \hat{a}_1^{\dagger}e^{i\phi}\sinh r$$

and for their Hermitian conjugates:

$$S_2^\dagger(\zeta)\hat{a}_1^\dagger S_2(\zeta) = \hat{a}_1^\dagger \cosh r - \hat{a}_2 e^{-i\phi} \sinh r$$

$$S_2^\dagger(\zeta)\hat{a}_2^\dagger S_2(\zeta) = \hat{a}_2^\dagger \cosh r - \hat{a}_1 e^{-i\phi} \sinh r$$

$$S_{\hat{a}} = \begin{pmatrix} \cosh r & 0 & 0 & -e^{i\phi}\sinh r \\ 0 & \cosh r & -e^{i\phi}\sinh r & 0 \\ 0 & -e^{-i\phi}\sinh r & \cosh r & 0 \\ -e^{-i\phi}\sinh r & 0 & 0 & \cosh r \end{pmatrix} \tag{40}$$

Changing basis, for $\phi = 0$ we get:

$$S_{2S} = T^\intercal L^\dagger S_{\hat{a}} L T = \begin{pmatrix} \cosh r \mathbb{1}_2 & \sinh r \mathbb{D} \\ \sinh r \mathbb{D} & \cosh r \mathbb{1}_2 \end{pmatrix}; \qquad \mathbb{D} = \mathrm{diag}(-1, 1)$$

In the case of single-mode squeezing, we already saw what the operation does to a state: it squeezes one quadrature at the cost of increasing the other. For two-mode squeezing, it also squeezes (this time two modes) and maintains zero mean but with some extra properties.

At first sight, something has to be different because we have the following inequality $S_{2S} \neq S_S \otimes S_S$. Two-mode squeezed states in Fock basis (Barnett and Radmore (1997)) are written as:

$$|r\rangle = \sqrt{1 - \tanh^2 r} \sum_{n=0}^{\infty} (-\tanh r)^n \, |n\rangle_a \, |n\rangle_b \tag{41}$$

From such a state $|r\rangle$ we can infer that occupation number of the two states are correlated: the state is probabilistic in the space but if there are n photons in mode a, then there must be n photons in mode b.

2.4.4 Entanglement

From the operations above, two-mode squeezing has some interesting properties particularly for quantum computing that deserve some attention. The effect on the states is only similar to single-mode squeezing and to see the difference let us return to the decomposition[1] of the gate:

$$S_2(r) = U_{BS}^\dagger(\pi/4, 0)[S(r) \otimes S(-r)]U_{BS}(\pi/4, 0) \tag{42}$$

1 This decomposition is used theoretically but physically both can be implemented: single-mode squeezing can be implemented by Spontaneous Parametric Down Conversion (SPDC) into one mode and two-mode squeeze can be directly by SPDC where the pair generated is emitted to non degenerate modes

With this expression we can see that the beam splitter is the key to entanglement (and is even a prerequisite) and the action on mode a is squeezing by r and on mode b by $-r$.

First of all, the physical meaning of entanglement of states is either they can not be written as a convex combination of tensor products of well-defined states of each subsystem. This is, if the state is separable then it is a non-entangled state and otherwise it is also relevant to see the correlation between them. This property described theoretically can be converted to mathematical equalities for e.g. discrete systems, whose discussion is simpler. A state is said to be non-separable if it can not be written as:

$$\rho = \sum_i \omega_i \rho_i^A \otimes \rho_i^B$$

where ω_i is a positive probability value and ρ^A and ρ^B are the density matrices for the subsystems A and B to analyse. Other than that, there are many different ways to verify such as with von Neumann entropy, measurements on the system and others.

This property of systems can be verified for different types of states; to do so, the only two mode gate we have so far (other than 2-mode squeezing but we saw it can be decomposed) is beam splitter so we can try to analyse the result of it applying to the three types of states given: Fock, coherent and squeezed.

The most generic transformation on Fock states is for input state $|n_1, n_2\rangle$. The output is a sum over the possible states $|N_1, N_2\rangle$ with some coefficient $B_{n_1,n_2}^{N_1,N_2}$ associated to each. This coefficients are in article Kim et al. (2002) where they make a complete analysis of entanglement by beam splitters. $B_{n_1,n_2}^{N_1,N_2}$ doesn't show much restrictions other than $n_1 + n_2 = N_1 + N_2$. However, the authors use the von Neumann entropy as a measure of entanglement and it exhibits a interesting behavior where it is not necessarily maximized for a $50:50$ beam splitter but for other similar values too and the reason why will become clearer next.

Considering a balanced beam splitter with two input states with same number of photons n each, it happens that:

$$U_{BS}\,|n,n\rangle = \sum_{m=0}^{n} e^{-i(n-2m)\phi} \left(\frac{1}{2}\right)^n \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} \binom{n}{2m-k} \frac{\sqrt{2m!(2n-2m)!}}{n!} \,|2m, 2n-2m\rangle$$

Despite all the coefficients associated to the possible states, one thing is for sure: the output state somehow pairs the particles so measuring it only returns even outcomes (odd number of photons have zero probability). Now we can conclude why a balanced BS does not maximize entanglement for all cases; for n an odd number the output would be distributed with odd-number states but these destructively interfere and do not appear according to our previous formula.

Another interesting case is the effect of a beam splitter acting on state $|0,n\rangle$ which gives:

$$U_{BS}\,|0,n\rangle = \sum_{k=0}^{n} c_k^n\,|k,n-k\rangle \quad ; \quad c_k^n = \binom{n}{k}^{1/2} r^k t^{n-k} e^{ik\phi}$$

In particular, if the beam splitter is balanced ($t = r = \sqrt{0.5}$), entanglement is maximized. For $n = 1$ it gives the known result

$$U_{BS}\,|0,1\rangle = \frac{1}{\sqrt{2}}\,|0,1\rangle + \frac{1}{\sqrt{2}}\,|1,0\rangle$$

The output state is maximally entangled for $U_{BS}(\pi/4,0)$ according to the definition given above either for n odd or even.

A related calculation can be done also for the coherent states. From the same article, we have the following action:

$$\hat{U}_{BS}\hat{D}_a(\alpha)\hat{D}_b(\beta)\,|0,0\rangle = \hat{D}_a(t\alpha + re^{i\phi}\beta)\hat{D}_b(t\beta - re^{-i\phi}\alpha)\,|0,0\rangle$$

This is an important result: apliying a beam splitter to two coherent states is the same as creating those states only with displacement operator changing each parameter according to r and ϕ of the beam splitter.

Lastly, the effect on squeezed states. As seen above we already have an intuition that it might create entanglement but to confirm via calculations, we have the next equality:

$$\hat{U}_{BS}\left(\frac{\pi}{4},\phi\right)\hat{S}_a(\zeta_1)\hat{S}_b(\zeta_2) = \hat{S}_a\left(\frac{\zeta_1 + \zeta_2 e^{2i\phi}}{2}\right)\hat{S}_b\left(\frac{\zeta_1 e^{-2i\phi} + \zeta_2}{2}\right)\hat{S}_{ab}\left(\zeta_1 e^{i\phi} - \zeta_2 e^{-i\phi}\right) \tag{43}$$

In this last equation we can see that two mode squeezing leads to a state that can not be written as separated operators and those created entanglement: Kim et al. (2002) studied the von Neumann entropy for a measure of entanglement and they found that entanglement of the output state depends on the degrees of squeezing for input fields and the reflection coefficient which maximizes for $\theta = \pi/4$. Also the phase parameter in the beam splitter and the phase in squeezing parameters ζ_1 and ζ_2 plays an important role so ϕ in equation 43 must be restricted to $\phi = k\pi/2$ with $k \in \mathbb{Z}$. From this last analysis, only coherent states can not be entangled with a beam splitter, independently of the parameters ϕ and θ since the entanglement property in output states is strongly related to the nonclassicality of input states and coherent states are considered the most classical-like states.

What is actually happening when applying this operator S_{2S} to $|0\rangle_a\,|0\rangle_b$, it creates a two-mode squeezed vacuum state, also known as an Einstein-Podolski-Rosen (EPR) state. Entanglement is an essential ingredient for quantum computation and was first introduced with the EPR paradox by Einstein et al. (1935). In the particular case of continuous variable CV quantum information, it is crucial for quantum teleportation and quantum key distribution QKD protocols[2].

2.4.5 Detection and Measurements

Measurements are said to be Gaussian if, if acting on Gaussian states, they project onto other Gaussian states (Weedbrook et al. (2012)). In this we have two types of measurements: Homodyne and Heterodyne. They are defined respectively by the projectors as in eq. 44 where $x_\theta = \hat{q}\cos\theta + \hat{p}\sin\theta$.

$$\int \mathrm{d}x_\theta \left|x_\theta\right\rangle \left\langle x_\theta\right| = \mathbb{1} \quad ; \quad \frac{1}{\pi}\int \mathrm{d}^2\alpha \left|\alpha\right\rangle\left\langle\alpha\right| = \mathbb{1} \tag{44}$$

For the case of Homodyne detection, the state is projected over a quadrature basis $\left|x_\theta\right\rangle\left\langle x_\theta\right|$. These are not discrete and as such a measurement returns a real value $h \in \mathbb{R}$ with a probability distribution according to the Wigner function with an integral over x_θ.

For Heterodyne measurements the projection is onto coherent states. This is an example of positive operator-valued measure (POVM) Adesso et al. (2014) where the operators are Hermitian but they do not need to be orthogonal as is the case for coherent states like we saw before. These measurements are inherently still continuous with the particularity that can be seen as a simultaneous measurement of both quadratures since α is related to position and momentum by its real and imaginary parts. Both can not be measured at the same time without some degree of uncertainty so intrinsically the detector combines the state to be measure with a vacuum state into a beam splitter and measures each output with a homodyne detector.

Both these measurements on one mode leave the remaining multimode state still Gaussian.

An example of a non-Gaussian measurement is photodetection. This projective measurement reveals the particle-like nature (rather then wave-like) of the qumodes. Here the returning value is non-negative $h \in \mathbb{N}$. Also within this type, it can be divided into two measures:

1. von Neumann measurement; This is in the number states basis projecting the state onto $\left|n\right\rangle\left\langle n\right|$.
2. Avalanche measurement; This distinguish between $\left|0\right\rangle\left\langle 0\right|$ and $\mathbb{1} - \left|0\right\rangle\left\langle 0\right|$. It returns 0 if no photon arrives and 1 (or another convention) if one or more photons are detected.

For a coherent state, the probability of measuring n photons using states in terms of Fock basis 10 can be easily found to be:

$$\Pr(n) = \left|\left\langle n|a\right\rangle\right|^2 = \exp\left(-|\alpha|^2\right)\frac{\alpha^{2n}}{n!}$$

2 QKD offers many advantages over classical methods and in particular CV-QKD can offer some over discrete variable ones. This was already physically implemented by Zhang et al. (2019) over 50km.

The expectation value is then:

$$\langle \hat{n}|\hat{n}\rangle = \sum_{n=0}^{\infty} n \Pr(n) = \exp(-|\alpha|^2) \sum_{n=0}^{\infty} n \frac{\alpha^{2n}}{n!} = |\alpha|^2$$

For a simple squeezed state, the statistics are a bit different, as illustrated by the squeezed vacuum state that we have already encountered: $|\zeta\rangle = S(\zeta)\,|0\rangle$.

Since the expansion over the number basis contains only even numbers (equation 19), the probability for $\Pr(n)$ for odd n is zero and for even is:

$$\Pr(2n) = \operatorname{sech} r \frac{(2n)!}{(n!)^2 2^{2n}} (\tanh r)^{2n} \tag{45}$$

where sech denotes the hyperbolic secant. Then the mean photon number can be found directly from this result and the outcome is:

$$\overline{n} = \sinh^2 r \tag{46}$$

In figure 9 we show the probability distribution over photon numbers for measurements on squeezed vacuum states with different squeezing factors. The range of values in figure is only to demonstrate the evolution; the amount of squeezing implemented and achieved in physical experiments will be discussed later in chapters 4 and 5.

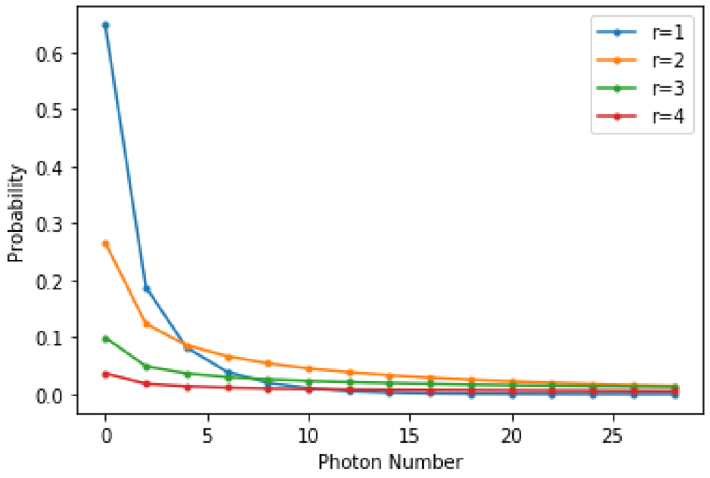


Figure 9: Probability for detecting different photon numbers as a function of the squeezing parameter, for the squeezed vacuum state.

As we can see, the squeezing operation is an active linear-optical operation, which in general increases the number of photons in the state.

In the case of two-mode squeezing operation, the state of both modes $|\zeta_{AB}\rangle$ written in the Fock basis is:

$$|\zeta_{AB}\rangle = \operatorname{sech} r \sum_{n=0}^{\infty} (-\exp^{i\phi} \tanh r)^n |n_A n_B\rangle$$

which should translate to a probability distribution of n given by the next equation 47.

$$P(n) = \operatorname{sech}^2 r (\tanh r)^{2n} \tag{47}$$

Except for the outcome $n = 0$, performing photodetection on a single mode of a multimode Gaussian system causes the resulting modes to become non-Gaussian. Thus, this measurement can be used as an ingredient to implement non-Gaussian operations.

2.5 summary

Quantum states of light can be represented in the phase space formalism which is a space that brings many advantages when dealing with continuous-variable states. The state in quantum representation is defined by a quasi-probability distribution Wigner function (or equivalently a Characteristic function) in such space that fulfill the requirements for minimum uncertainty and for the quasi-probability. With this we can represent each state and study properties such as classicality and many others.

Then we started introducing the states of light we will be using throughout this thesis and started by Fock states. They have a well defined number of particles and this is expressed in they ket representation in equation 7 which gives us:

$$|n\rangle = |n_1, n_2, ..., n_m\rangle = \prod_{i=1}^{m} \frac{(\hat{a}_i{}^\dagger)^{n_i}}{\sqrt{n_i!}} |0\rangle$$

In phase space, these states can show negative values for the Wigner function which indicates that they have a nonclassical behaviour.

Next we presented Gaussian states. The first in this group was Coherent states given by equations 10 and 11:

$$|\alpha\rangle = \exp(-|\alpha|^2/2) \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle = \exp\left(\alpha\hat{a}^\dagger - \alpha^*\hat{a}\right) |0\rangle = \hat{\mathcal{D}}(\alpha) |0\rangle$$

Their properties can be summed up by their phase space representation (see figure 3); generating coherent states from vacuum correspond to a displacement in phase space from the origin and each $\hat{\mathcal{D}}(\alpha)$ is a displacement by $\mathrm{Re}(\alpha)$ in x axis - position - and $\mathrm{Im}(\alpha)$ in y axis - momentum. They have equal uncertainty both in x and y axis (or any other rotated space) and they saturate the Heisenberg bound for variances. A pertinent property is that they never show negative values for the Wigner functions.

Continuing Gaussian states analysis, we proceeded with Squeezed states. They are summarized in equations 19 and 18:

$$|\zeta\rangle = \sqrt{\sinh r} \sum_{n=0}^{\infty} \frac{\sqrt{(2n)!}}{2^n n!} \left(-\exp(i\phi)\tanh r\right)^n |2n\rangle = S(\zeta)\,|0\rangle$$

Similar to coherent states, in phase space (see figure 4) they still do not present negative parts in Wigner function and are states of minimum uncertainty but unlike the previous, this uncertainty is not equally distributed. Here, the squeezing operation squeezes a quadrature at cost of increasing the other in the same amount as Heisenberg's principle demands.

These two states - coherent and squeezed - are called Gaussian states due to their special characteristics in phase space, more precisely the uncertainty behaviour since it is translated to a Gaussian function. As so, they are fully described by two entities: the displacement vector d and covariance matrix σ, both in equation 23.

These are the three families of states we introduced and the ones we will be using to initialize our qumodes in the two models of linear optical computation that we will discuss next. With this complete we began introducing linear-optical transformations and started with phase shifters and beam splitters that their actions on previous states are mathematically translated to equations 27 and 30. These elements combined form an interferometer and for computational purposes it is important to be able to decompose the interferometer given by an arbitrary matrix U into a combination of basic elements. Two decomposition methods are presented: Reck and Clements design. Both use the same number of elements but they are arranged very differently and usually the chosen approach is the one by Clements due to the symmetry it has from the point of view of losses and induced errors, as all input-output pairs go through the same number of beam splitters.

Continuing the analysis of evolution of states, we introduced the symplectic transformations as a way to study how elements of Gaussian states - namely displacement vector and covariance matrix - change with transformations. This alternative representation (as opposed to Hilbert space and discrete valued states as Fock states) brings advantages when dealing with continuous variable quantum states (as Gaussian ones that have an infinite-dimensional Hilbert space) because we can deal only with two entities: d and σ. Applying interferometer $U_{S,b} = \mathcal{D}(b)U_S$ their evolution is translated to equation 31, this is, a matrix multiplication. Since Gaussian states will be essential for our case, we rewrote matrices found so far (displacement, beam splitters etc) and took the opportunity to present two-mode squeezing. Two-mode squeezing is an operation of the same type of one-mode squeezing when considering quadrature squeezing but this type introduces an important mark: entanglement. These states can be obtained by physical generation (SPDC for instance) of by unitary decomposition (separate squeezing with parameters $r_1 = -r_2$ followed by a balanced beam splitter) and are of utmost importance for quantum computation. Both Fock states and squeezed can be entangled by a beam splitter but coherent states can not.

Last but not least, we discussed detection and measurement. We divided measurements essentially in Gaussian and non-Gaussian ones. The first preserves the Gaussian nature of states and includes Homodyne and Heterodyne measurements that are defined by projectors over the quadrature basis and onto coherent states respectively. The second is based on photo-detection either for projection on number operator (counting n) or avalanche detection (also called bucket detectors).

3

BOSON SAMPLING

In this chapter we will study the Boson Sampling model which was originally introduced by Aaronson and Arkhipov (2013). This is a model for quantum computation where the authors explored the complexity of simulation of non-interacting bosons undergoing a linear-optical evolution. The boson sampling model starts by initializing input qumodes as Fock states passing them through a linear interferometer and finally measuring the output with photodetection. Such program for simulation in a classical computer requires calculating the permanent of matrices which is hard to simulate and, opposed to this approach, a quantum computer would solve the problem naturally. In fact, the complexity of doing even an approximate simulation is exponential in the number of photons/modes, which motivated experiments as a way to show quantum computational advantage in practice.

With the concepts of linear optics from last chapter, here we will start by giving a mathematical description of the system for the particular conditions of boson sampling, then present both analysis (classical simulator and quantum simulator with the code for an actual photonic quantum computer), study and compare their complexity and finalize with some numerical experiments.

3.1 mathematical description

Linear optical quantum computers (LOQC) use both active and passive elements however, the Boson Sampling model only uses passive elements which can make the calculations easier: since the input states are Fock states $|n\rangle$ with $n \in \mathbb{N}$, the operators describing them can be restricted to creation operators $a^\dagger$ (recall that $\hat{a}|0\rangle = 0$ and applying $\hat{a}$ to $|n\rangle$ $n \neq 0$ would only cancel out the $\hat{a}^\dagger$ used to increase n); the matrix for this transformation only requires elements $\overline{\alpha}$ in the form in Eq. 25 leaving $\beta = 0$ specifically for active operations which matches the matrices for beamsplitter and phase shifter above. So the matrix for the transformation for m modes is described by a $m \times m$ unitary instead of $2m \times 2m$.

A beamsplitter acts on two modes as:

$$\begin{aligned}U_{BS}\left|n_1,n_2\right\rangle &= \frac{1}{\sqrt{n_1!n_2!}}U_{BS}\hat{a}_1^{\dagger n_1}\hat{a}_2^{\dagger n_2}\left|0,0\right\rangle \\ &= \frac{1}{\sqrt{n_1!n_2!}}U_{BS}\hat{a}_1^{\dagger n_1}\hat{a}_2^{\dagger n_2}U_{BS}^{\dagger}U_{BS}\left|0,0\right\rangle \\ &= \frac{1}{\sqrt{n_1!n_2!}}U_{BS}\hat{a}_1^{\dagger n_1}\hat{a}_2^{\dagger n_2}U_{BS}^{\dagger}\left|0,0\right\rangle\end{aligned}$$

As we can see Brod et al. (2019), the linear interferometer transforms creation operators so that:

$$\hat{a}_i^{\dagger} \to U\hat{a}_i^{\dagger}U^{\dagger} = \sum_{j=1}^{m} U_{i,j}\hat{a}_j^{\dagger}$$

Given two m-mode states in the form of Eq. 7, suppose the two input and output states respectively $|S\rangle$ and $|T\rangle$:

$$|S\rangle = |s_1,...,s_m\rangle \qquad |T\rangle = |t_1,...,t_m\rangle$$

The original matrix can be manipulated to obtain $U_{S,T}$, a sub-matrix of U. The entries of $U_{S,T}$ are constructed by taking s_i copies of row i and t_j copies of column j. With this, in Scheel (2005) it was shown that the probability of measuring in the output the state $|T\rangle$ having state $|S\rangle$ as input is given by:

$$\Pr(S \to T) = \frac{|\mathrm{perm}(U_{S,T})|^2}{\prod_i s_i! \prod_j t_j!} \tag{48}$$

where $\mathrm{perm}(U_{S,T})$ is the permanent of $U_{S,T}$ and is given by

$$\mathrm{perm}(U) = \sum_{\sigma\in S_n}\prod_{i=1}^{n} U_{i,\sigma_i} \tag{49}$$

where S_n is the set of all permutations of the set $\{1..n\}$. The permanent is not a common formula however most readers will be familiar with the determinant of matrices:

$$\det(U) = \sum_{\sigma\in S_n}\mathrm{sgn}(\sigma)\prod_{i=1}^{n} U_{i,\sigma_i}.$$

The only difference is the sign in the determinant that changes according to the permutation ($\mathrm{sgn}(\sigma) = +1$ is permutation is even and $\mathrm{sgn}(\sigma) = +1$ otherwise). Although this is a small difference in the formula and calculations, computationally it has a major impact. A naive proposal to solve them implies the sum of the equal number of terms since the number of permutations is one and the same however determinant shows some properties one can explore. One is $\det(AB)$ $=\det(A)\cdot\det(B)$ where we obtain the determinant of a matrix by calculating determinant of two matrices that are more easily computed and add this to the method using Gaussian elimination. This difference makes the determinant take only n^3 steps to be complete (with n the size of U) which is polynomial and considered easy to compute as we will expand in later sections.

3.2 classical implementation

The classical code follows directly from the mathematical description given above; the reader may find the full code in Appendix B.3 and the pseudo code associated is below in Algorithm 1.

Algorithm 1 Boson Sampling Classical Algorithm

```
function PROBC($U$, $S$, $T$)
    for $j \in m$ do
        if $t_j \neq 0$ then
            $U_{S,T} \leftarrow$ take $t_j$ copies of the $j$th row of U
        end if
    end for
    for $i \in m$ do
        if $s_i \neq 0$ then
            $U_{S,T} \leftarrow$ take $s_i$ copies of the $i$th colunm of U
        end if
    end for
    permanent $\leftarrow$ perm($U_{S,T}$)
    permanent $\leftarrow$ |permanent|$^2$
    factorial $\leftarrow$ fact(S)$\times$fact(T)
     return permanent/factorial
end function
```

To begin with, to realize the simulation according to the equation of the model Eq. 48 we need the linear interferometer unitary matrix U that describes the evolution of the system, the input state S and the output state T. Having this data as arguments of the main function, we proceed to the calculation. Note that our function assumes the programmer enters valid arguments to the physical circuit and description which means that input and output states must be valid Fock representations with total number of photons in S equal to T and U is an arbitrary unitary matrix. Boson sampling usually uses Haar-random matrices, to avoid symmetries that could simplify the classical calculation, and to obtain evidence for hardness-of-simulation based on a few well-understood complexity theoretic assumptions Aaronson and Arkhipov (2013). There are several reasons for the choice of Haar-measure matrices over arbitrary unitaries: uniformly distributed matrices for the free parameters of the random unitary lead to clusters in the poles of the Block sphere (see appendix A.2) which leads to a restricted or more constrained structure that classical simulation might exploit contrary to Haar-random unitary matrices; also, for these distributions, with high probability the outcomes will not have two photons or more per mode.

The key part in this code is the calculation of the permanent. The perm function implemented (see code B.6) is the basic one doing the calculation using the definition running for all permutations (eq. 49). Analysing the complexity we see that it runs in time $T(N)$:

$$T(N) = \begin{cases} 1 & \text{if } N \leq 1 \\ N \times ((N-1)^2 + T(N-1)) & \text{if } N \neq 1 \end{cases}$$

which represented in a non-recursive way corresponds to $\mathcal{O}(N!N)$ operations, which represents an intractable computation.

There are other algorithms that calculate the permanent more efficiently for some classes of matrices or algorithms that implement an approximate evaluation of the permanent. The most efficient known general algorithm was introduced by Ryser and has a run-time that scales as $\mathcal{O}(N2^N)$. It is still exponential but far better than the naive algorithm discussed above.

For m modes and n photons in input, the number of combinations is given by $B_{m,n}$:

$$B_{m,n} = \frac{(m+n-1)!}{n!(m-1)!} = \binom{m+n-1}{n}$$

Running the code for all input-output possibilities (combinations $B_{m,n}$), it takes:

$$\mathcal{O}\left(\binom{m+n-1}{n}(N2^N)\right) \text{ time steps.} \tag{50}$$

3.2.1 Simulation

Now we have a well defined model, the description of the system and the code to simulate it on a classical computer. To study applications or make some experiments, we need to do a simulation under certain/given conditions. Calculating the probabilities is a theoretical prediction and the complete calculation corresponds to a task that not even the quantum computer will do, as the quantum computer only outputs some sample of events, rather than the exact probabilities given by permanents. Also, soon we will study the complexity of computing Boson Sampling, which is a difficult problem (as Eq. 50 indicates). For these reasons, here we aim to simulate Boson Sampling by generating samples to predict the results from a photonic quantum computer.

In what concerns simulating quantum algorithms by a classical computer there are two different notions of simulation: strong and weak simulation. The most common is strong simulation - given a set of parameters (describing the physical system) and possible output events, it calculates the exact probabilities for each output which is equivalent to calculate the corresponding permanent, as specified by Eq. 48 directly with algorithm 1. Weak simulation consists of an algorithm that produces samples from the output distribution of the ideal quantum process. With strong simulation, we can obtain the probabilities associated with all possible outcomes and use that for weak simulation. Sometimes, however, weak simulation algorithms may exist that do not require strong simulation. Note also that weak simulation can be used to estimate the probabilities via the relative frequencies of events, but only if there are not too many possible output events. If the number of possible output events scales exponentially with the size of the system (as in Boson Sampling, recall the number of possible combinations $B_{m,n}$), we cannot hope to estimate all probabilities via weak sampling, as getting enough data for a reasonable estimate would involve accumulating data for a time interval that scales exponentially with the system size. The quantum computer itself only samples from its output distribution, so strong simulation is a stronger concept than what the experiment itself does.

We implemented two methods to generate samples; here are doing weak simulation after strong simulation i.e. we calculate the probability distribution and then sample from this. This is feasible here, as the size of the systems we are simulating are small. There is an algorithm presented in Clifford and Clifford (2017) that does not require strong simulation and for that reason it lowers the complexity of the algorithm but for now let us focus on the two generators implemented.

randomGeneratorA focuses on the rejection sampling method. This function takes as arguments the unitary U, the number of photons f for the input and number of samples $times$ to generate. f is sufficient to describe the input state since for m modes (taken from U) and f photons, the input state will be $S = \left|1_f, 0_{m-f}\right\rangle$ (restriction of the occupation numbers to only 0 and 1 in simulation will become clearer and is due to physical limitations). This method starts by generating all possible output combinations and saving in a numpy array $states$. Then, we pick a possible output state x using a uniform random distribution over the output space and pick y uniformly distributed in the interval $\in [0,1]$; each probability $y[i]$ corresponds to the state in x in the same position, this is $x[i]$. At last, we filter x accepting the states fulfilling the inequality:

$$\text{probC}(U, S, x[i]) \leq y[i]$$

Since the length of x is $times$ and it is filtered to select a set of accepted output events, it is extremely unlikely to accept them all so we run again the algorithm with an auxiliary function so that we don't require to make once more the initial calculations (the array $states$..).

Finally, the function returns the accepted states in a numpy array of the type:

$$\text{numpy.array}[\text{numpy.array}[m] * times]$$

To simplify the understanding, in figure 10 is represented the flowchart for this algorithm showing each step.

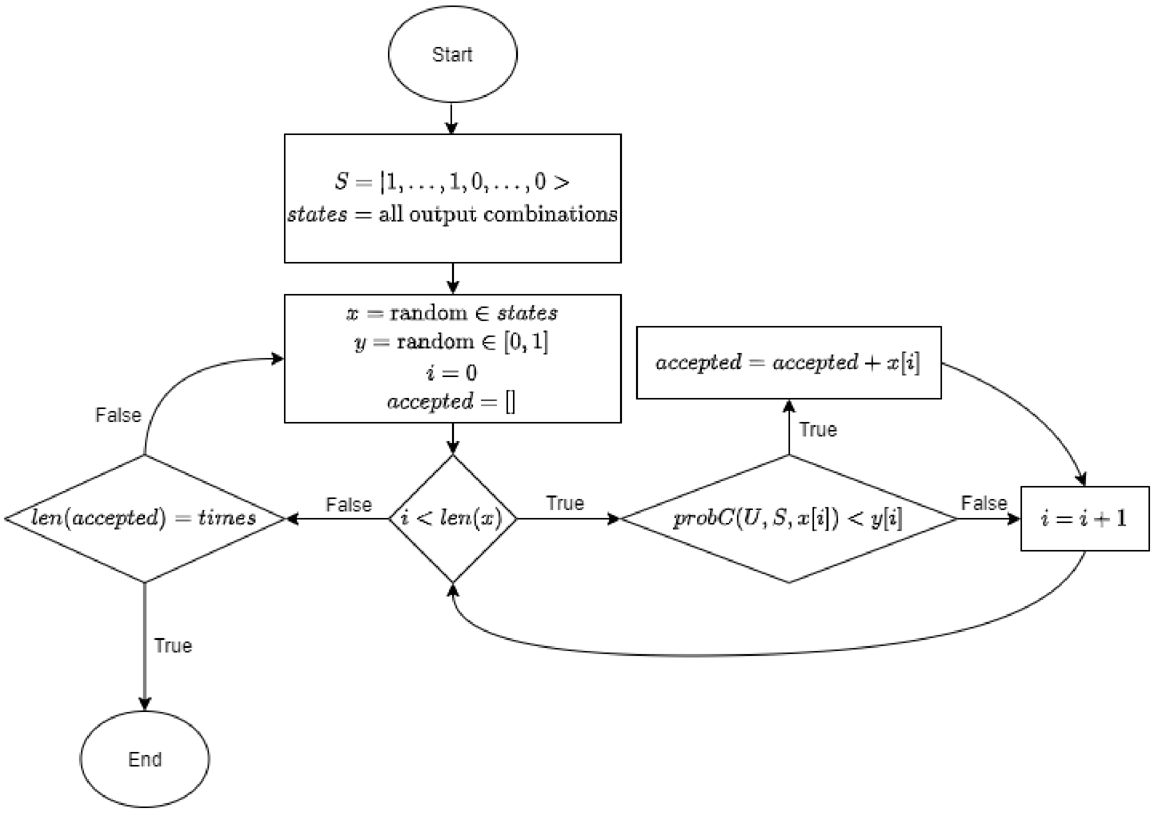

Start
$S = |1, \dots, 1, 0, \dots, 0>$
$states$ = all output combinations
x = random $\in states$
y = random $\in [0,1]$
$i = 0$
$accepted = []$
$accepted = accepted + x[i]$
True
False
$len(accepted) = times$
False
$i < len(x)$
True
$probC(U, S, x[i]) < y[i]$
False
$i = i + 1$
True
End

Figure 10: Flowchart of radomGeneratorA algorithm.

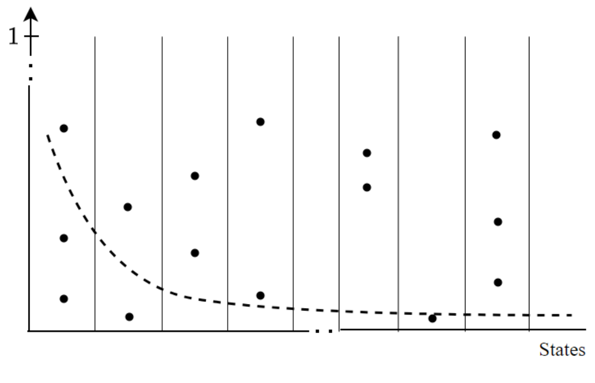
Now let us explore the second method by function randomGeneratorB; this one has the same arguments as randomGeneratorA and returns also an array of type numpy.array[numpy.array$[m] * times$] since they are both sampling from identical conditions only with different methods. Here, we start at the same point as before generating an array with all possible output combinations but then calculate the probability for each possible state and save the cumulative sum of the found values in a new array *probs*. This means that now we have an array in range 0 to 1 where each positional spacing corresponds to a probability of a state in array *states*. Next we randomly select values in this range in y. The distribution is uniform but the difference $probs[i+1] - probs[i]$ for $0 < i < len(states) - 1$ is not. The last step is associating the values in y to the corresponding state and save those in *accepted*.

Comparing the two approaches we have that:

- Both generate an array *states* of the possible output combinations for a given m and n;
- Both methods select random numbers from a uniform distribution: method A generates that for two arrays (one discrete for x yielding integers and one continuous in $[0,1]$) and method B generates one for the probabilities;

- Rejection sampling from method A can be represented as in sub-figure 11a where we only need to calculate probC$(U, S, x[i]) \leq y[i]$ for the chosen values of x. This means that we don't need to do strong simulation for all $\binom{m+n-1}{n}$ states at the cost of possibly having a high rejection rate;

- Method B requires calculating the probabilities for all output states before starting to chose samples which implies calculating $\binom{m+n-1}{n}$ permanents of size n but each value in *probs* corresponds to a sample so that rejection rate is null (all samples are used, see sub-figure 11b).



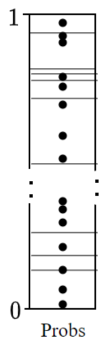


Figure 11: Methods for sampling with both processes: sub-figure 11a represents sampling with randomGeneratorA and sub-figure 11b represents sampling with randomGeneratorB. Dashed line in sub-figure 11a represents the theoretical probability evolution of given ordered states and the vertical lines represents the separation of states (it is a discrete value to chose in algorithm for x) but the same does not happen for the probabilities. Sub-figure 11b represents the cumulative vector *probs* and separations are proportional to the value in each position of *probs*. In both of them we represent arbitrary samples given by black dots.

3.3 quantum implementation

Now let us discuss the quantum implementation using the software Strawberry Fields in Python. The code presented below is still a classical implementation internally but it can run in a quantum computer by changing the engine from simulator to remote engine which will allow access to quantum chips in the future (see Appendix section B.1). The code is not as transparent to the mathematical description as the previous classical simulation was but it's quite intuitive and easy to understand in comparison with the physical implementation.

For example take the code in 3.1 This code has the same functionality as function probC from the previous classical implementation, taking the same input parameters:

```
def probQ(U, inputQ, states):
    m=len(inputQ)
```

```
3      boson_sampling = sf.Program(m)

5      with boson_sampling.context as q:
           for i in range(m):
7              if inputQ[i]==0: Vac     | q[i]
               else: Fock( inputQ[i] ) | q[i]
9          Interferometer(U)            | q

11     eng = sf.Engine(backend="fock", backend_options={"cutoff_dim": m+1})
       results = eng.run(boson_sampling)
13
       return results.state.all_fock_probs()
```

Listing 3.1: Boson Sampling Quantum Algorithm

Recall that linear-optical quantum computation starts with state preparation, in this particular model, with input Fock states $|n\rangle$ followed by linear interferometry and finally measure in Fock basis. In resemblance with quantum computation with the usual model, i.e. using qubits, we can easily see the commonalities and let us name them explaining the previous code.

The first thing to do is start a program for the linear-optical circuit we want and that is done in line 3 using the number of modes attributed to variable m. Then, according to the steps of quantum computation, we need to specify the initialization of the input states. In order to do that, we create a cycle to run the entries (with variable i) of our input inputQ and initialize that mode with a well-defined number state $n_i = inputQ[i]$ in lines 7 and 8.

After this, once the states are prepared, line 9 applies to our program the circuit represented by matrix U for evolution of initial states with the help of operation Interferometer(U) which we'll soon examine. The decomposition (made by a classical computer always) returns the parameters for each beam splitter and rotation operations that are the equivalent gates constituting our circuit (see circuit in figure 8).

The next step is to create the engine with the appropriate backend and it is not only important but necessary to put a limit in the dimension using backend_options because, as we know, the number of photons in a Fock state has no bound so we put one to be possible to simulate in computation terms.

In probQ the arguments are similar but not equal to probC and it is because of the function in return, namely all_fock_probs() that automatically generates the probabilities for all combinations. So lastly, we take results from the engine and return the probabilities associated with each of the possible outcomes ready to be used.

3.4 complexity analysis

Now let us start the comparison between these two methods (classical versus quantum implementation) by taking a look at the computational complexity classes first in order to be properly classify the run time and then see for the particular case of Boson Sampling.

3.4.1 Complexity classes

The first class we introduce is P problems where P stands for polynomial where a classical computer can run/solve the program/problem in a number of computational steps (time) that increases polynomially with the input size. This quantification is obtained estimating the number of basic computational steps and finding how it scales asymptotically, as we did for example in equation 50. The second class introduced is NP-problems where NP stands for non-deterministic polynomial. In this class, finding the solutions can be inefficient and have a characteristic that if they have one, it can be verified in polynomial time, that is, the computational problem of verifying a proposed solution to a NP problem can be done efficiently. In particular, all P problems are also in the NP class. To better illustrate this computational complexity class, there are several problems thought to be hard and which fall in this class, for example:

- $3-$coloring - the problem consists in attributing 3 different colors or labels to neighbours in a given structure usually a graph (we will develop this later in applications) that might represent a map, a social network or many others and see if one can color each neighbour with different colors. In order to see if there is a solution we need to run all nodes and its neighbours while attributing labels. Finding solutions for the worst case inputs is hard, but verifying it falls into P;

- Prime factoring - decomposing a given integer into product of smaller prime numbers. Once again, verifying a solution is a simple multiplication but finding the correct answer does not have a shortcut;

- Travelling salesmen - another problem solved with graph theory but consider a map of cities and distances between them and a salesmen; what is the shortest route for the salesmen to run every city exactly once and return to the origin? To find a solution we would have to run all cities and its neighbours similar to $3-$coloring continuously.

No algorithm was found for these problems that does not require running all possible paths i.e. where the number of steps does not scale exponentially with the problem size. Related to this and part of this second class, there's another (subclass) called NP-complete. The NP-complete set of problems have the property that every other NP problem can be reduced to them, that is, any problem instance can be translated to a problem instance of the complete problem. An example for such class is SAT problems (logical formula satisfiability) where we attribute logical properties to the problem and exhaust all possible results to find the correct one. It consists on creating a proper model (assignment for the logical variables) for a given Boolean formula such that it evaluates as True. If this model is found, the formula is called satisfiable and, if for all possible models (different assignments) it returns False, then the formula is unsatisfiable. For instance, in $3-$coloring problem, instead of running all nodes and its neighbours, we attribute colors to nodes and verify if this configuration is a solution. If not, we distribute the colors over the nodes differently. Later we will encounter a class even higher in this category: NP-hard; a problem NP-complete class can be reduced to a problem in this class in polynomial time playing these two a similar role as NP completeness to NP. A NP-hard problem is not exclusively a decision problem.

For our analysis we still require another class and it is called #P. This class includes a set of problems which returns the total number of possible solutions to a problem, this is, counting problems not decision. An example is the permanent and in the next chapter we will introduce the hafnian which has the same nature. One can look at this class in the following way: given a P or NP problem, instead of finding a solution, count how many different solutions there are. This implies that most probably problems in this class are harder to solve than P or NP.

For this part we don't need to get in a lot of detail but as was mentioned in several papers including Brod (2021) that there's also a #P-complete subclass of problems similar to NP-complete to NP which every problem in this counting category can be reduced to and it has been proved by Valiant (1979) that the permanent is a #P-complete problem. This is particularly important, as an efficient algorithm for the permanent would entail efficient algorithms for all other #P problems. In the same article by Brod (2021), the author compares the calculation of determinant to permanent to see where their complexity stands. The conclusion is that the determinant can be efficiently calculated because of Gaussian elimination and this lowers the complexity from #P to P. The same can not be done with the permanent and this is ultimately the reason why Boson Sampling devices, whose probability amplitudes are given in terms of permanents, are hard to simulate classically.

A problem is considered easy to compute if it runs in time that is at most a polynomial with the input (any structure coded in bits representing either vertices or nodes of graph or arrays etc). Some different run times are presented and ordered in table 1.

3.4.2 Complexity for Boson Sampling

The analysis and model studied by Aaronson and Arkhipov (2013) showed the complexity and quantum advantage for boson sampling and here we show how the two implementations (quantum and classical) are related. We will study now the complexity of calculating all

Class	Name
$\mathcal{O}(1)$	Constant
$\mathcal{O}(\log N)$	Logarithmic
$\mathcal{O}(N)$	Linear
$\mathcal{O}(N \times \log N)$	Quasi-linear
$\mathcal{O}(N^2)$	Quadratic
$\mathcal{O}(N^c), c > 1$	Polynomial
$\mathcal{O}(c^N), c > 1$	Exponential

Table 1: Run time for algorithms; a program with complexity $\mathcal{O}(g)$ limited by the function g.

probability amplitudes (eq. 50), each of which is given by a permanent (eq. 48). Depending on the method to adopt, the complexity is very different for computing the permanent with our direct naive algorithm or with Ryser formula but either way it is important to note that the size of the input matrix $U_{S,T}$ is the number of photons $N = \sum_i^m s_i$ and not the number of modes of the system. In the case of quantum algorithm, the complexity falls mainly into the interferometer decomposition. The algorithm for implementing this decomposition falls into solving a linear system (translation for finding $T_{m,n}$ that nulls certain elements) and a cycle to run the diagonals. This leads to a complexity proportional to N^3 which is polynomial and as such can be efficiently done by a classical computer.

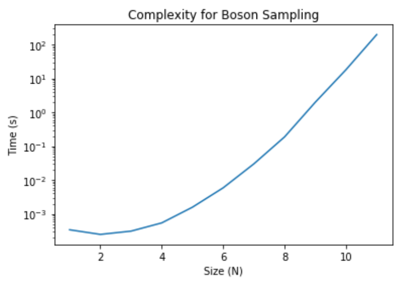
In the original paper for this model Aaronson and Arkhipov (2013), the authors consider a "standard initial state" $|1_k\rangle$ that consists of one photon in the first k modes (the remaining are all zero) without any loss of generality:

$$|1_k\rangle := |1, ..., 1, 0\rangle \,.$$

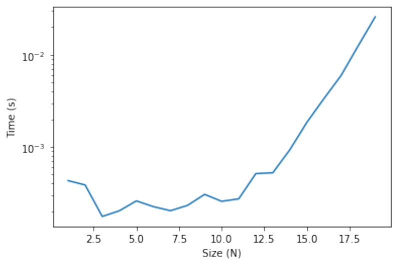
For our main code of classical implementation via formula in Eq. 48, to test the complexity of the program we test for an input state $|1_k\rangle$ as discussed followed by calculating the probability of a single state. Since the complexity grows mainly with the permanent, calculating the probability for all possible output combinations equals solving the permanent of a matrix with same dimensions; so the time to solve in the latter is a multiplication of the one obtained times $B_{m,n}$. In figure 12 are the plots for our complexity tests. All the tests and simulations presented where processed in a computer with the following specifications:

```
Caption            : Intel64 Family 6 Model 142 Stepping 10
Processor          : Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
Cores              : 4
Logical Processors: 8
Max Clock Speed    : 1992 Mhz
RAM                : 8GB
L1 cache           : 256KB
```

```
L2 cache            : 1.0MB
L3 cache            : 8.0MB
```



(a)



(b)

Figure 12: Study of complexity for boson sampling code; in sub-figure 12a was tested the function probC for the same input and output states given by $|1_N\rangle$ with N equal to dimension of a Haar-random matrix U which equals a state made out of 1's in all input modes (a feasible state in physical apparatus) and calculating the probability of the same state in the output. In sub-figure 12b we altered our main function probC in the permanent to use the more efficient algorithm mentioned by Ryser. Both sub-figures are in a logarithmic scale.

This evolution at an exponential increasing rate leads us to predict the permanent of higher order matrices, for instance 60 lines/columns, would take over 3000 years to complete. To obtain this estimative, we extrapolated results from calculating the permanent from sizes 1 to 27 with the function for the fit being the following given that we are using Ryser algorithm from the library The Walrus:

```
def fit_BS(n,c):
    return c*n*2**n
```

Listing 3.2: fit_BS function to extrapolate results.

The are several algorithms proposed to lower the complexity as already mentioned that take in account properties of matrices. In Valiant (1979) was studied the computational complexity of the permanent of matrices with only 0 or 1 entries and concluded it was still #P-complete. Other two examples in this category are Gurvits's algorithm Aaronson and Hance (2014) that can solve the permanent efficiently (in polynomial time) for matrices with both real and positive numbers and Glynn's formula for repeated rows or columns Glynn (2013). Another proposal was present in Clifford and Clifford (2017) that lowers the time complexity from Eq. 50 to:

$$\mathcal{O}\left(\text{poly}(m,n) + (n2^n)\right)$$

Their algorithm does weak simulation without requiring strong simulation first (as opposed to our proposal); to generate a sample (or event) we need approximately to calculate two permanents and this follows the same theoretical probability distribution as it is required.

This too small and simplified presentation of the algorithm is enough to see that the classical approach is still exponential but at a much smaller rate forcing quantum implementation to reach a much higher level to show quantum supremacy.

3.5 experimental challenges

Besides the computational complexity, there's another part to consider: the physical implementation. As we all know, classical electronic computers are a mature technology. However, for quantum computers there are several points like photon loss, decoherence and others that make implementation difficult so let us consider them too. A first detail that is strongly related to computational complexity is photon indistinguishability. Distinguishable particles can be considered as classical ones: they can be distinguished by different times of arrival, polarizations, or frequency (spectrum). If this is the case, the system can efficiently be simulated because the absence of superposition leads to the absence of the phases in unitary matrix U because, in fact, the change on elements lead to a probability for those photons in input i and output j to be related to $|U_{i,j}|^2$, i.e positive number; the permanent is now of a positive real matrix and we stated there is an efficient algorithm for matrices of this type by Aaronson and Hance (2014). A second detail is state preparation; photon number states are relatively easy to understand and they are the basic states of the quantum theory of light but they are less easy to generate experimentally. This is an important point in which this model becomes difficult to implement and control because it is physically hard to produce a single photon[1] for scalable solutions. Due to nonlinear optical processes being probabilistic, they cannot simultaneously achieve a high probability of producing a photon and a high single-photon fidelity. Not only producing $|1\rangle$ is not deterministic, the typical source emit the zero-photon term with highest probability and emit higher order terms with exponentially decreasing probability.

The major concerns about optical elements are focused on single photon preparation, photon loss, network errors and detection imperfections. This increased the quantum complexity in such way that this model did not offer advantage: computationally, classical computers run Boson Sampling in the #P class and quantum computers with only in a polynomial complexity but physical implementation for the first is almost automatic but for the latter increased a lot. Fortunately, more recently in Wang et al. (2019) a Boson Sampling device was reported with 20 input photons in a linear interferometer with 60 modes. The authors were able to detect up to 14 output photons with single-photon detectors.[2] Nevertheless, their computer is non-reconfigurable so it is yet not ready for practical applications.

The difficulties and the non-universality of this model motivated to consider other similar approaches such as Gaussian Boson Sampling due to deterministically preparation of input

1 Usually used spontaneous parametric down-conversion (SPDC) or quantum dots.

2 There are detectors that can count a few number photons, but they are based on superconductor wires. More common are APDs that saturate a single photon, do not detect more than one photon per mode.

states (that we will study in next chapter) and Scattershot Boson Sampling Bentivegna et al. (2015).

One thing that we need to work with in all implementations of quantum computation is that the outcome is probabilistic so it is not easy to interpret the samples. Another aspect that already was mentioned is they have errors associated such as losses of coherence, particles absorbed, preparation of states (either with a different value or gates with slightly different parameters and many others). Besides, when analysing the complexity of programs in earlier sections we considered classes where validation of a solution is efficiently verified but verifying outputs from a quantum computer might not be efficient. A straightforward reason is that we do not have previous access to how it should behave and, as we are well aware, strong simulation via classical computer is not the way to do it. Some solutions might be verified efficiently, for instance factoring, but that is not true for all problems. For all these reasons, we need to find reasonable validation criteria for near-term experiments. This is, if we use the intended model, how do we know if the outcome is as expected and a valid sample? There are several methods for verification and validation of quantum samples.

A way to do this is with Bayesian analysis of results. There are several proposals for this, we will describe one example next. Produce a number x of samples with the quantum computer; if we can calculate the probability for these outputs, we do it and conclude the following: if a quantum computer is working as predicted, the probability of returned events tend to be higher than other distributions other than the one sampled from. This is, if there is too much noise, it will generate outputs that are either uniform or weakly correlated with the output we would have from an ideal device. Otherwise, it will output events with higher probability leading to higher probability values of samples on average.

Let us translate this idea into a simple validation procedure, a statistical test called the likelihood ratio test Cover and Thomas (2005), that we can apply to our simulated samples. To process the results we can give for instance 3 hypotheses that may describe the process generating the samples:

A It is generating samples according to the theoretical probability distribution as intended;

B The device is dominated by noise and output probability is uniformly distributed;

C Input photons are distinguishable .

To validate actual samples we need to generate data according to a distribution (here either A, B or C), then calculate these three probability output given the method we are comparing to and observe the likelihood ratio to see which hypothesis it approximates. Given samples from a GBS device, if the answer is the first one (A), we can start to gain some trust in it. So now we make some tests: generate samples as above in particular conditions and compare.

The probability distribution for uniform samples is simply $1/_{\text{number of possible states}}$ for each state. Now we need to calculate the probabilities for distinguishable photons. From Moylett et al. (2019) we have that for this case the same arguments (input S, output T and unitary U) but the formula 48 is rearranged to:

$$\Pr(S \to T) = \frac{\text{perm}(|U_{S,T}|^2)}{\prod_i s_i! \prod_j t_j!} \tag{51}$$

This is, the probability for our arguments instead of being the permanent of the amplitudes for the photonic many-particle paths, we make the permanent of the squared absolute values of amplitudes. As consequence of distinguishability for input photons, the system can be efficiently simulated as referred in complexity section with Gurvits's algorithm since $|U_{i,j}|^2$ for each i and j leads to a positive real number (the corresponding probability). This is implemented in the following function probCdisting.

```
def probCdisting(U, S, T):
    factorial=1
    for i in T: factorial=factorial*math.factorial(i)

    rows = [i for sublist in [[idx] * j for idx, j in enumerate(T)]
                    for i in sublist]
    columns = [i for sublist in [[idx] * j for idx, j in enumerate(S)]
                    for i in sublist]
    prob = perm(np.abs(U[:, columns][rows])**2)/factorial
    return prob
```

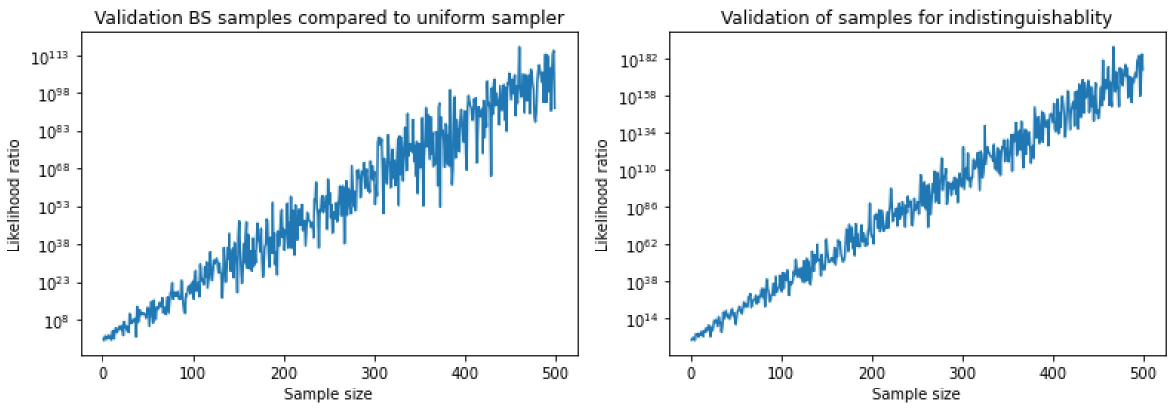
Listing 3.3: Function to calculate the probability of a BS device with distinguishable input photons.

The function in code B.14 returns the probabilities for BS samples for theoretical prediction and for the uniformly distributed outcomes [*probBSampler*, *probUniSelec*] in order to compare which of A or B from our hypothesis is more likely. This comparison that we are about to make is called the likelihood ratio test; it compares two statistical models based on their likelihoods. To perform this test we do the following: with the results for the two probabilities returned we divide these factors (*probBSampler*/*probUniSelec*) and take the product of each fraction obtained. In analogy with this function, we have the next function distingVal() that returns probabilities for BS model with indistinguishable and distinguishable photons in [*distBS*, *distDisting*] respectively given samples from BS distribution in Eq.48.

```
def distingVal(U, f, times):
    m=len(U); stateIn=states_giverIn(m,f)
    samplesBS=randomGeneratorB(U, f, times)

    distBS=np.array( [probC(U, stateIn, T) for T in samplesBS] )
    distDisting=np.array([probCdisting(U,stateIn,T) for T in samplesBS] )
    return [distBS, distDisting]
```

Listing 3.4: Generation of samples from BS distribution to test validation; returns probabilities for distinguishable and indistinguishable distributions.

In figure 13 we have the resulting values for these two comparisons: hypothesis A vs B in sub-figure 13a and hypothesis A vs C in sub-figure 13b.



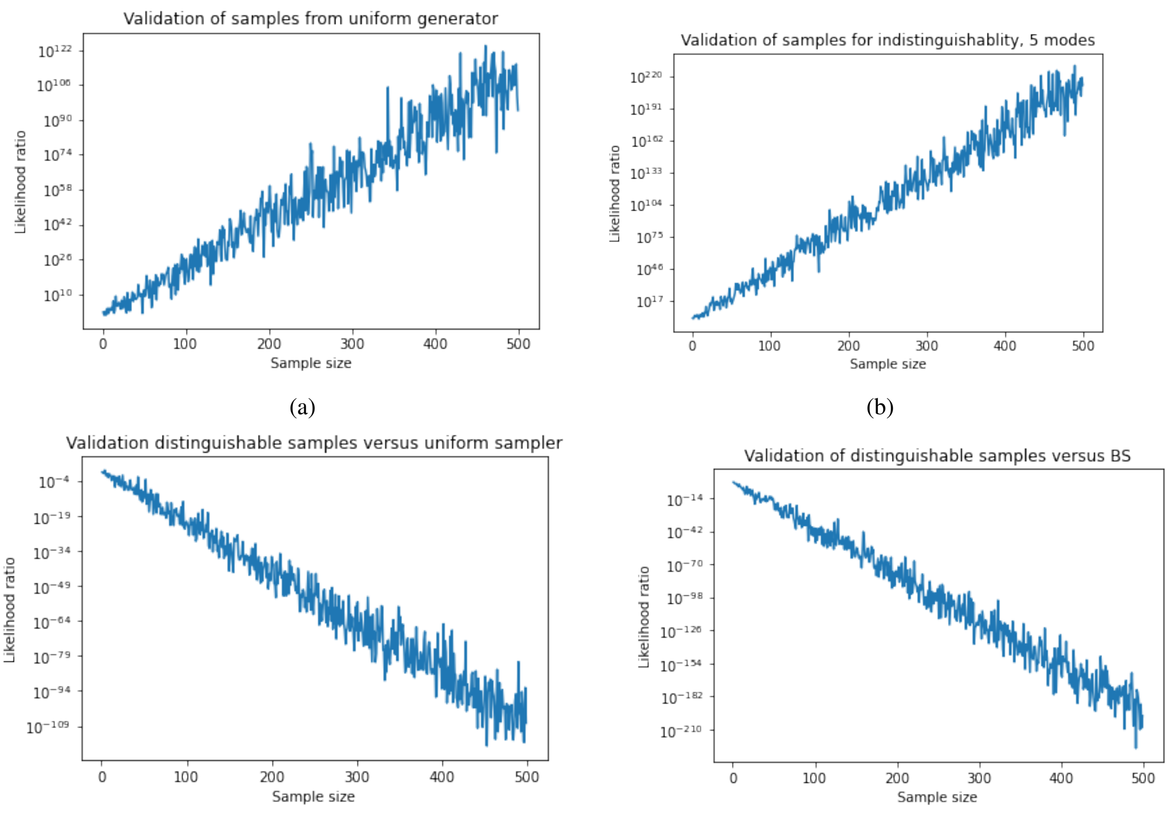
(a) (b)

Figure 13: Validation of Boson Sampling samples in comparison to uniform sampler in sub-figure 13a and the model with distinguishable input photons in sub-figure 13b. Both plots show the evolution of probability of success in a logarithmic scale for number of samples in range 1 to 500 for a Haar-random unitary of size 4 and input state $|1,1,1,1\rangle$.

From this figure we can conclude that if our device samples like the ideal model, we can trust it is sampling correctly since the increase in the fraction product is exponential and for only 500 samples we have a result in order of 10^{113} in the comparison against a device that outputs uniform noise, and 10^{182} against a device whose input photons are perfectly distinguishable.

On the same note, we can generate samples from a device dominated by noise i.e. where samples are uniformly distributed and test validation once more. In this case, instead of generating samples with our generators (rejection sampling or brute force) we simply generate the possible states and randomly chose from this array a given number of times for the sample size. The function unifGenVal in code B.15 does this selection and returns $[probUniSelec, probBS]$. In sub-figure 14a we can trust that those samples are much more likely to be from a uniform sampler than a BS device reaching a product of probability fraction in order of 10^{122}. We see that validation is in the same order as tested previously with BS generator in sub-figure 13a. Also, in the same figure, in sub-figure 14b we compare once more BS model with distinguishable photon (similar to sub-figure 13b) but for more modes and observe that the confidence in our model grows - the ratio is of order 10^{220} as opposed to previous 10^{182}.

The three hypotheses here considered are extremes: ideal boson sampling and a all-noise device but we have a considerable difference between them, the rate at which the trust in our



(a) (b) (c) (d)

Figure 14: Testing samples for a uniform sampler testing hypothesis A versus B in sub-figure 14a once again for a Haar-random unitary of size 4 and input state $|1,1,1,1\rangle$; in sub-figure 14b we tested samples from BS distribution for higher modes (a Haar-random unitary of size 5 and input state $|1,1,1,1,1\rangle$) to test distinguishability. Last two sub-figures show the likelihood ratio test from a BS sampler with distinguishable input photons compared to uniform distribution $probUniSelec/probDisting$ in 14c and to an ideal BS $probBS/probDisting$ in 14d.

samples goes is exponential and we only tested for a small device; for 4 mode interferometer we detected exponential rate quite visible and for 5 mode interferometer we saw an evolution with even less noise. To complete these validation experiments, we also generated samples according to distinguishable photons distribution; it was compared hypothesis C to A and B calculating the likelihood ratio of probabilities $\text{prob}(B)/\text{prob}(C)$ and $\text{prob}(A)/\text{prob}(C)$. We can observe the opposite of what was seen so far: tested from a probability distribution and the product of their probability values was in the denominator. The resulting plots (in sub-figures 14d and 14c) is still in logarithmic scale having a mirror effect compared to tests above.

All quantum circuits that show quantum supremacy used some form of validation to gain trust in their samples and all of them require some probability calculation via a classical computer. As we are well aware now, this has a clear limitation due to the size that it is possible to simulate and, in order to limit the size, a possible trick is to prepare a system arbitrarily as before and change some settings afterwards; these changes are in this case the two mode gates connecting the middle part of the circuit where now the particles in the first half do not interact with the second and a classical computer can simulate them separately, i.e., calculate permanents with half the size which is a huge difference considering exponential growth of complexity.

A last consideration we should reflect on is then the rate of sample generation by an actual BS device since for each test we need more than a single sample. Sampling rates decrease with the depth of the interferometers, due to photon losses, and the quality of the output is directly affected by less-than-perfect photonic indistinguishability. For the experiment by Wang et al. (2019) in generation of 20 photons and detection of 14 they generated samples at rate ~ 6 per hour. Other proposals detect at different rates and for Gaussian Boson Sampling by Arrazola et al. (2021) (next chapter: the experiment is similar with linear interferometry and photon detection), photon number event rates were: for detection of 4-photon events, an average event rate of 10000 events per second; 10-photon events at 270 events per second; and 19-photon events at 0.3 events per second. There is a variant of Boson Sampling called Scattershot Boson Sampling that increases the speed of quantum devices and Bentivegna et al. (2015) made this experiment and estimated a runtime of $\sim 10^7$ to 10^8 seconds for a 2000-event with Boson Sampling that with their experiment, under same conditions, is about 50 seconds.

3.6 numerical experiments

Having only results for classical simulation, we can process those samples and compare one another (rejection sampling and brute force) and with the theoretical prediction. To do so, we first implemented a function in Appendix B.16 to treat this data: it generates samples from randomGeneratorA and randomGeneratorB mentioned in the Simulation section and calculates the three distributions. Distribution from samples is a simple fraction of samples for a given state divided by the total number of samples. To better visualization we eliminate states whose probability in all three distributions are equal to zero (or near zero around a value $\epsilon = 0.00001$). It returns the following list:

$$dists = [statesp, distT, distA, distB, occurrencesA, occurrencesB]$$

where $statesp$ are the remaining states with non-zero probability, $distT$, $distA$ and $distB$ are the three distributions and $occurrencesA$ and $occurrencesB$ are the occurrences from sampling methods. Last but not least, we organize and plot the results. To do a complete plot, we calculate the error bars for the samples using this last two ($occurrencesA$ and $occurrencesB$) that are not essential for plotting distributions alone but are important factor for sampling the error associated. In Appendix code B.17 is the function that plots the error graphics from previous function and the next two lines of code represent the calculation of error for each sampling distribution computing the standard deviation for binomial distribution i.e. the error is given by:

$$\text{error} = \frac{\sqrt{Np(1-p)}}{N}$$

```
    yerrA=np.sqrt(dists[4]*(1-dists[2]))/samples
    yerrB=np.sqrt(dists[5]*(1-dists[3]))/samples
```

Listing 3.5: Error bar calculation for methods A and B.

Now, running the code from both implementations (classical and quantum), it is expected to run according to the model and this allows to simultaneously show properties of the physical construction, i.e. show how the elements of the interferometer behave with different inputs since the model is based on physical elements. The number of samples to test from is chosen to obtain a reasonable error bar where our simulation can be compared to theory. Also, from rates mentioned in implemented experiments, they seem likely to allow them. Since for now we have restrictions to use Xanadu's quantum computer, in this section we will only proceed with our code instead of Strawberry Field's simulators.

A property to easily observe is bosonic coalescence which for the case of two indistinguishable photons incident on a balanced beam splitter interferometer, they stick together in the output of the device. This is predicted by quantum theory as a consequence of the bosonic nature of light as we saw in previous chapter the so-called Hong-Ou-Mandel effect Hong et al. (1987) and can be generalized to several photon numbers on the input and multiple modes. To verify these properties let us consider two different types of matrices that describe interferometers having some degree of symmetry and which have recently been implemented experimentally: Sylvester in Viggianiello et al. (2018) and Fourier.

To begin with, let us define Sylvester matrices: for dimension $N = 2^p$ with p an integer are defined as:

$$H(2^p) = \begin{pmatrix} H(2^{p-1}) & H(2^{p-1}) \\ H(2^{p-1}) & -H(2^{p-1}) \end{pmatrix} \text{ where } H(2^0) = H(1) = 1$$

Fourier matrices (in accordance to Discrete Fourier Transform DFT) is defined as:

$$F_{i,j} = \left(\frac{\omega^{jk}}{\sqrt{N}} \right) \text{ where } \omega = \exp\left(-\frac{2\pi i}{N} \right) \text{ and } \{j,k\} = 0, ..., N-1$$

The unitaries associated to each Sylvester matrix are obtained by the simple rescaling factor $1/\sqrt{N}$. For matrices with dimension 2×2 and 4×4 we have Eqs. 52 and 53 respectively for Fourier and Sylvester constructions:

$$U_H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \quad U_F = \frac{1}{\sqrt{2}}\begin{pmatrix}\exp(-\pi i 0\cdot 0) & \exp(-\pi i 0\cdot 1)\\ \exp(-\pi i 1\cdot 0) & \exp(-\pi i 1\cdot 1)\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \tag{52}$$

$$U_H = \frac{1}{\sqrt{4}}\begin{pmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\end{pmatrix} \quad U_F = \frac{1}{\sqrt{4}}\begin{pmatrix}1 & 1 & 1 & 1\\ 1 & -i & -1 & i\\ 1 & -1 & 1 & -1\\ 1 & i & -1 & -i\end{pmatrix} \tag{53}$$

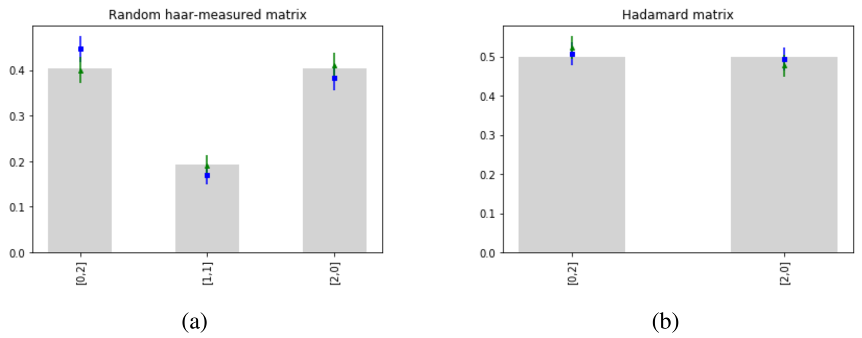
For a two mode system, these two constructions are the same and the output probability distribution is the same for both however, for higher dimensions they differ - the modulus of each matrix element is always constant, but the phases differ in the two constructions; in particular, H only has ± 1 phases, contrary to the Fourier in d dimensions, where the phases are d^{th} roots of identity. This seemingly small difference leads to a distinct pattern in the output where we verify a suppression of states: combinations of input and output Fock states which result in zero transition probability due to destructive interference are considered suppressed. Sylvester matrices present more suppression then Fourier ones (the fraction of suppressed states is higher) and this can and will be verified with our code. Consider the states to test (output) for each example with the following construction: for N photons, the sum of number of photons per mode is N and each n_i is less or equal to N ($\sum_i n_i = N$ and $n_i \leq N$ where $i = 0, ..., m$).

For two modes, matrices above are equivalent to the balanced beam splitter which is defined with $\theta = \pi/4$ and $\phi = 0$ in our previous characterization. In figure 15 we have the probability distribution from theoretical model and sampling from two methods for this case ($m = 2$) for two different matrices: random and DFT/Sylvester.

There are several points to note from the figure. First of all, the sampling distributions and theoretical ones are similar and the error bars shows us a good measure of error for our samples. In the case of random matrix in sub-figure 15a probability of $|0,2\rangle$ and $|2,0\rangle$ are equal and the same can be observed in the following. Sub-figure 15b shows a zero probability for output $|1,1\rangle$, with one photon per output mode. This last property was introduced and developed in last chapter discussing entanglement, in section 2.4.4.

For three modes there is no Sylvester matrix, as it is only defined for dimension which are powers of 2. On the contrary, the Fourier matrix in dimension 3 is:

$$U_t = \frac{1}{\sqrt{3}}\begin{pmatrix}1 & 1 & 1\\ 1 & e^{i2\pi/3} & e^{i4\pi/3}\\ 1 & e^{i4\pi/3} & e^{i8\pi/3}\end{pmatrix}$$



(a) (b)

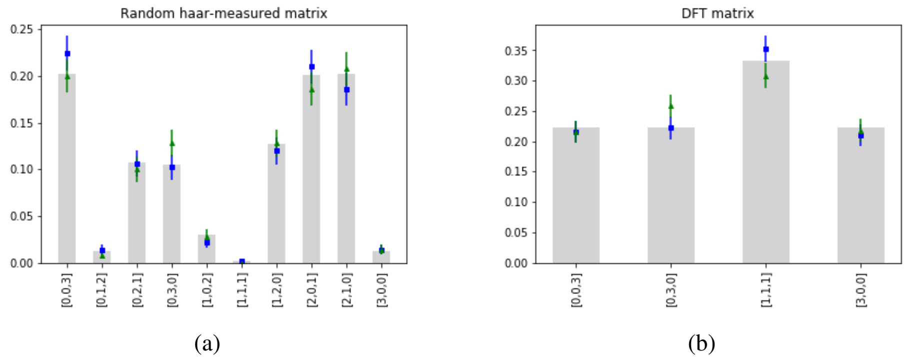
Figure 15: Distributions obtained for random matrix for the uniform Haar ensemble in 15a and DFT/-Sylvester matrix in 15b. Grey bars represent the theoretical distribution obtained from the model equation in 48, blue squares represent a distribution from samples via randomGeneratorA and green triangles represent a distribution from samples via randomGeneratorB. For each test, 300 samples were generated.

Making the calculations for this matrix and denoting $|\{n,0,0\}\rangle$ as the superposition state of n photons distributed with equal probability in each mode, for input $|1,1,1\rangle$ the output state is:

$$|1,1,1\rangle \xrightarrow{U_t} \sqrt{\frac{1}{3}}\,|1,1,1\rangle + \sqrt{\frac{2}{3}}\,|\{3,0,0\}\rangle$$

This matrix is called a tritter which is the equivalent of balanced beam splitter for 3 modes.

Now, we draw a Haar-random unitary of 3 modes, and compute the exact probabilities for different outputs, using a $|1,1,1\rangle$ input. Again, we perform a sampling of a fixed number of events, using the two previous generators. The results are presented in figure 16.

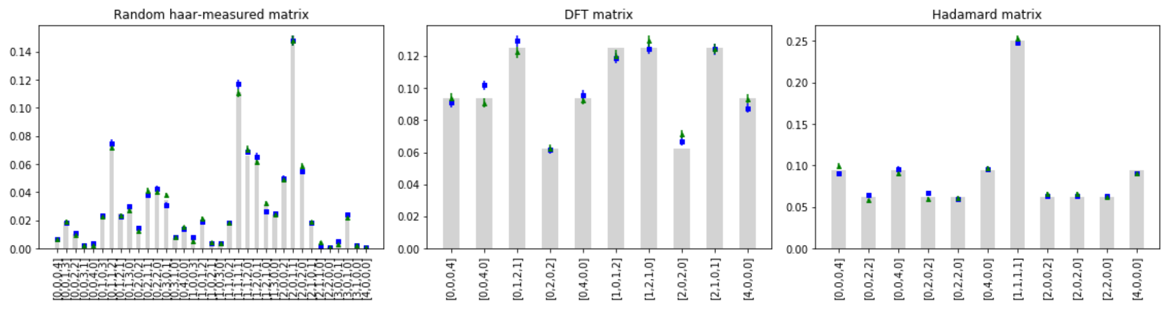


(a) (b)

Figure 16: Distributions of 3-mode systems for the uniform Haar ensemble in 16a and DFT matrix in 16b. Grey bars represent the theoretical distribution obtained from the model equation in 48, blue squares represent a distribution from samples via randomGeneratorA and green triangles represent a distribution from samples via randomGeneratorB. For each test, it was generated 500 samples.

Here comparing the two sub-figures 16a and 16b we can see the number of states without zero probability decreased noticeably as expected. For sub-figure 16a the distribution is random and for sub-figure 16b follows the prediction above: output $|1,1,1\rangle$ had probability around $1/3 \approx 0.33$ and each $|\{3,0,0\}\rangle$ state had probability around $^{2/3}/_{3} \approx 0.22$.

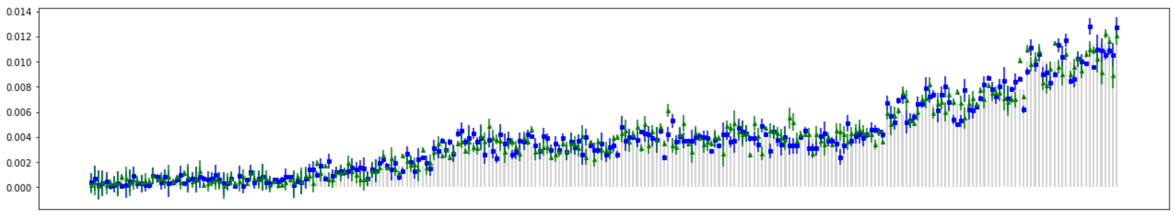
For four modes we can repeat the simulation of distributions for Haar-random, DFT and Sylvester matrices where now we expect to see a difference between the last two as in Eqs. 52 and 53 as opposed to two modes where they were the same and the sampling corresponded. So what do we expect this difference in matrices to be translated to? For two modes we have a matrix composed with only 1's but for four modes we have the imaginary number i substituting some of those 1's. The implication is the phase in these modes which will lead to distinct interference of particles. In figure 17 we show the probabilities of different outputs, again for an input of one photon per mode.



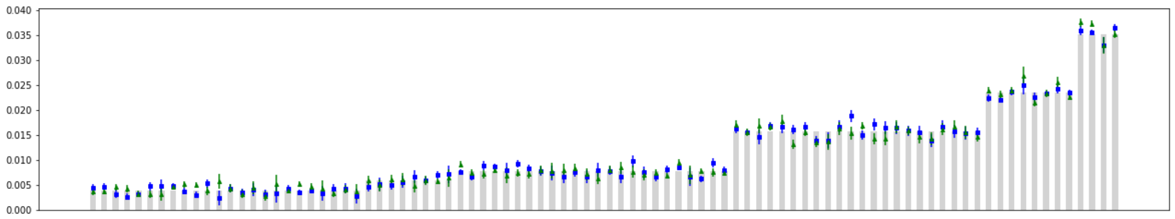
(a) (b) (c)

Figure 17: Distribution plots for 4-mode matrices. Distributions obtained for random Haar-generated matrix in 17a, DFT matrix in 17b and Hadamard matrix in 17c. Grey bars represent the theoretical distribution obtained from the model equation in 48, blue squares represent a distribution from samples via randomGeneratorA and green triangles represent a distribution from samples via randomGeneratorB. For each test, it was generated 10000 samples.

Recall that for better visualization, states with zero probability (both theoretical or from sampling) or a value close by a small error where discarded. This allows to see a significant difference between sub-figure 17a and sub-figures 17b and 17c. Between all these sub-figures we can observe in sequence the suppression for the three matrices as interferometers: the first has little or no suppression (using coefficient $B_{m,n}$ for the number of output combinations we have 35 which is the exact number of states in sub-figure of Haar-random matrix so there is no suppression), the second has a substantial difference visible directly from the number of states and in the last one the number of states allowed are similar to DFT-matrix but has a state with a significantly higher probability. One can observe this last difference for the same number of particles but more modes which we have an example in figure 18 for $m = 8$ and $n = 4$. There we can clearly see that many more input/outputs are suppressed for the case of Sylvester, than for the case of Fourier. In this particular simulation, the number of states with non-zero probability for Fourier matrix was 264 states and for Sylvester matrix was only 90. We can see once more the steps in the evolution of probabilities but suppression leads to smaller error bars in Sylvester matrices.



(a)



(b)

Figure 18: Output probability distributions for 8 modes with 4 input photons for DFT matrix in 18a and Sylvester matrix in 18b. Each sampling distribution was calculated with 10000 samples.

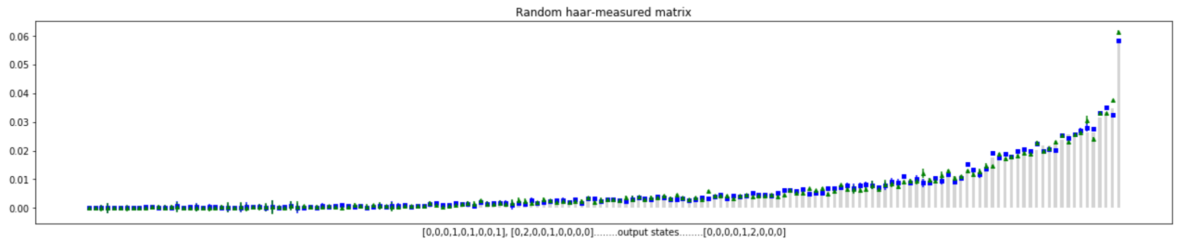
At last, we realized a simulation closer to the ones mentioned for physical experiments where number of photons was related to modes via $n^2 \leq m$. The results for 3 photons in 9 modes are plotted in figure 19. For such simulation, we expect arbitrary Haar-random interferometers to have distribution that is uniform just like in figure 19a where the ordered probabilities for each output form kind of a continuum. In contrast, in figure 19b the ordered probabilities have steps due to the symmetry present in Fourier interferometers.

3.7 summary

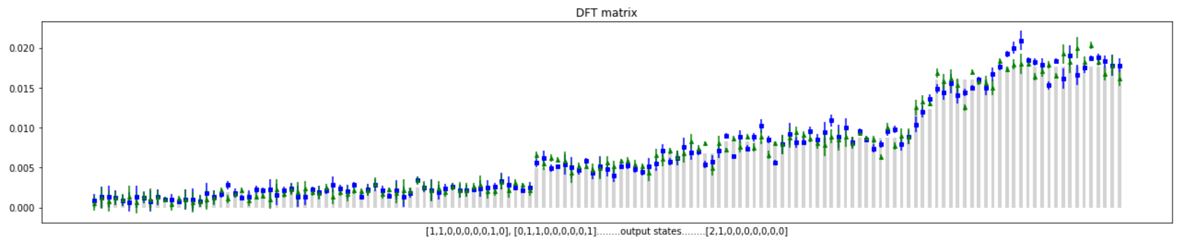
In this chapter our goal was to describe a model for quantum computation based on a linear-optical network with input Fock states called Boson Sampling. To achieve that goal we started by describing the model mathematically; we saw how an interferometer acts on our input, simplified some formulas for the particular case we consider. We concluded that such network followed by photo-detection is given by equation 48 that gives us the next probability equation:

$$\Pr(S \to T) = \frac{|\text{perm}(U_{S,T})|^2}{\prod_i s_i! \prod_j t_j!}$$

From here we know that the probability prediction is intimately related to the calculation of the permanent so for our next step we implemented a code to solve this problem - used classical function probBS - can took a more careful look at this new function by analysing the complexity. Our code (transparent to the formula) corresponds to $\mathcal{O}(N!N)$ operations which is intractable. However, there are other algorithms to perform this and a good example is Ryser's that has a run-time $\mathcal{O}(N2^N)$, still exponential but at a much lower rate. To fully



(a)



(b)

Figure 19: Output probability distributions for theoretical results and two sets of samples with randomGeneratorA and randomGeneratorB for 9 modes with 3 input photons for random Haar-measured matrix in 19a and DFT matrix in 19b. Each sampling distribution was calculated with 10000 samples.

simulate the model we have to consider all possible output states for a given input so the code for this probabilities has time steps defined in equation 50.

In order to generate samples and predict this model behaviour we pursued two approaches. The first consists on rejection sampling method where we generate samples and exclude those outside our interest (see sub-figure 11a) and in the latter we generate random numbers where each corresponds to a sample (see sub-figure 11b) that follows our model probability function. Note that both are based on strong simulation. Then we took advantage of code writing to implement a quantum algorithm by Strawberry Fields language, a code that also simulates the quantum system but can be used to code actual quantum chips by a small change of backend to a Boson Sampling device when access is provided. With our code complete, it is time to analyse its complexity in more detail. For the model, we showed that running this code is a complex problem belonging to a classic of counting problems #P-complete. This started to be a good starting point to demonstrate quantum supremacy since this is a hard problem to solve with a classical computer but a quantum approach would to it naturally which leads us to discuss the experimental challenges.

Since there are many algorithms to solve the permanent, some are considered efficient (polynomial run time) for particular conditions such as for a real, positive-valued matrices in the argument of the permanent function as a consequence of photon distinguishability. Other challenge is state preparation where single photon generation proved to be hard for scalable solutions. Detection of photons is also hard if we intended to return von Neumann measurement instead of avalanche. Along with these there is also photon loss, network errors and detection imperfections that we expect to improve in a near future. Concerning validation of quantum outputs, i.e. actual samples, we made some tests to show how to validate samples from a quantum device. We compared samples from a Boson Sampling distribution to a system with distinguished input photons and also to a uniform sampler (device fully made out of error). For validation proposes it was also generated samples for a system with distinguishable input photons to compare to BS and dominated by noise systems.

Finally, we made some numerical experiments for Haar-random, Sylvester and DFT matrices where the last two have some symmetries. Here we were able to see some aspects previously mentioned for instance entanglement and quantum suppression.

Given this model, a Boson Sampling device is an amazing choice to show quantum supremacy but their applications are still limited due to physical implementations and theoretical development.

4 GAUSSIAN BOSON SAMPLING

Following the analysis in chapter 2, we can conclude that Gaussian states are states whose phase-space description is simple and that is precisely why one can easily simulate them in a classical computer; initialization is based on describing the displacement vector and covariance matrix. Taking those states and applying transformations that preserve their Gaussian nature (Gaussian transformations), it is possible to track the evolution efficiently on a classical computer because it is given by matrix multiplication, and the dimension of the matrices D scale as $2 \cdot N$ with N number of qumodes (Bartlett et al. (2002)). However, if one introduces a non-Gaussian element either to the input states, the operations in the middle or in measurements, the computation is no longer simple/trivial. That is what was introduced in last chapter 3 where the non-Gaussian elements were the state initialization (Fock states) and measurements. Gaussian Boson Sampling (GBS) is a linear-optical computation model where the non-Gaussian ingredient appears only at the final measurement step. This model uses Gaussian states with photodetection which is the best proposal for computation advantage. Besides demonstrations of quantum computational advantage, GBS also serves as a building block for non-Gaussian approaches such as GKP states with efficient non-Guassian input preparation (with subtraction of photons in state prepartion) that we will mention in Applications chapter.

4.1 gbs model

Up to now we have reviewed different quantum states of light and their representation, and used Fock states and linear-optical evolution to describe the Boson Sampling model. The next step in constructing our GBS model is equivalent to what was studied with boson sampling: with the previous background on Gaussian states and using the phase space methods as above, it was shown in Hamilton et al. (2017) that if we generate N single mode squeezed states and input them in a N-mode linear interferometer (built out of beam splitters and phase-shifters as described previously), the probability of measuring a photon in each mode (Fock measurement with $n_j = \{0, 1\}$) is given by:

$$\Pr(\overline{n}) = \frac{1}{\overline{n}!\sqrt{|\sigma_Q|}} \mathrm{Haf}(A_S) \tag{54}$$

In this equation we have the following elements:

- $|\sigma_Q| = \det(\sigma_Q)$ is the determinant of σ_Q;
- σ_Q is defined has

  $$\sigma_Q = \sigma + \frac{\mathbb{1}_{2N}}{2}$$

  where σ is the covariance matrix of the system to be measured/observed and $2N$ indicates the dimension of the identity;
- $\mathrm{Haf}(A_S)$ is the hafnian of the matrix A_S and the explicit formula is given by:

  $$\mathrm{Haf}(A_S) = \sum_{\mu' \in PMP} \prod_{j=1}^{N} A_{S_{\mu'(2j-1),\mu'(2j)}}$$

  where PMP stands for perfect matching permutations for a general graph Björklund et al. (2019);
- A_S arises from a matrix A that is given by:

  $$A = \begin{pmatrix} 0 & \mathbb{1}_N \\ \mathbb{1}_N & 0 \end{pmatrix} (\mathbb{1}_{2N} - \sigma_Q^{-1}) \tag{55}$$

  Similar to the matrix $U_{S,T}$ in Boson Sampling, A_S is also a sub-matrix obtained from A ($A_S \subseteq A$) where the choice for the terms are according to the detected photons $S = (S_1, ..., S_m)$ in the measurement: the rows and columns i and $i+m$ are repeated s_i times (if $s_i = 0$, the corresponding rows and columns are deleted).

Note that the initialization is using only squeezed input states - a continuous variable state as opposed to Boson Sampling that used photon number states - and for physical implementation both squeezed and coherent are created from lasers but squeezing requires an extra effort so why use this instead? There is a simple reason why this is the case: this model (along with Boson Sampling) was developed to show quantum supremacy and coherent states alone can be simulated efficiently, that is, in polynomial time, on a classical computer. Moreover, coherent states are described by a displacement in the phase space, one that can easily be added to the calculations and has no effect on the covariance matrix which plays a major role in the model. Also, these states do not contribute to entanglement and any other informationally relevant properties. Schuld et al. (2020) studied the effect of coherent states in the output since they change the mean of the states while leaving the covariance matrix intact. For this reasons and considering that states can be displaced at any time in the system without changing the fundamental properties along with the fact that squeezed states already result in a difficult simulation using photodetection, the model needs only use squeezed states as inputs. Another consideration is the occupation number n_j; the authors created that model for $n_j = \{0, 1\}$ but developed for $n \geq 2$ in Kruse et al. (2019). Repeating the calculations done for 0 or 1 output photon per mode but for this case, we need to repeat the row and column for that mode j as many times as n_j (similarly to Boson Sampling). The resulting matrix A_S does not correspond to a valid covariance matrix so it is only used to determine the probability of higher order events in the hafnian expression. This combination of state preparation with squeezed gates to Gaussian operations and measurement with photo-detection is represented schematically in figure 20.

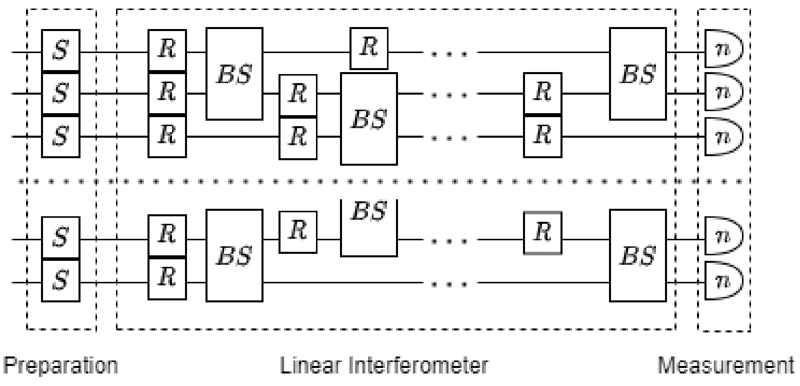


Figure 20: GBS arbitrary circuit; according to description, the preparation is obtained with single mode squeezing in each mode, then linear interferometry to process our states (here we have phase gates R and beam splitters BS according to interferometer decomposition as described in chapter 2), followed by photo-detection measurement of each mode.

Since this model in the original work (and the one we are using) is with specific initialization and interferometry, together with the characterization on previous chapters about Gaussian quantum information we can rewrite equation 54 in order to still have a generic one and make it more transparent to use considering the specific generation/creation of modes and the CV gates. Specifically, we want to use the code more easily to specify squeezing parameters and unitaries, as directly as possible.

First, all the states start in vacuum state so the covariance matrix is $\sigma = {}^1\!/_2\mathbb{1}_{2N}$ and the initialization relies on squeezing each mode by a value r (generic squeezing can be decomposed as we saw in squeezing with r and phase rotation by ϕ that the latter can be incorporated in the interferometer for simplification) which, based on equation 37, in the quadrature basis corresponds to the evolution:

$$S_S \sigma S_S^\dagger \text{ with } S_S = \begin{pmatrix} \bigoplus_{i=1}^N \cosh r_i & \bigoplus_{i=1}^N \sinh r_i \\ \bigoplus_{i=1}^N \sinh r_i & \bigoplus_{i=1}^N \cosh r_i \end{pmatrix}$$

The interferometer is made out of passive elements so the matrix describing it in quadrature basis is a diagonal one with the unitaries U and $U^\dagger$ just like what we saw in the boson sampling model. Here we use directly the equations for phase rotation 27 and beam splitter 30 for describing the evolution where the symplectic matrix for a general interferometer U is then $S_I = \begin{pmatrix} U & 0 \\ 0 & U^* \end{pmatrix}$. The covariance matrix is then:

$$\sigma = \frac{1}{2} S_I S_S \mathbb{1} S_S^\dagger S_I^\dagger = \frac{1}{2} \begin{pmatrix} U & 0 \\ 0 & U^* \end{pmatrix} S_S S_S^\dagger \begin{pmatrix} U^\dagger & 0 \\ 0 & U^\intercal \end{pmatrix} \tag{56}$$

We can start by calculating $|\sigma_Q|$ for one mode: one squeezed with parameter r and identity for the interferometer $U = 1$. Then σ_Q becomes:

$$\sigma_Q = \frac{1}{2} \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \cosh r & \sinh r \\ \sinh r & \cosh r \end{pmatrix}^2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right]$$

where we used $S_S = S_S^\dagger$ clearly obtained from the above. The identities multiplying are trivial, then proceed with the square S_S^2 and finally sum the last identity. Now, with the property of determinants of multiplication for constants ($|c \cdot A| = c^k |A|$ with k the dimension of A) we have the following:

$$|\sigma_Q| = \left(\frac{1}{2}\right)^2 \left| \begin{pmatrix} \cosh^2 r + \sinh^2 r + 1 & 2\cosh r \sinh r \\ 2\cosh r \sinh r & \cosh^2 r + \sinh^2 r + 1 \end{pmatrix} \right|$$

To finalise, use the next two hyperbolic relations:

$$\cosh^2 x + \sinh^2 x = 2\cosh^2 x - 1 \quad ; \quad \cosh^2 r - \sinh^2 r = 1$$

Then comes the simple result: $|\sigma_Q| = \cosh^2 r$. Generalizing this result, the phases for example would be canceled since the elements multiply on the right by a phase and on the left by the corresponding conjugated term; the phases that pass this direct calculation, are canceled when calculating the determinant. As the matrix is unitary, it preserves the modulus, so for m modes, $|\sigma_Q| = \cosh^{2m} r$ if all modes are squeezed with the same parameter. For different squeezing parameters, $\sqrt{|\sigma_Q|}$ in formula equals to:

$$\sqrt{|\sigma_Q|} = \prod_{i=1}^{m} \cosh r_i \tag{57}$$

Now, instead of calculating the covariance matrix, by matrix multiplication from function arguments and calculating the determinant we can simply apply the previous equation. This equality is also used in Strawberry Fields tutorials.

In the original paper of GBS, Hamilton et al. (2017) showed another equivalence. Given a covariance matrix as defined in equation 56 above, we can rewrite 55 as:

$$A = B \oplus B^* \text{ where } B = U(\oplus_{j=1}^{m} \tanh r_j)U^\intercal \tag{58}$$

turning equation 54 with these considerations into:

$$\Pr(\overline{n}) = \frac{1}{\overline{n}! \prod_{i=1}^{m} \cosh r_i} \left| \text{Haf}\left([U(\oplus_{j=1}^{m} \tanh r_j)U^\intercal]_S \right) \right|^2 \tag{59}$$

4.2 code implementations

To study the behaviour of GBS, we first write the code translating eq. 54. Similar to boson sampling, here we will first see the classical code implementation and then quantum code via Strawberry Fields platform.

4.2.1 Classical and quantum algorithms

Let us begin with the classical code then defining function probGBS() in algorithm 4.1 where the arguments (similar to probBS() in algorithm 1) are U and r that describe our system - interferometer and initialization with squeezing, respectively - and T is the output state to measure.

```
def probGBS(U, r, T):
    N=len(U)
    rows=[i for sublist in [[idx]*j for idx,j in enumerate(T)] for i in
        sublist]
    factorial=1
    for i in T: factorial=factorial*math.factorial(i)

    if type(r) is not type(0) and N>1:
        elements=[np.tanh(i) for i in r]
        tanhR=np.diag(elements)
        A=multi_dot([ U, tanhR, np.transpose(U) ])
        numerator=np.abs(thewalrus.hafnian(A[:,rows][rows]))**2
        coshR=np.cosh(r[0])
        for i in range(1,N):
            coshR=coshR*np.cosh(r[i])
        return numerator/(factorial*coshR)

    A=np.dot(U, np.transpose(U))*np.tanh(r)
    numerator=np.abs( thewalrus.hafnian(A[:,rows][rows]) )**2
    return numerator/(factorial*np.cosh(r)**N)
```

Listing 4.1: GBS Python Algorithm

From eq. 54 there are a few aspects we always need to calculate and those appear in the first lines of code (from line 2 to 6): in rows there are the selected rows and columns to chose from A to obtain A_S calculated with two lists comprehension and then is calculated the factorial of the output ($n!$). Now we go into specifics; r is a parameter describing the squeezing parameters for each mode thus an array and the function enters the if condition if r is an array as opposed to a single integer value applied to all of them. In this first case, we calculate the matrix $A = U \cdot \tanh r \cdot U^{\intercal}$ in line 10 and calculate the hafnian of this matrix with selected rows and columns all in line 11. Lines 12 to 14 correspond to calculation of eq. 57 and finally return the probability value in line 15. Alternatively, when r is a number, that same value is applied to all qumodes and the last 3 lines of code do the simplified equivalent work for this case.

Next we write the quantum algorithm with the help of function probGBSQ() which takes similar arguments as probGBS() (equivalent to probBSQ()). The difference is that it does not require a particular output state to measure so default is none and otherwise the function returns a specific value to state in that argument. This difference is due to the intrinsic and generic functions that run the program.

```
def probGBSQ(U, squeeze, measure_states='None'):
    modes=len(squeeze)
    gbs = sf.Program(modes)

    with gbs.context as q:
        for i in range(modes):
            if squeeze[i]==0: continue
            else: Sgate( squeeze[i] ) | q[i]
        Interferometer(U) | q

    eng = sf.Engine(backend="gaussian")
    results = eng.run(gbs)

    if measure_states=='None': return results.state
    for i in measure_states:
        prob = results.state.fock_prob(i)
        print("|{}>: {}".format("".join(str(j) for j in i), prob))

    return results.state
```

Listing 4.2: GBS Strawberry Fields Algorithm.

Once more, we would like to have the program clearly allow both for calling a classical simulation routine, or a real quantum device. This appears in the code as: state preparation in lines 6 to 8 right after the program initialization in lines 2 and 3; then we call the Gaussian backend to run this code and output the results in case that measure_states is 'None'. If not, we use the fock_prob function to print the specific result(s). As we know, this language is for quantum algorithms and implementation which can be achieved by changing the engine in line 12 to obtain actual real samples.

4.3 quantum advantage

When analyzing Boson Sampling in section 3.4.2, we investigated the computational complexity of classical simulation - in that case, it was due to the calculation of permanents. We can easily spot a correspondence between equations 48 for BS and 54 for GBS; both calculate the factorial of output and/or input distribution and then evaluate matrix functions whose computation scales exponentially with matrix size - the permanent in the case of Boson Sampling, and the hafnian in the case of Gaussian Boson Sampling. Some aspects remain the same, for instance, interferometer decomposition which means the quantum program complexity does not increase so let us analyse the parts were they are distinct.

In computational terms, we can assume that GBS is at least as hard to simulate as BS because we have the following equality:

$$\mathrm{Haf}\begin{pmatrix} 0 & H \\ H^{\intercal} & 0 \end{pmatrix} = \mathrm{Perm}(H)$$

which means that the hafnian is a particular case of calculating the permanent and hence one can always compute the permanent of a $n \times n$ matrix computing the hafnian of a symmetric $2n \times 2n$ matrix. Put simply, if the hafnian could be computed efficiently, so could the permanent. Thus, the computational cost of simulating a GBS model of interference of k photons is the same (up to a polynomial factor) as simulating a k-photon boson sampler. In figure 21 we have a plot of the run time of the Hafnian and we can see that it is exponential, which demonstrates resemblance with figure 12 for the permanent.

In paper by Gupt et al. (2020), GBS was benchmarked by running a classical code for threshold detectors on the Titan supercomputer. Many algorithms were proposed to lower the complexity of the programs by using threshold detectors instead of photon-number resolving ones which is a good approximation for the case where the probability of photon collision is low, i.e. particles distribute over different modes (for Boson Sampling it was sufficient to work with $n^2 < N$ and use Haar-random matrices). The time-complexity for their algorithm under these conditions is $\mathcal{O}(N^2 2^n)$ where N stands for number of modes and n number of clicks in detectors. This algorithm lowers the time complexity comparing to algorithms with photon-

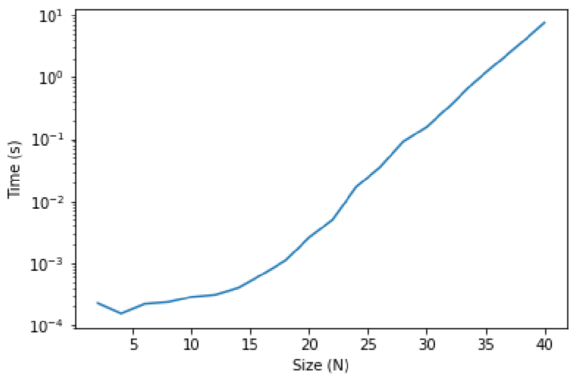


Figure 21: Run time of Hafnian for GBS complexity study using hafnian() function by The Walrus library (see code B.18). Hardware is the same as previous chapter. The figure represents the time it took to calculate the Hafnian for matrices with increasing size from 2 to 40 in a logarithmic scale for time.

number resolving detectors but, once more, it continues exponential and, as such, inefficient. They treat each measurement as a linear combination of Gaussian operators and this requires storing twice as many covariance matrices for each click detected hence the 2^n part associated to memory resources; thus, to compute all these matrices, run time also escalates exponentially.

4.3.1 Programmable nanophotonic chip

An important aspect to explore better is the physical implementation of such system; we studied the computational hardness, now we will address more closely the physical requirements. For boson sampling this was the motivation to focus in a slightly different model since state preparation was hard due to generation of single photons with high probability and fidelity leading to a huge problem specially for scalable solutions. Here, the only different feature in physical implementations is instead of Fock states we use squeezing gates to prepare squeezed states from vacuum. Before advancing, note that although state preparation was the main concern, switching these does not solve other problems not related to it directly such as photon losses in interferometer (absorption), photon distinguishability and detector errors. Exploring the covariance matrix σ or matrix A is the same since they are related as seen before. Starting from all states in vacuum followed by squeezing and linear interferometry, the matrix A can be decomposed as equation 58 where U is the unitary of the linear elements and $\lambda_i = \tanh r_i$ determines the squeezing parameters.

$$A = U\text{diag}(\lambda_1, \lambda_2, ..., \lambda_n)U^\intercal$$

In next section we will study the requirement of squeezing parameters for specific applications but for our purposes of quantum advantage in this chapter's numerical experiments we will use similar values to an implementation made by Zhong et al. (2021) and for the interferometer U we will use a random Haar-matrix. To do so, we created a function that generates random Gaussian values around a given mean value from function arguments that it is expected to be in a range of squeezing parameters to implement in experiments:

```
def squeezeDist(m, meanV):
    randVals=np.random.normal(meanV,0.01,m)
    if meanV-0.01<np.mean(randVals)<meanV+0.01:
        print(np.mean(randVals))
        return randVals
    else:
        return squeezeDist(m, meanV)
```

Listing 4.3: Generation of random squeezing values from a normal distribution around a value in function argument.

A valid use for this function is for example squeezeDist(9, 0.45) where 9 stands for the number of modes for our system and 0.45 the mean for the 9 squeezing parameters. This mean value will be the one used in the experiment by Zhong et al. (2021) and from the Xanadu's chip by Arrazola et al. (2021) we see they are reasonable values to work with. If we have $m = 9$ and $meanV = 0.55$, calculating the expected mean photon in output using equation 46 we have that $\overline{n} = \sinh(0.55)^2 \times 9 \approx 3.01$ hence we have a similar relation to $\overline{n}^2 = m$ which was one of the conditions in Boson Sampling for low collision of photons (this can also be deduced from statistical analysis) and consequently allowing calculations/simulation with 0 or 1 photons or the use of threshold detectors in physical implementations. For classical experiments this will become important to permit larger simulations.

The recent most robust GBS experiment is by the previously cited article by Zhong et al. (2021) - they prepared two-mode squeezed state as input states with 25 power sources and inject them into a 144-mode interferometer finalizing with 144 single-photon detectors. They detected up to 113 photon detection events and for these values, their quantum computer produces samples in a sampling rate approximately 10^{24} times faster than a brute-force classical supercomputer. The experiment here (similar to proposal by Wang et al. (2019)) is not able to program the interferometer leaving the controlled portion to the emission of squeezed photons. This work followed the implementation by Zhong et al. (2020) which performed experiments with 50 highly distinguishable input single-mode squeezed states, which were fed into a 100-mode ultralow-loss interferometer and finally sampled using 100 high-efficiency single-photon detectors. They observed up to 76 output photon-clicks, which yield an output state space dimension of $\sim 10^{30}$ and a sampling rate that is $\sim 10^{14}$ faster than using the state-of-the-art simulation strategy and supercomputers. The authors could sample from the GBS distribution in 200 seconds that they estimate these would take 2.5 billion years to calculate on China's TaihuLight supercomputer.

The paper by Arrazola et al. (2021) introduced is a small device with eight modes (states are initialized with four two-mode squeezing sources forwarded to two interferometers of size 4, one for each four-mode subspace). However, it is the first device based on GBS that is programmable, and open for use on the cloud. Also, in chapter 3 we referred to this device because of the sample rate that is higher than other proposals.

4.4 simulation

In the previous model with Boson Sampling, for each configuration, we could choose an input state - usually with the relation of number of photons n being much smaller than number of modes m - and then generate all possible output combinations for the number n, calculating the probability associated with each possible output state. Now, with a Gaussian Boson sampler, we will still make strong simulation with methods A and B (rejection sampling and brute force) but we have an extra consideration; the input is not defined by a fixed number n but rather squeezed states associated with a mean photon number $\overline{n}$ and a probability $P(2n)$ already predicted and given by:

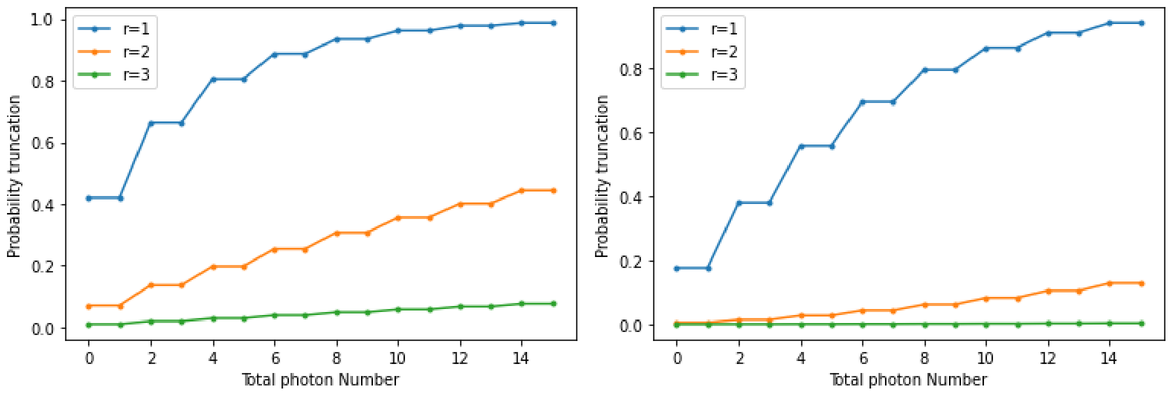
$$\overline{n} = \sinh^2(r); \qquad P(2n) = \operatorname{sech} r \frac{(2n)!}{(n!)^2 2^{2n}} (\tanh r)^{2n}$$

With this, we need to introduce an upper bound on the number of output photon numbers for which we will calculate the distribution since it is not possible to simulate all possible output states, as strictly speaking there is no upper bound on the number of photons generated. This is a consequence of using continuous-variable squeezed states, which are a superposition of Fock states with arbitrarily high occupation number, as can be seen in Eq. 19. In order to have a sensible choice for the cutoff photon number for simulation purposes, we observe how $P(2n)$ behaves and test the truncation number for different choices for this cutoff Fock state. Function $P(2n)$ was already shown in figure 9 when we introduced squeezed states. As we can see from the figure, the vacuum state is always the state corresponding to maximum probability, but the probability of higher occupation numbers increases as we increase the squeezing parameter. To see explicitly the effect of simulation with a truncation number it can be simply calculated with the function below sum_trunc():

```
def sum_trunc (U, r, f):
    inputs=states_giver( len(U) , f)
    return sum([probGBS(U, r, T) for T in inputs])
```

Listing 4.4: Function sum_trunc to test the Cumulative Distribution Function for specific input arguments.

As before, we have a linear interferometer with passive optical elements which does not change the total number of photons. This means the analysis of how the truncation affects the fraction of events detected is independent of the choice of interferometer, so for figure 22 we picked a Haar-random one to plot the cumulative distribution function for different truncation numbers; we chose to represent three different values for the squeezing parameter (r = 1,2,3). This implies verifying the behaviour of the Cumulative Distribution Function (CDF), defined as the total probability associated with the calculated events up to the chosen photon truncation. As expected, the CDF increases monotonically as we increase the photon number truncation. It is expected to approach 1 as the total photon number goes to infinity. As we can see, the most significant increments are in the lower photon numbers so if we increase the detection number for simulation we improve our results but at some point we do not gain any significant improvement while making the calculations unnecessarily harder.



(a) (b)

Figure 22: Cumulative Distribution Function (CDF) for different squeezing parameters (U Haar-random matrix, r = 2 and equal in every mode) for two modes in sub-figure 22a and for four modes is in sub-figure 22b for all output events corresponding to each choice of total photon number. Note that photons are emitted in pairs as seen theoretically in equation 19 so the CDF does not increase from any even integer to the following odd integer.

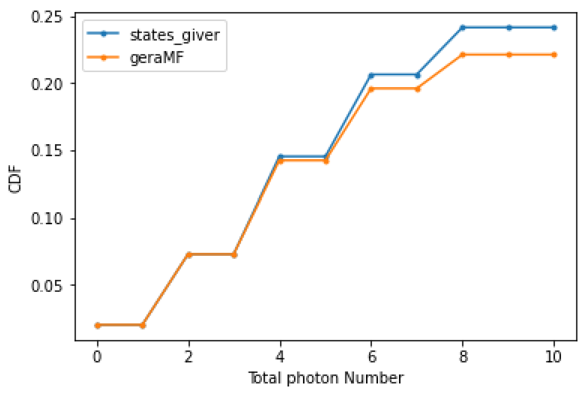
If we simulate GBS for a Haar-random matrix with all input squeezing parameters with $r = 1$ in a 9-mode interferometer the mean photon number is higher relative to the previous test (with mean squeezing value according to an actual physical experiment) in a programmable nanophotonic chip and, as consequence, the CDF is much smaller. The run time for detection of 6 output photons was $\approx$ 63.11 seconds and for 8 output photons was $\approx$ 159.87 seconds and the truncated CDF value was $\approx$ 0.19979 and $\approx$ 0.21067 respectively. We can conclude that the state with higher values for probability are simulated but there is a huge cut due to truncation. However, this is not much improved by increasing the cut-off value; the run time grows exponentially, with a slow-growing CDF. Since there is low collision in output samples, we now add another limit in truncation by imposing another restriction in output samples; after all, the complexity is mainly focused on the Hafnian but increasing exponentially the number of states to detect also has a major impact not only in time but also memory resources. We then write a different function using geraMF() to simplify and allow larger experiments:

```
def geraMF(m, f):
    sett = [[x] for x in range(2+1)]
    a=aux(m,f,sett); a.sort(key=lambda st : sum(st))
    return a

def aux(m,f,sett):
    if len(sett[0])==m:
        return [ l for l in sett if sum(l)<=m ]
    new = []
    for l in sett:
        for i in range(2+1):
            if sum(l)+i<=f:
                new = new + [l + [i]]
    return aux(m,f,new)
```

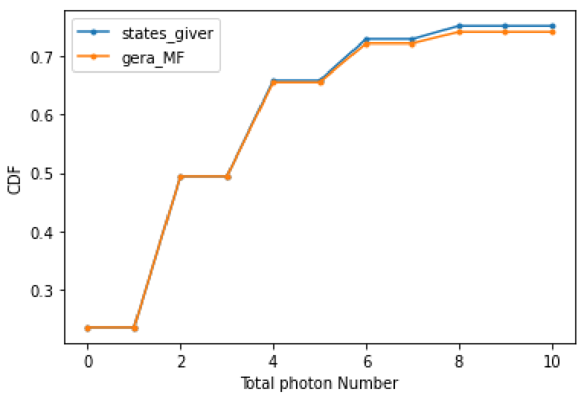
Listing 4.5: Second truncation method we propose to lower run time of output state generation.

m denotes the number of modes, f the total number of photons (summed over each qumode) and for here, each mode, we impose a maximum photon number of 2, a value defined in the second line of code 4.5. Comparing to previous state generation, given any m and $f \geq 0$, they can coincide (the returning array is the same) if we change this restriction presented but this function is more efficient. For $m = 16$ and $f = 2$, previous method took ≈ 53.557 seconds and geraMF() took ≈ 0.501 seconds. The difference in possible states is photons per mode where for $m = 6$ and $f = 6$ states such as $[0, 0, 1, 0, 3, 1]$, $[4, 0, 0, 2, 0, 0]$ and $[4, 0, 1, 1, 0, 0]$ are not generated now. For $m = 10$ and $f = 4$, first method generates an array with length 1001 and takes ≈ 3.735 seconds to do it while this generates 891 states in ≈ 0.0160 seconds. For the purposes of this chapter - for small simulations and to show a quantum advantage - we will continue with the first and more complete state generation; in the next chapter this motivation will become clearer/more explicit and even necessary to run some codes.

In figure 23 we have the CDF as before but now with a goal to see the effect of the second truncation where we compare CDF for (1) a constant input squeezing parameter and (2) a lower mean value that approaches values from physical implementations. In sub-figure 23a we use arbitrary conditions with $r = 1$ and U Haar-random and comparing to sub-figure 23b we have a squeezing distribution around a fixed mean value and the same interferometer U; the first not only has higher cut in output samples i.e. lower CDF but the two state generation have a more significant difference. With this we can infer that when conditions permit it - mean photon number can be related to an approximate relation $\overline{n}^2 = m$ and interferometer does not have a particular symmetry that favours collision - the second technique should be used to allow computational simulation.



(a)



(b)

Figure 23: CDF for the two processes of state output generation (codes B.19 and B.20). In sub-figure 23a we have this function for a constant input of $r = 1$ in every qumode where in sub-figure 23b we have the plot for a system with the same interferometer but the squeezing parameters are the return from squeezeDist$(9, 0.5)$. It was also tested with a constant squeezing in every mode with the mean value $r = 0.5$ and the result was equal (by inspection) to sub-figure 23b.

Note that the CDF is independent of the interferometer so we could think of discarding it for this test or use identity. It is correct to assume this but for the second truncation we can not use this tactic since the output occupation numbers do depend on the interferometer design, photons will not distribute and it leads to huge truncation by limiting n_i to 2 for qumodes with higher squeezing parameters.

These two aspects impose changes in our simulators from Boson Sampling. Recall method A that uses the rejection sampling method; to choose random values of probability we randomly selected values in range $[0, 1]$ while selecting corresponding states to attribute those values. This led to a high rejection rate because we could not lower the upper limit because we did not have a clue of how much we could cut but most probabilities had small values. Here, from the analysis of function $P(2n)$ we know that the the lowest occupation number corresponds to the maximum value for probability. The method is subsequently adjusted to this new characteristic lowering the rejection rate. The code is mainly altered with the next lines of code where line 1 calculates the maximum value and this is used as upper bound in uniform selection in line 4 for variable y.

```
  limit_prob=probGBS( U,squeeze , np.zeros(len(squeeze), dtype=int) )
2 photonOutput=states_giverOut(len(U), limit_photons)
  x=np.random.randint(0, len(photonOutput), times)
4 y=np.random.uniform(0, limit_prob, times)
```

Listing 4.6: Adjustments in rejection sampling method to GBS simulator.

For method B each random value corresponded to a valid sample so to include all possibilities we also randomly selected values in range $[0, 1]$. The limitation in photon number detector for computational purposes leads to a truncation (as we saw with function sum_trunc()) so our interval now is $[0, limit\_prob]$ too where limit_prob is equivalent to returned value from sum_trunc() which leads to next changes in code:

```
dist=np.array( [probGBS(U, squeeze, T) for T in photonOutput] );
limit_prob=sum(dist)
probs=np.cumsum(dist)
y=np.random.uniform(0, limit_prob, size=times); y=np.sort(y)
```

Listing 4.7: Main code of method B for sampling by adjusting to the correct probability interval.

As we see, both methods needed to be adapted to these conditions, and we did this by including a new default argument corresponding to maximum n to detect called $limit\_photons$ and the value assigned is 10. In this chapter we are missing numerical experiments and in this case for small experiments with squeezing operator in the limit we expect collision of photons in detection so the truncation number should be higher. However, for more realistic simulations (for applications as we will see) and many others, we do not expect collision and can even consider algorithms of single-photon detection as mentioned by Gupt et al. (2020) for instance.

4.4.1 Numerical Experiments

Now we are ready to simulate and test some numerical experiments for Gaussian Boson Sampling devices.

Returned samples from methods above have one small issue yet not solved. As we saw, the sum of probabilities with a truncation number have a limit (< 1) and we can not simulate in the full range so comparing the theoretical distribution to the sampling distribution one can not expect them to correspond. In order to have matching values and have them on equal footing, we will divide the theoretical distribution by the truncation from sum_trunc().

For two modes we can do a similar approach or a parallel as in Boson Sampling generating samples for a Haar-random matrix and with a balanced beam splitter both with same input squeezing parameter. Also for a two-mode interferometer, we can implement two-mode squeezed gate which given the decomposition in equation 42 corresponds to using once more the balanced beam splitter but opposite squeezing parameters, i.e. $r_1 = -r_2$. In figure 24 we have the distributions for different conditions over two modes. See codes B.23, B.24 and B.25.

A first aspect we observe is the reduction of states with an associated, non-null probability and even that in all three sub-figures we have the highest probability state associated with

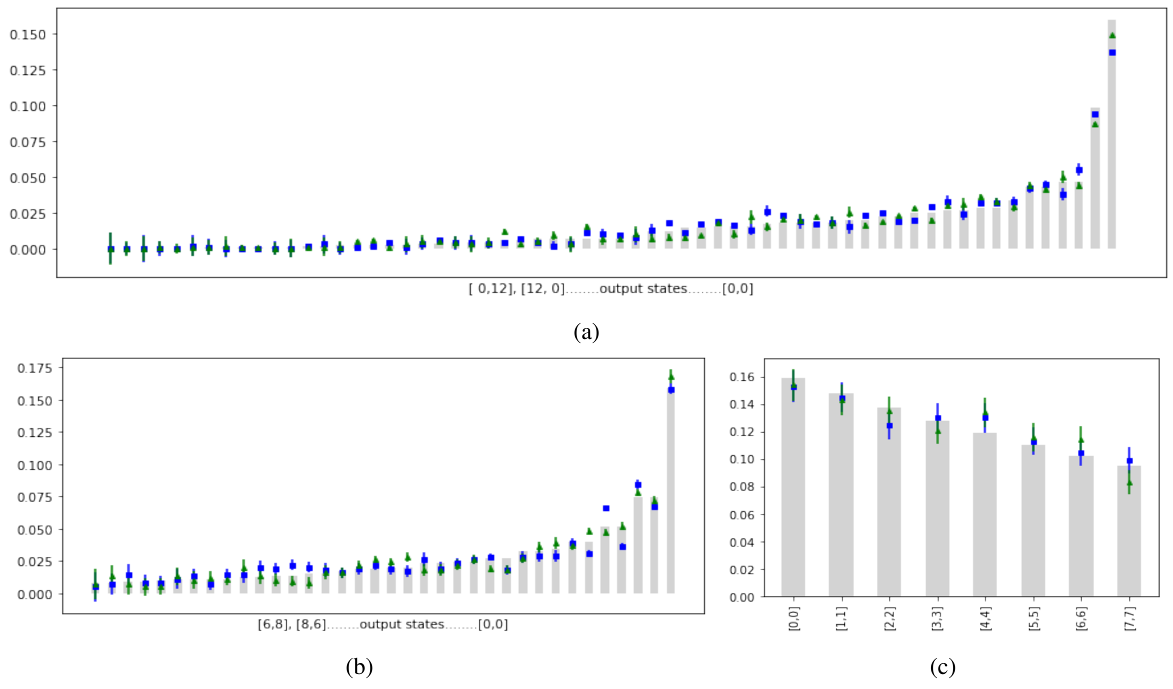
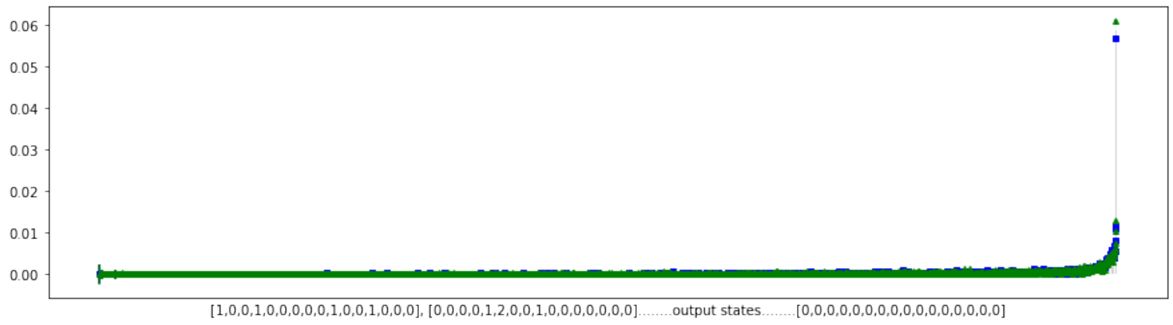


Figure 24: Distributions obtained for 2 modes with the uniform Haar ensemble in 24a, DFT matrix in 24b and two-mode squeezed gate in 24c. Vertical axis represent the normalized probabilities with rescaling of theoretical values. Input squeezing parameters for the first two cases were $[2, 2]$ and for the latter was $[2, -2]$. Grey bars represent the theoretical distribution obtained from the model equation in 54, blue squares represent a distribution from samples via sampleGeneratorA and green triangles represent a distribution from samples via sampleGeneratorB. For each test, it was generated 1000 samples with $limit\_photons = 15$. It was calculated the truncation number for each returning approximately 0.4435 and the number of states represented (with non-null probability) was 62, 36 and 8 respectively.

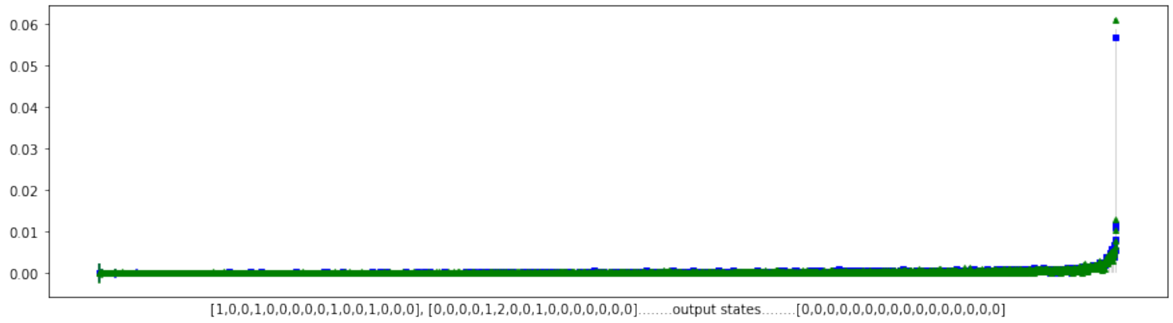
the vacuum $[0, 0]$. From input states with $r_1 = r_2 = 2$ followed by a balanced beam splitter we notice a reduction in diversity of states comparing to Haar-random matrix as should be expected in light of previous conclusions of interference in DFT matrices. Even so, the major difference was observed in the data presented in the last sub-figure 24c where we only got 8 states. Notice that the mean photon number for the two qumodes according to equation 46 is $2 \cdot \sinh^2 2 \approx 26.308$. The first sub-figure 24a is a random distribution and even though the limit is 15 photons, the times we ran the simulation the state $[15, 0]$ or $[0, 15]$ hardly appeared which shows the low necessity for these extreme truncation values. Also, experimentally it is used many modes with only a few photons per mode to simplify the detection process because we are simulating a subset with computational boundaries but physical experiments have some too. In the second sub-figure 24b it is easier to verify that the states are emitted in pairs as envisaged by equation 19 and for the third sub-figure 24c we can observe the impact of switching to two-mode squeezed vacuum states as predicted in equation 41.

Another important aspect is that considering these constraints of two modes and upper bound 15, the cardinality of possible outcomes is 136. However, for the random Haar-generated matrix we only have 62 represented and running the code several times we did not obtain simulations with more than 70 which means that even though we are restricting the system to allow computational simulation, we did not obtain outputs with significant probability for the highest occupation states.

For a more realistic case to study impact of truncation, allow us to consider an arbitrary example for the conditions described in section 4.3.1 and Haar-random unitaries for interferometer once more. Here we are interested in studying the output truncation for simulation since number of possible states increases considerably with the upper bound. We are simulating a system with $m = 16$ modes, mean squeezing value 0.47 and total maximum number of output photons 4 which is a reasonable value since $16 \cdot \sinh^2 0.47 \approx 16 \times 3.8024 \approx 4$. In figure 25 we can see the resulting plots for a constant squeezing parameter versus the above mean value. The plot itself is not very informative other than the clear step to vacuum with highest value and the last two states represented are the same which indicates that even with different inputs, the states with higher probability are focused in the same ones eliminating states with zero probability those with more collision and higher total occupation.



(a)



(b)

Figure 25: GBS simulation of 16 modes with output generation with function states_giver() with limit of output photon number 4. Vertical axis represent normalized probabilities. In sub-figure 25a the input is $r = 1$ and constant for all qumodes. For 25b the input has a mean squeezing value $meanVal = 0.47$. Both were tested for an Haar-random interferometer.

An important aspect to note is the effect of truncation in our tests using function sum_trunc(): for the case of sub-figure 25a we obtained ≈ 0.0164998, for sub-figure 25b ≈ 0.5619297 and for a last test using similar conditions as sub-figure 25b but with the second truncation (geraMF()) we obtained ≈ 0.5611493. The abrupt difference between the first two is due to mean photon number since we tested with limit 4 and the former has $\overline{n} = \sinh^2(1) \times 16 \approx 22.098$ and the latter around 4 photons. The difference the last two is only in the fourth decimal place which leads us to conclude that we have motivation enough to use this latter truncation over the original one.

From these tests, we have the vacuum state with highest probability value by far in all of them but this is a state that usually is not the most relevant one for many applications and often we will be looking for outcomes in a certain range. To select this it is implemented the function postselect() that given samples and bounds for upper and lower limit, it returnes the filtered samples:

```
def postselect(samples, min_count, max_count):
    return [s for s in samples if min_count <= sum(s) <= max_count]
```

With the numerical experiments above we can conclude that for Haar-random matrices the output state displays reduced collisions and we do not need to simulate for much higher values than 2 photons per mode and for real cases we can limit the upperbound for some applications since they stand outside of our interest area.

4.5 summary

Motivated by the experimental challenges of Boson Sampling, in this chapter we studied the Gaussian Boson Sampling problem by Hamilton et al. (2017). With squeezed vacuum states as inputs, for a linear interferometer with photon detection in the end in each mode, the output probability of pattern $\overline{n}$ is given by equation 54 where we have:

$$\Pr(\overline{n}) = \frac{1}{\overline{n}!\sqrt{|\sigma_Q|}}\text{Haf}(A_S) \quad \text{where } \sigma_Q = \sigma + \frac{\mathbb{1}_{2N}}{2} \text{ and } A = \begin{pmatrix} 0 & \mathbb{1}_N \\ \mathbb{1}_N & 0 \end{pmatrix} (\mathbb{1}_{2N} - \sigma_Q^{-1})$$

where Haf is the Hafnian, a mathematical function is the same family as the permanent and the determinant. S indicates the output where for each S_i and m the number of modes, the rows and columns i and $i + m$ of A are repeated s_i times. The implementation is represented schematically in figure 20. In order to a clearer code given the conditions of the interferometer and input being restricted to squeezed states, it was made a mathematical adjustment to simplify the code. We rewrote equation 54 as:

$$\Pr(\overline{n}) = \frac{1}{\overline{n}! \prod_{i=1}^{N} \cosh r_i} \mathrm{Haf}(A_S)$$

With theory presented we proceeded to writing the algorithms: first we dealt with the classical approach, making use of Strawberry Fields function to optimize the calculation of the Hafnian and then present the quantum algorithm that can be used to simulate as the classical one or to run in the device by Xanadu company Arrazola et al. (2021). There we were ready to analyse the complexity of GBS. The relation between Hafnian and permanent is useful since if an efficient algorithm could be found to compute the first, the later could be too and the complexity of current formula by Strawberry Fields was verified (figure 21). In what concerns experimental challenges, for interferometer and detection the implementation is the same as Boson Sampling but state preparation is easier (our initial motivation). To study current implementations and be conscious of the limitations, we explored two proposals of photonic chips constructed by the groups Zhong et al. (2021) and Arrazola et al. (2021). Considering squeezing values from a normal distribution around a mean value ≈ 0.5 and are shown to be doable but higher values can also be achieved.

With the algorithm to calculate equation 54 and the physical implementation discussed, we then made a few adjustments to randomGeneratorA and randomGeneratorB; here we must consider truncation since squeezed states have no upper bound on the possible observed number of photons. A good estimate can be done by studying the mean photon value and probability of detection to limit the output states to generate and test. This limitation is required since classical algorithms can not test up to infinity and this helps making classical simulation more productive. To improve this we created two functions for output generation; both have the same arguments (number of modes m and total number of photons f) but return different arrays. The first is states_giver() that returns an array where each position represents a state of size m, each mode is in range 0 to f and the sum over all modes is equal or lower to f. The second called geraMF() is motivated by low collision events; it also returns an array for a system of m modes and total photon number equal to f but each mode is limited to 0, 1 and 2 photons. For higher dimensions this shows an enormous impact.

Finally, we made some numerical experiments and tested the previous assumptions for several modes.

5

APPLICATIONS

The first model presented, Boson Sampling, is a non-universal model and although it was important for the development of the field, here we will be mainly focusing on GBS since the physical implementation is more doable. It has been shown that Gaussian Boson Sampling has applications in quantum chemistry, graph theory and optimization Bromley et al. (2020). The applications and problems by a programmable GBS device should start by analysing the matrix A in equation 54.

An important property about this matrix is being symmetric because many important and computationally-hard problems are described by graphs that can be represented by symmetric matrices. Graphs are particularly important since so many things can be represented by them. This indicates that some applications can take advantage of this property and those are the first ones we will review here.

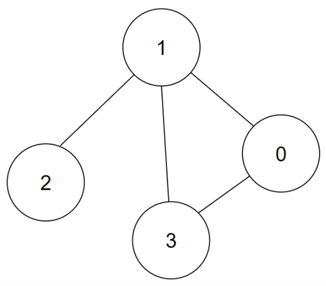
5.1 graph theory

Before jumping to applications to graphs, let us review what graphs are, and some basic properties.

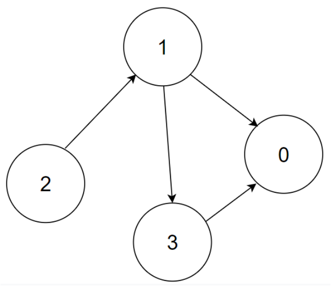
First, graph theory is the study of graphs which are mathematical structures composed with two sets: nodes and edges. The nodes (or vertices) work as points and the edges (or links) connect them. So a graph G is a pair $G = (V, E)$ where V is the set of nodes and E the set of edges. They are defined by the nodes they connect:

$$E \subseteq \{\{x, y\} | x, y \in V \text{ and } x \neq y\}$$

Secondly, an important feature in the edges make a clear distinction on the type of graph: directed where edges have a direction and undirected where edges are undirected links. This has an impact in their description and hence in the matrix A as will become clearer soon. More precisely, the illustration of this difference is in figure 26; edges in directed graphs are represented by arrows and, in the undirected case, a line or a two-way arrow.



(a) Undirected Graph



(b) Directed Graph

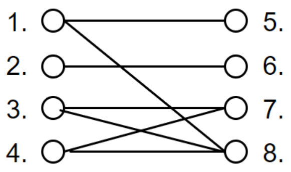
Figure 26

Both nodes and edges can be weighted and the interpretation of this weight depends on the problem. For instance, for a graph representing a map, weighted edges can translate the time, distance or cost to go from one city (node) to another.

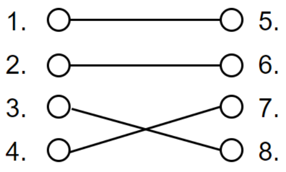
Having this basic definition of graphs, allow us to conclude with two concepts important for the next applications; Bipartite Graphs and Perfect Matchings:

- In a bipartite graph one can divide the nodes into two separated and independent groups, U and V for example, knowing that all edges connect nodes in U with nodes in V. These graphs enable the coloring of the graph with two colors without the same color in two adjacent nodes. Graphs in figure 26 are not bipartite graphs but in Fig. 27 we see two examples of bipartite graphs.

- A matching in a graph is a set of edges without common vertices. Take, for instance, the undirected graph from figure 26: two possible matchings are the sets $\{\{1,2\},\{0,3\}\}$ and $\{\{0,1\}\}$. Many computational problems in graph theory consider finding the maximal matching: when no other matching has as many elements as that matching, it is called maximum. For perfect matchings we consider matching nodes two by two in a subset $M \subseteq E$ such that if the cardinality of M equals cardinality of $V/2$, i.e. if $|M| = {}^{|V|}/_{2}$ then it is called perfect. In figure 27 we have one example. Note that we gave an example in a bipartite graph but it can be found in any random graph always noticing that perfect matching can only occur in a graph with even number of vertices.



(a) Arbitrary graph G

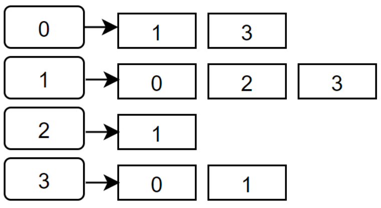


(b) Perfect matching of graph G

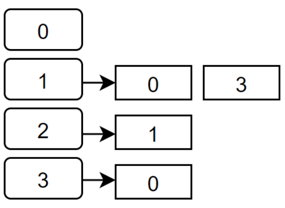
Figure 27

Lastly, graphs can be stored or represented in different ways and, taking figure 26 as an example, the most popular ones are the following:

1. Edge lists: lists/arrays with cardinality $|V|$ where the index is a node of the graph and to each is associated the set of nodes connected to the index node. Figure 28 has the two representations for our example.



(a) Edge list for undirected graph



(b) Edge list for directed graph

Figure 28

2. Adjacency matrix: $\text{GraphMat}_{i,j} = |V| \times |V|$ Boolean matrix (i.e., entries $\in \{0,1\}$) for unweighted graphs where:

$$\text{GraphMat} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

For an undirected graph, the adjacency matrix is symmetric and for a weighted graph each matrix entry represents the corresponding edge weight.

$$\text{Undirected graph: } \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}; \quad \text{Directed graph: } \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Counting perfect matchings

We will be interested in counting the number of perfect matchings of a graph, this is, the number of perfect matchings a graph can have. For an undirected bipartite graph G with $2n$ nodes, sides of partition U and V such that $|V| = |U| = n$ and the correspondent adjacency matrix $A_G \in \{0,1\}^{n \times n}$ then the number of perfect matchings, i.e., the number of different ways to make perfect matchings can be calculated via $\text{Per}(A_G)$, this is this number is given by the permanent of the bipartite adjacency matrix.

A more general problem is to calculate the number of perfect matchings for a graph with even number of nodes and this can be computed by the introduced function, the Hafnian that, as we can see, is strongly related to the permanent. The Hafnian and the computation of the number of perfect matchings will then be of most importance to study applications since they define the probability amplitudes associated with boson sampling experiments. Also, the hafnian of an odd sized matrix is defined to be zero, which reflects the fact that there are no perfect matchings for graphs with odd number of vertices.

Now that we have some basic facts about graphs, we can encode them in a GBS device using the adjacency matrices GraphMat as matrix A in equation 54. A Strawberry Fields implemented function will decompose the matrix for the graph along with the mean photon number $\overline{n}$ to the elements in equation 58 as we will see soon in more detail.

5.1.1 Coding an adjacency matrix into GBS

Now we intend to make a strong relation from the aspects described for graphs with the GBS model. Here we want to design a GBS experiment by choosing the interferometer unitary and squeezing parameters as presented in chapter 4, so that the output probabilities, as given by equation 54, are proportional to the hafnian of any adjacency matrix we may want to test. Sampling events with a probability that is proportional to the hafnian can be helpful since it translates some graph properties; for instance, counting perfect matchings (computing the hafnian) is related to finding dense subgraphs which is useful in graph-theoretical applications that we will study.

So let us first see how to encode these matrices into a GBS experiment, this is, how to start by A to obtain U and the appropriate r's. Directly from the original article for the GBS model, we have the following decomposition:

$$A = U\text{diag}(\lambda_1, \lambda_2, ..., \lambda_N)U^{\intercal}$$

where $\lambda_i = \tanh r_i$ is the hyperbolic tangent of the squeezing parameter applied to the vacuum state that is used as input of each mode and U is the interferometer matrix as usual. Also, from equation 46, we can rewrite the mean photon number in terms of λ_i and sum over all modes obtaining:

$$\overline{n} = \sum_{i=0}^{N} \frac{\lambda_i^2}{1-\lambda_i^2} \tag{60}$$

With this we supposedly only require $\overline{n}$ ($\overline{n}$ do not necessarily need to be distributed equally over all modes) and U to have A which is our adjacency matrix. The distribution of $\overline{n}$ over all modes is not necessarily uniform and this decomposition will be made by using a Strawberry Fields function based on Takagi factorization (see article by Cariolaro and Pierobon (2016)). Although all this seems quite simple and direct, some things might not add up and here are two examples why:

1. If we go back to Eq. 54, the output probability is proportional to the hafnian, i.e. the number of perfect matchings of the graph. Take for instance a complete graph with $2M$ nodes. Here all vertices have connection to all others and the number of perfect matchings grows with M as:

$$(2M-1)!!$$

   This number might be enormous and lead to probabilities outside of the probability interval $[0,1]$;

2. The adjacency matrix is obtained indirectly from the covariance matrix using σ_Q. Inverting the equation to find σ_Q we have:

$$\sigma_Q = (\mathbb{1}_{2N} - X_{2N}A)^{-1} - \frac{\mathbb{1}_{2N}}{2} \tag{61}$$

   where $X_{2N} = \begin{pmatrix} 0 & \mathbb{1}_N \\ \mathbb{1}_N & 0 \end{pmatrix}$. Since σ_Q has some restrictions (as mentioned above about commutation and others), A must be such that σ_Q remains valid.

In the article Brádler et al. (2018) the authors explain in detail how to adjust and overcome these two limitations in order to be able to encode arbitrary adjacency matrices. First, recall the conditions that σ has to satisfy: σ is a real, symmetric, positive-definite matrix in the case of quadrature basis in Eq. 9 and Hermitian, positive-definite matrix for the Heisenberg basis. This was already mentioned above and also presented Eq. 24. Here, let us analyse it in the Heisenberg basis. Since we are analysing in the Heisenberg basis, we have the following for this particular case in correspondence to 24 for the quadrature basis:

$$\sigma + \frac{1}{2}K \geq 0 \text{ with } K = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & -\mathbb{1} \end{pmatrix}$$

Now, starting from Eq. 61 we see the conditions A has to follow to lead to a valid σ_Q. A first step is: if a matrix is invertible and symmetric, then its inverse is also symmetric and from this, one can show that if $(\mathbb{1}_{2N} - X_{2N}A)^{-1}$ is symmetric, so is $\mathbb{1}_{2N} - X_{2N}A$. For this part to be symmetric, we can divide A in four sub-matrices (as already done for analysing physical behaviour) and they must satisfy:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad \Rightarrow \quad \begin{cases} A_{12} = A_{21} \\ A_{22} = A_{11}^{\intercal} \end{cases}$$

With this, we have the symmetry issue solved for these conditions.

Secondly, for $\sigma_Q > 0$ to hold, the eigenvalues λ_k of A must satisfy $|\lambda_k| < 1, \forall_k$. With this we now have to find a way to turn all eigenvalues to values smaller than one. One way to do so is to find λ_{max} (the largest eigenvalue of A) and choose a value c such that:

$$0 < c < 1/\lambda_{max}$$

and rescale A to cA which leads to a valid arbitrary covariance matrix as it's Hermitian and positive definite.

Turning this matrix into Hermitian and positive definite are the main points in encoding A, but this is not enough. To allow this encoding to work for arbitrary matrices there is still one small aspect to consider: back to the sub-matrices of A, let A_{11} and A_{12} commute and consider f_k and h_k their respective eigenvalues. Then they are related to the eigenvalues ν_k of our σ_Q via the next equation:

$$\nu_k = \frac{1}{2}\sqrt{\frac{(1+ch_k)^2 - c^2 f_k^2}{(1-ch_k)^2 - c^2 f_k^2}}$$

Since, to satisfy the commutation relations, we have $\nu_k \geq 1/2, \forall_k$ then $h_k \geq 0, \forall_k$ and hence $A_{12} \geq 0$. If $h_k = 0 \forall k$ then the covariance matrix is pure and A is consequently in a diagonal form. This leads to the doubling process where one can take the original adjacency matrix and use $A^{\oplus 2}$ such that $A^{\oplus 2} = A \oplus A$. Process doubling simplifies some aspects namely the requirement of $A_{12} \geq 0$ is already satisfied but at the cost of preparation of states is with single mode squeezers needing twice the number of qumodes. In particular applications for graph theory, this does not raise a major concern since most matrices already fulfill this condition because of two reasons: for unweighted graphs, values in A are either 0 or 1 (both in agreement with $A_{12} \geq 0$) and weighted graphs usually use values ≥ 0 representing a cost, length or many other positive numerical entities.

In short, we are interested in analyzing how these restrictions change the probability equation 54. A must be positive valued matrix and might need a rescaling factor c and equation 54 turns into:

$$\Pr(\overline{n}) = \frac{c^k}{\overline{n}!\sqrt{|\sigma_Q|}} \mathrm{Haf}(A_S) \text{ with } k = \sum_i n_i$$

If we use the doubling process either for dealing with negative entries or for physical implementation we also change the previous probability equation in the hafnian by considering $\mathrm{Haf}(A^{\oplus 2}) = \mathrm{Haf}A\mathrm{Haf}A$.

Last but not least, how to code or make this a computational and automatic task? One can use the Strawberry Fields library to obtain this and help getting some answers. If the choice is to code from the adjacency matrix A, we can use Strawberry Fields function sf.decompositions.takagi() or sf.decompositions.graph_embed(). The first does a direct decomposition of a matrix H into inteferometer unitary U and rl such that $H = U\mathrm{diag}(rl)U^\intercal$

(rl are singular values from decomposition and related to squeezing parameters via $\arctan(rl)$) and the latter does a specific decomposition for graph applications; it starts by selecting the value for the scaling factor (with the help of mean photon number per mode from the argument), creates a new adjacency matrix with it and only then uses sf.decompositions.takagi() to decompose returning squeezing parameters and the interferometer unitary.

5.2 graph algorithms

As we have indicated, the matrix A from the GBS model is related to graph adjacency matrices and there are multiple important applications in using graph algorithms so lowering their computational complexity is very beneficial for several areas.

A great and easier starting problem is dense subgraph identification that can be used to detect denser subgraphs, such as required to identify communities in social networks.

5.2.1 Dense subgraph identification

Let us start by defining the problem of dense subgraph identification. The density of a graph is found relating the cardinality of nodes with the number of edges connecting them. Consider a graph G, a set of nodes V and edges $E(V)$ of those nodes. Then, the density of a graph given by nodes V is:

$$d(V) = \frac{2|E(V)|}{|V|(|V|-1)}$$

Note that the density of a complete graph is exactly one so this formula follows that the number of possible edges in a graph of $|V|$ nodes is $|V|(|V|-1)/2$ and the density is a value according to this maximum, this is, number of edges it has over the number it could have. Also, a subgraph H of a graph G is an Induced Subgraph if every edge in G with both nodes in H is also an edge in H, this is, an induced subgraph of a graph is formed from a subset of the vertices of G and all of the edges of G connecting pairs of vertices in that subset.

The dense subgraph identification problem can then be stated formally as the following: given a graph with fixed dimension and a chosen subgraph size, the algorithm returns the nodes/graph corresponding to the subgraph with highest density. Applications for such algorithm might be identification of a social network, i.e. a group of people strongly connected, finding a dense cluster in a molecule or even in communication networks to capture a congested part of the network (traffic above a certain threshold). Note that we defined the problem with two arguments: the graph and size of subgraph to return; if the size is not specified, although the number of subgraphs is exponential in the number of vertices, there is a polynomial classical algorithm in Charikar (2000) that can find the subgraph corresponding to maximum density but finding such a subgraph with size constraints is NP-hard Khuller and Saha (2009).

Now, how to solve this with a classical computer? There are several classical approaches and first let us consider a naive version. First we could code this problem into a SAT problem with specified conditions: look for nodes of given graph for a particular size and save the density of the induced graph corresponding to these selected nodes; in the end return the graph with highest density. This would blindly run over all nodes and exhaust all combinations for subgraphs which is a number that grows exponentially with the number of nodes from the original graph. A slightly improved algorithm would begin with nodes that are probably better candidates. Since the density is strongly related to number of edges, a good starting point is choosing the node with highest degree (degree of a vertex is the number of edges connecting it) and preferably select neighbour nodes to complete the subgraph to analyse. Taking this logic, we could also do it the other way around: start from all nodes of the graph and exclude those with lowest degree leading to a simple resizing but no guarantees of finding the correct result. Algorithms that use this last method usually work better but can be fooled by graphs with a special layout and a good example is used in Bromley et al. (2020) that is accessible by Strawberry Fields library via Applications module with apps.data.Planted().

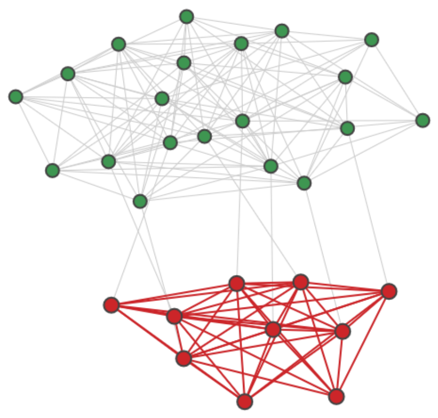


Figure 29: Planted graph from Strawberry Fields; highlighted region in red is the denser 10 node subgraph.

This represents a graph with 30 nodes constructed from two disconnected graphs: a 20-node graph with edge probability $p = 0.5$ and a 10-node graph with $p = 0.875$ and both connected via 8 randomly chosen nodes. A graph resulting from this construction is plotted in figure 29. From construction of this ensemble, we know that the 10 node subgraph is denser than the 20 node part because of the edge probability connection however this is not very obvious by inspection. Although the 20 node subgraph has lower p, those nodes have higher degree so a simple algorithm that is based on removing or adding edges according to its degree, would not work since density is not only proportional to the number of edges (and degree) but to the inverse number of nodes involved. The samples in Strawberry Fields documentation for this graph are available to make some direct numerical experiments.

Having mentioned a few classical approaches to finding dense subgraphs, we shall now proceed to take a look at the quantum implementation. First of all, as we saw before, the output probability of GBS devices is strongly related to the matrix A that fully represents the graph and we analysed the operation of Hafnian in such matrix which, in this particular case, the Hafnian returns the number of perfect matchings. Thus, encoding a graph G into a GBS device it will preferably output states corresponding to subgraphs with higher number of perfect matchings. It was shown in Arrazola and Bromley (2018) that the number of perfect matchings in a graph G with $2m$ vertices is upper bounded by a monotonically increasing function of the number of edges l, which means there is a correlation between the number of perfect matchings and edges of random graphs. This allows one to interpret the output of a GBS device codified with a graph G as a construction that naturally outputs denser subgraphs.

This is an amazing conclusion: we have a device that with the right encoding outputs with high probability the result we are looking for! There is a detail on the value $\overline{n}$: the mean photon number should be the typical sample size for GBS outputs, which should ideally be matched with the size for the densest subgraphs expected to be found. These samples have the following interpretation: if $n_i = 0$, the node in graph corresponding labeled as position i is not part of the subgraph (this node is excluded) so clearly if $n_i = 1$, i is relevant and part of the solution. However we know the optical modes are not restricted to 0 or 1; in that case, if $n_i = 2, 3, 4, \cdots$ that particular node is repeated meaning the subgraph in output makes a copy of the i node n_i times leading to an extended induced subgraph. This is not a major concern since we developed models where both Boson Sampling and GBS have low probability of collision events and there are many algorithms that make use of threshold detectors (Planted graph from Strawberry Fields is one example) that simplify the physical implementation and can result in lower time complexity. The samples for graph in 29 are generated with average number of photons $\overline{n} = 8$ and with threshold detectors. This indicates that even though the structure to identify has 10 nodes, the probabilistic nature of samples allows to select a smaller size 8 and still obtain the results we need. This will be verified in numerical experiments.

The quantum implementation itself has already been specified: we encode the graph G as the matrix A with a chosen mean photon number $\overline{n}$ and use the function sf.decompositions.graph_embed(). To run this program the only missing piece is to describe the quantum algorithm explicitly.

So finally, the steps to dense subgraph identification with a linear-optical quantum network are:

1. Encode the graph into the device with help of sf.decompositions.graph_embed() where arguments are the adjacency matrix of G and the mean photon number to detect. The returning values are the squeezing parameters and unitary ready for implementation;

2. Generate N samples from the device programmed as described above.

3. Process these samples classically: from each sample, if the size of subgraph $k = \sum n_i$ is smaller than our intended value, add a node with the highest degree from the remaining nodes and remove the node with lowest degree otherwise.

4. Calculate the density for each sample and output the one with the highest value.

Thus, the dense subgraph identification quantum algorithm consists of showing us a seed of where we should start looking for solutions, this is, the algorithm identifies optimal structures for specific sizes and the classical algorithm has then a starting point. This is called a hybrid algorithm since it makes use of both quantum and classical parts in the algorithm i.e. it has a classical post-processing portion.

The classical post-processing is also helpful as it allows to use the same quantum-generated seed to look for dense subgraphs of different sizes this is, we do not need to rescale the mean photon number of Takagi decomposition each time we look for a different size.

5.2.2 Maximum clique

From the above algorithm, the theoretical introduction tells us that GBS devices can be programmed to naturally sample subgraphs with higher density. Many problems can take advantage of this feature and an example is the maximum clique problem. In graph theory, a clique is a subset of nodes where each vertex has an edge to every other vertex, this is, a subset is a clique if this induced subgraph is complete. For this problem we intend to find the subgraph of largest size having all possible edges. In contrast to previous solutions, the algorithm for this problem is NP-hard with or without size constraints.

Once again, encoding a graph and sampling from it with a GBS device will work as a starting point and we then post-select from these samples. As the hybrid algorithm is probabilistic, cliques will not be identified always, and we need to consider which classical post-processing is necessary to increase that probability of success. So how do we handle our samples? Usually we won't get a clique hence we take the subgraph and remove nodes until we find one; the selection of which ones to discard are chosen by those that have lowest degree relatively to the subgraph identified. After having a clique $\mathcal{C}$ - either directly from the sample or by rescaling - check the remaining vertices for a node that can be added and consequently increase the clique size. This is done by successively choosing a node from the set $s_0(\mathcal{C})$ - a set where all nodes have a connection to all vertices in $\mathcal{C}$. This selection can also be based on the node degree or it may be done uniformly.

The selection can guide us to a clique smaller than the largest clique contained in the graph. To prevent this, consider another set $s_1(\mathcal{C})$ - the set of nodes from the original graph with connections to all the nodes in the clique except one. If we find a node in this condition, we replace the node it is not connected to with this new node. Note that before we arrived at a dead end hence swapping nodes would never be a step back. If this set is empty, we end the algorithm returning the size and corresponding clique found; otherwise, we go back to expanding $\mathcal{C}$ with the updated set $s_0(\mathcal{C})$.

The steps to solve maximum clique algorithm are the following:

1. Encode the graph into the device with the graph's adjacency matrix and the mean photon number. Use the returning values (squeezing parameters and unitary) for a direct implementation of the GBS device;
2. Generate N samples from the device encoded as described above;
3. From each sample, if the subgraph is not a clique, remove nodes until we find a clique $\mathcal{C}$;
4. After a clique is found, select nodes from the set $s_0(\mathcal{C})$ readjusting after each step;
5. Construct the set $s_1(\mathcal{C})$. If any vertex is found, swap nodes for this set and run step 4 again.
6. Return the size of the maximum clique found and output the induced subgraph that corresponds to this subset of vertices identified.

The theoretical part for implementation on a quantum device in this algorithm is very close to the one presented for dense subgraphs identification: we use GBS device to 'seed' our algorithm and process the returned samples classically. For this reason, numerical experiments will be only carried out on one of these problems.

5.2.3 Graph similarity

There are many possible ways of quantifying the similarity between two graphs. For instance, some classical algorithms determine the degree of similarity between G and H by returning a value in the interval $[0, 1]$ with initial consideration that both already have the same number of nodes and have the correspondence (labelling of nodes) leaving open to explore by the algorithm edges properties like its connectivity and weight. For our purposes, allow us to start by defining graph isomorphism. Two graphs are isomorphic if is there is an isomorphism (structure-preserving mapping) that is classified as a bijection between the sets of edges of G and H. More precisely, for a map f between the sets of nodes of both graphs, an edge in G defined by nodes (u, v) exists if and only if the edge $(f(u), f(v))$ exists in H.

This previous concept is quite restrictive because of the bijection: this requires that the graphs have the same number of nodes, edges and an equal correspondence between them or this relation up to a graph permutation[1]. Also a wide range of graphs might be similar but non-isomorphic and testing such resemblance turns out to be doable in computational terms if the vertex labelling correspondence of the two graphs are given. If not, the classical algorithm can become very inefficient. One aspect to consider too is that although the probability of

1 This permutation has no relation to the permanent or any other related function. Graph permutation refers to exchange of roles between nodes and edges where the nodes represent the elements of permutation and the edges are turned into pairs of elements. Different permutations may give rise to the same permutation graph.

an output is related to the hafnian and as so it takes exponential time to solve, this result is usually exponentially small (recall that random Haar-unitaries are chosen to avoid symmetries) so if the algorithm requires returning this exact value we need to collect enough information to predict such value which would require running the experiment for an exponential time for a reasonable chance of success. So there is no known classical or quantum efficient way of testing for graph isomorphism. So for GBS applications we loosen up this condition and test graph similarity. A way to test similarity is to measure the distance between feature vectors of each graph to analyse with a so-called kernel where the inner product of these vectors is a possible way, which is a well studied technique. A feature vector is an m-dimensional vector of numerical features that represent some object. This approach maps a graph G into those vectors and these in turn can be represented and compared with other graphs in the Feature Vector space. These vectors are obtained from the resulting samples of a GBS device encoded with A as the adjacency matrix of G as before and then postprocessing the outcomes.

The key in this algorithm is to turn GBS into feature vectors. A good proposal is to associate features with probability of a measurement from the GBS device; is was shown in Brádler et al. (2021) that two graphs are isomorphic if and only if their probability distributions from GBS are equal up to permutation[2]. Finding the probability distribution brings two problems: (1) the number of possible events grows with a factorial function over the total number of photons to detect which would lead to an explosion on the dimension of the feature vectors and (2) each probability might be an exponentially small number as mentioned, which would require an unacceptable, exponential number of samples. To treat the first problem we use a technique called coarse-graining. This method turns the feature vector of probabilities for single events into probabilities of certain types of photon events. The first proposal is to group events into orbits introduced in Brádler et al. (2021): an orbit combines all samples that are equivalent. For example, samples $[2, 1, 0, 0, 3]$ and $[3, 0, 1, 0, 2]$ both belong to orbit described by the set $\{3, 2, 1\}$, this is, orbits group samples into equivalent classes according to the set of output mode occupation numbers only. For computational purposes, they are ordered (descending order), this is, a given orbit O_{n} represents all permutations of detection event n so that we group samples from the same orbit together more easily and have a more compact feature vector which is our first goal. The probability for an orbit O_{n} is then the sum over the samples under this permutation leading to next equation:

$$p(O_{\mathrm{n}}) = \sum_{\mathrm{n} \in O_n} p(\mathrm{n})$$

This lowers the dimension for the feature vector but it still leads to a lot of features - for k photons to distributed over M modes, even considering the usual relation $k \ll M$ - so there is another coarse-graining strategy. The second proposal (on top of the first strategy) turns these

2 Two square matrices A and B are permutationally similar if $B = PAP^{\intercal}$ where P is a permutation matrix.

orbits into meta-orbits $\mathcal{M}_{k,n}$. These meta-orbits summarizes samples containing k photons in total and at least one mode has the maximum occupation number n. For better illustration, see figure 30.

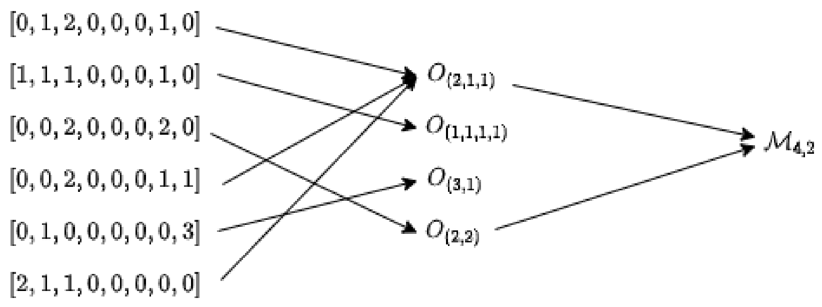


Figure 30: Example of coarse-grain strategies; first to each sample was attributed an orbit and only those belonging to the meta-orbit $\mathcal{M}_{4,2}$ where selected.

The probability for a meta-orbit $\mathcal{M}_{k,n}$ is depicted as:

$$p(\mathcal{M}_{k,n}) = \sum_{\mathrm{n} \in \Delta_S} p(O_{\mathrm{n}})$$

where Δ_S is the combinations of all possible states under those conditions.

Having the coarse-graining process defined, we are left with finding how many samples we require from the GBS device to compute this strategy: too many samples would be heavy for the quantum device and too few would not be enough for a good feature vector. Schuld et al. (2020) defines the a variable S as the number of samples we need from a quantum GBS device making a compromise between number of possible outcomes and the estimated error due to finite sampling, this is, combines sampling size and experimental uncertainties returning a numerical bound. Here due to some open questions in this algorithm and it is used for actual implementations, we will not use this variable for numerical experiments.

Therefore, given two or more graphs to test similarity, one needs to construct feature vectors for them and then run/compare them in a kernel. The probabilities for these vectors from a classical computer can be obtained via $p(\mathcal{M}_{k,n})$ described above and via a quantum computer by the fraction of outcomes in those conditions by the total number of samples $^{n_i}/_{N}$.

At last, we have all necessary steps to the algorithm to test graph similarity in a linear-optical quantum network.

1. Encode graph G into the GBS device through its adjacency matrix A just like previous algorithm and generate preferably S samples from a distribution with mean photon number $\overline{n}$;

2. Group together output samples corresponding to a selected meta-orbit $\mathcal{M}_{k,n}$ with given k and n;

3. Compute the probabilities for each meta-orbit via $p(\mathcal{M}_{k_i,n}) = {}^{n_i}/_{S}$ assigning this vector to graph G.

We will use the Euclidean distance between the resulting feature vectors to quantify the similarity of the corresponding graphs.

Differently from the previous algorithm, here the quantum part of the algorithm does not provide a seed for classical post-processing, but the end-result itself.

In Schuld et al. (2020) the authors described all the theoretical background in detail, showed motivation for measuring the similarity and present their experiments. The authors explain the relation between output photon events with the coefficients for graph properties which can gives us a better understanding and intuition in choosing the appropriate values k and n for what meta-orbits to study.

5.3 numerical experiments on graph problems

In order to test the previous algorithms, it is wise to generate structures to study with the particularities we are looking for, this is, we will generate graphs for a specified input size and some characteristic to test in algorithm. Also, now we have the tools to study the physical requirements with more applicability compared to the approach in previous chapter so in the next subsection we will deal with these two issues and then test the algorithms described.

5.3.1 Graph generation and physical requirements

Let us start with generation of graphs that we can test our algorithms on. To a simple graph generation with size *n_nodes* and connection probability p we can create G with function graphGenerator():

```python
def graphGenerator(n_nodes, p):
    #initializing graph G with n_nodes nodes
    G = nx.Graph()
    G.add_nodes_from([k for k in range(n_nodes)])

    #adding edges with probability p for the given nodes
    for i in range(n_nodes):
        for j in range(i,n_nodes):
            r=np.random.random()
            if r<p: G.add_edge(i, j)
    return G
```

This is the simplest way to generate a random graph given those conditions but for dense subgraph identification it is not a good proposal since the edge probability is the same for all nodes and consequently it is not expected to have denser areas. To generate more adequate structures, we implement a new function that can create denser subgraphs (see full code B.26 or algorithm 2 below).

Algorithm 2 Graph generator with denser regions

```
function GRAPHGENDENSEST($n\_nodes$, $p1$, $n\_densest$, $p2$)
    G1 $\leftarrow$ new graph with $n\_nodes - n\_densest$ nodes
    G2 $\leftarrow$ new graph with $n\_densest$ nodes
    for $i, j \in$G1.nodes() do
        $r$ =np.random.random()
        if $r \leq p1$ then
            G1.add_edge(i,j)
        end if
    end for
    for $i, j \in$G2.nodes() do
        $r$ =np.random.random()
        if $r \leq p2$ then
            G2.add_edge(i,j)
        end if
    end for
    $G$ $\leftarrow$nx.compose(G1,G2)
    Connect $n\_densest/2$ random nodes from G1 to G2
    if There are isolated parts then
        $G \leftarrow$ graphGenDensest($n\_nodes$, $p1$, $n\_densest$, $p2$)
    end if
     return G
end function
```

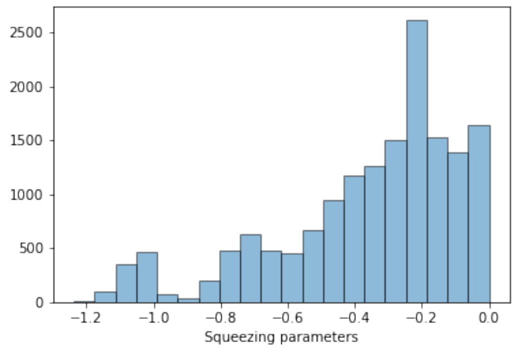
The arguments for this function are the size of graph n_nodes (total number of nodes) with the respective edge probability for these nodes $p1$, the size of denser subgraph $n\_densest$ and edge probability for the denser structure $p2$ where we assume the user attributes valid values for each (meaning $p1 < p2$ and $n\_nodes > n\_densest$). The ensemble is built randomly in the following way. We generate two separate graphs: one of size $n\_densest$ for the denser one and another of size $n\_nodes - n\_densest$; in the end we connect both by selecting $n\_densest/2$ nodes from each and deterministically connect them. To add edges in new graphs we run through all N nodes and consider the $N(N-1)$ possible edges in turn, picking them with a certain chosen probability p. This procedure sometimes will result in disconnected subgraphs, if that is the case we discard the graph and start anew.

Before testing our algorithm on this graph construction, let us discuss the physical requirements for a quantum GBS implementation.

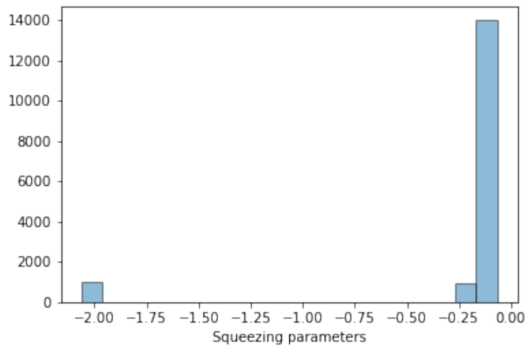
Physical requirements

We can start by analysing the squeezing parameters required for the encoding of different graphs. In order to program a GBS device we need to specify U and each squeezing parameter or, alternatively, A and each mode mean photon number (recall equation 58 that easily relates to r_i by equation 46) where the Autonne-Takagi decomposition accessible by Strawberry Fields function sf.decompositions.takagi(): mean photon number can be equally distributed over the defined modes and then U is obtained with the decomposition.

Allow us to consider the ensemble of matrices we have described, where a denser subgraph is embedded in a larger random graph with smaller density.This ensemble of matrices translating specific graphs results from the above function graphGenDensest. Extracting an adjacency matrix from a graph in that ensemble, we input it along with mean photon number in the previously mentioned function by Strawberry Fields sf.decompositions.graph_embed() to analyse the squeezing parameters they require. In figure 31 we present the histograms of squeezing parameters distribution for an ensemble of graphs with the structure here described.



(a)



(b)

Figure 31: In sub-figure 31a is represented the histogram of distribution of squeezing parameters for an ensemble of 1000 graphs with 16 nodes of edge probability of 0.2 and 4 selected nodes to be a dense subgraph with edge probability of 1. In sub-figure 31b we show the histogram of distribution of squeezing parameters for an ensemble of 1000 graphs with 16 nodes of edge probability of 0.2 and 15 selected nodes to be a dense subgraph with edge probability of 1.

There are two tests: the first is for a common case where we have the relation $\overline{n}^2 \leq m$ and is according to conditions used in chapter 4; the latter is for an extreme example where we test for a denser subgraph with size close to the total/whole graph to see the upper limit for parameters. This last not used in practice but for purposes of studying the demand for the parameters under extreme conditions, i.e. as may be required when the subgraphs being identified are much larger than the sizes we study here. As we can observe, the requirements for squeezing parameters are in range zero to two which is a value that can be done experimentally as presented for example in Arrazola et al. (2021) where they estimated the effective input squeezing in each mode to be approximately $8dB$ which equals squeezing $r \approx 1.776$ in each mode in a 8 mode device (recall subsection 2.3.2). An important note here is that the input of $8dB$ in each mode was obtained by non integrated sources and values for this in a chip are usually smaller. This conclusion says that the second and unusual condition in sub-figure 31b can be hard for integrated technologies but can be done and, on top of that, recall that a typical example is for similar distribution to sub-figure 31a that the mean squeezing value was 0.36 and maximum 1.2 and this is expected to be able to implement in an integrated chip.

5.3.2 Algorithm testing

Now that we have the algorithms and made sure the physical requirements to implement them are doable, we are ready to test the theory to verify that such program outputs what was predicted. Let us take a look at dense subgraph identification. A first test here will be comparing the density from outcomes of a classical random graph generator versus a GBS generator. The former is calculated via a trivial function we called classicalGenerator() in code 5.1 below. Given a graph G, the size of subgraph to identify size and number of samples n_samples, the classical generator randomly chooses size nodes from G; this random choice is made n_samples times (see line 3 of code). After having the random subgraphs identified, we calculate the density of each one and organize these results into an array of densities in line 10.

```
1  def classicalGenerator(G, size, n_samples):
       nodes=list(G.nodes); density=[]
3      amostras=[np.random.choice(nodes, size) for i in range(n_samples)]
       for i in amostras:
5          H=G.subgraph(i)
           density.append(2*len(H.edges)/(size*(size-1)))
7      n_edges=int(size*(size-1)/2)
       n_d=np.zeros(n_edges+1)
9      for i in density:
           ind=int(i*n_edges); n_d[ind]=n_d[ind]+1
11     return n_d
```

Listing 5.1: Classical random selection of subgraphs to test dense subgraph identification.

The latter generator with sorting from GBS sampler is implemented in code B.27. Here, we start by decomposing our graph's adjacency matrix as described before, generate a given number of samples with our method B and post-select the samples for the size we are looking for - vacuum and samples of size that is not the one intended (for this direct approach of classical versus quantum selection) and for considering here only GBS generators directly we do not consider extended induced subgraphs. This selection will discard some and have less samples than the original number generated so we will call an auxiliary function to calculate the remaining the same way. Then we calculate the density of each subgraph from our samples and last but not least reorganize this result into an array of densities. For example, for detection of subgraphs with size 4, the density array has length equal to 6, which is the possible number of densities for a 4-node, 6-edge graph (position i has number of samples with density $i/6$). In figure 32 we have a plot of graph density for results from 5 graphs from both classical and quantum generators described above.

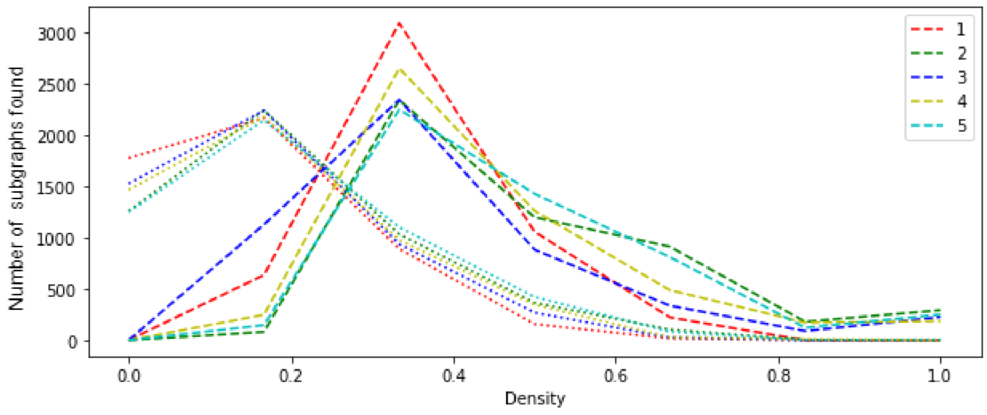


Figure 32: Density of subgraphs generated by random classical selection in dotted lines and by GBS distribution in dashed lines. The figure is the result of subgraph identification for 5 graphs in ensemble $[16, 0.25, 4, 0.95]$ i.e. 16 nodes with edge probability 0.25 and denser subgraph of size 4 with edge probability 0.95. 5000 samples were generated for each case.

There are two main points to note from the figure: the difference in density value 0 and the mean values for both generators. The subgraphs identified by a GBS distribution show densities far higher then classical random ones and at extreme points such as zero density or a clique ($d = 0$ or $d = 1$) we have the opposite: the classical generator returned around $1/3$ of those graphs with zero density and the quantum generator did not print one; for a fully connected output, only the quantum generator was able to obtain such clique samples. There are other classical algorithms that surely result in a better choice than the one presented but this is only the beginning of demonstration of quantum capacity for these problems that makes use of a natural selection and output from both devices, i.e. we do not post-process any of the outcomes.

Using the hybrid algorithm, that is, the one with subgraphs from GBS as a 'seed', we can use the Strawberry Fields functions that resize or search our samples. Both functions can be found in the Applications layer and the first can be reached via sf.apps.subgraph() where the arguments are a subgraph (listed by the numerical label of each node), the graph, maximum and minimum size to resize to (sizes of subgraphs we are looking that are in the specified range) and a default argument for node selection (in case next nodes to remove/add have the same degree). The second function has an additional argument max_count also default for the number of graphs to output for the densest found, this is, for each size we save the *max_count* densest subgraphs. Using our code to generate a random graph and simulate samples, the hybrid dense subgraph identification code can be done using the next code 5.2 where defGraph is an array with graph specifications; In this code we have the Strawberry Fields implemented functions search() and to_subgraphs() in lines 6 and 7 to do some classical processing:

```
1  G=graphGenDensest(defGraph[0], defGraph[1], defGraph[2], defGraph[3])
   n_nodes=len(G.nodes())
3  (r,U)=sf.decompositions.graph_embed(nx.adjacency_matrix(G).todense(),
       mean_photon_per_mode=size/n_nodes)
5  G_A=sampleGeneratorB(np.asarray(U), r, times, limit_photons=6)
   G_A=sample.to_subgraphs(G_A, G); k_min=3; k_max=6
7  results=subgraph.search(G_A, G, k_min, k_max)

9  plot.graph(G)
   plot.graph(G,results[4][0][1])
11 plot.graph(G,results[3][4][1])
```

Listing 5.2: Hybrid algorithm for dense subgraph identification.

Taking the results, we can plot figure 33 where sub-figure 33a was obtained from the last 3 lines of code. We can observe for the first graph that the algorithm worked for different sizes, identifying both the dense subgraph purposefully inserted, but also other dense subgraphs generated by the random graph construction. For the second graph in sub-figure 33b with higher edge probability, the results from the algorithm were also satisfactory.

With improvements in output generation mentioned in last chapter, we were able to simulate 25-node graphs with denser regions of size 5. In figure 34 we have the results for two graphs characterized by defGraph= $[25, 0.3, 5, 0.95]$. Note that the structure identified as denser for the first graph generated (sub-figure 34a) does not correspond to the 5-node subgraph with edge probability 0.95; in fact this is identified in second place with smaller density. This can happen because although edge probability for the general graph is lower, the number of edges to connect is much higher. Still, the algorithm was able to detect them as predicted for both random graphs.

With these numerical experiments for different sizes and densities, we conclude that we succeeded in detecting the denser subgraphs. We also compared the outcomes of a random classical generator with a GBS generator which is an experiment that as low cost for both (classically it is only random selection of number and for GBS is in the device nature with proper codification) and GBS outputs showed being much more efficient.

Since the clique identification algorithm has the same theoretical basis as the above demonstrated code, the code is alike the previous; we encode the graph into GBS with selected mean photon number ideally, corresponding to the clique size we want to detect, and could make

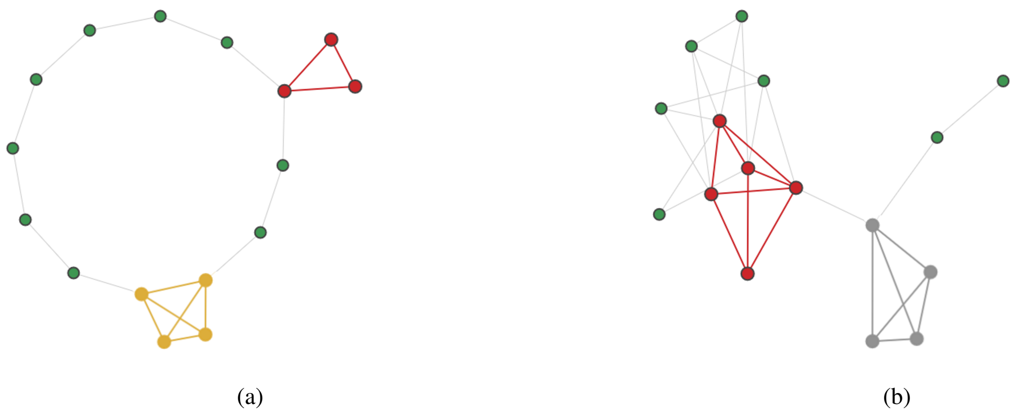


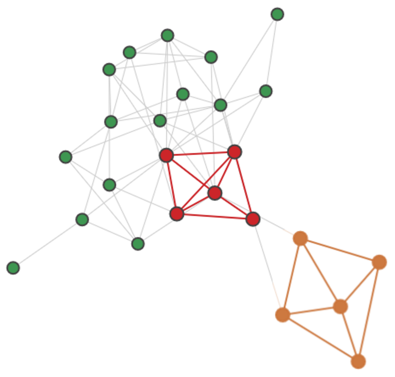
Figure 33: Plot of two random 16-node graphs with one denser region each of size 4 with edge probability 0.95; graphs in sub-figure 33a and 33b have edge probability of 0.2 and 0.35 respectively. Gold and grey highlighted areas are the denser regions from random graph generator. Golden area was identified as the denser region of size 4 in results from code described above and in red at sub-figure 33a represents the identification of a dense region of size 3. This was obtained with the forth densest detected because the first three were attributed to the combinations of the 4-denser region (they all have density$=1$). The red structure in sub-figure 33b corresponds to the denser region found for size 5 while in grey is the first attempt to find such region with size 4.

use of Strawberry Fields to resize and search - steps 3, 4 and 5 of algorithm. So now allow us to make a brief numerical experiment for graph similarity. As mentioned before, there are still some open questions about what values to attribute to meta-orbits, this is, there is not a more appropriate way to chose them so we will use one of our choice that seems pertinent for the case study.

To complete the necessary code to test similarity, we implement an auxiliary function that does the last two steps described in earlier section: it assigns each sample to a meta-orbit and calculates the corresponding probability, this is, returns the feature vector according to description of meta-orbits for given samples both given as function arguments. See next code 5.3 for more detail.

```
def featureVec_metaOrb(samples, totals, n):
    N=len(samples); l=[]
    ordered=[np.flip(np.sort(sample)) for sample in samples]
    for i in ordered:
        if i[0]>n: continue
        else: l.append(sum(i))
    l=np.array(l)
    return [sum(l==i)/N for i in totals]
```

Listing 5.3: Function that given samples and meta-orbits description, returns the feature vector for these conditions.


(a)

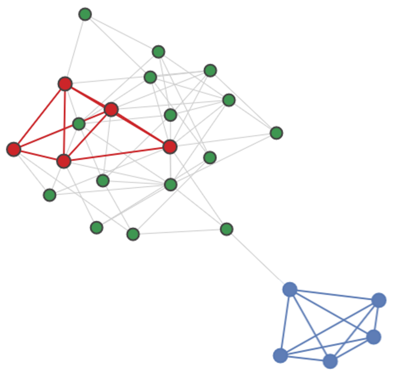
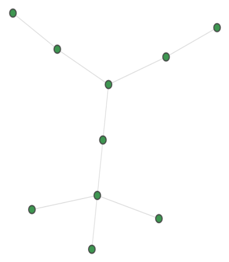
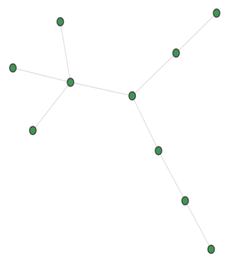

(b)

Figure 34: Plot of two random 25-node graphs. Orange and blue highlighted structures correspond to denser regions from our graph generator code. The first 5-node subgraph identified for sub-figure 34a was red region and for sub-figure 34b was blue region with densities of 0.9 and 1 respectively. The other regions are the 5−node subgraphs identified in second place with densities 0.8 and 0.9 respectively.

Having a function to calculate feature vectors, we use function graphsFetVec() (see full code in B.28) that returns feature vectors for meta-orbits $\mathcal{M}_{k_i,n}$ given parameters n and k as function arguments along with the graphs to test and two default arguments - for mean photon number and number of samples required. With this, we have all necessary steps to run the algorithm.

We start by generating 3 graphs: two closely related and one fundamentally different; all three represented in figure 35. To test our algorithm, we decided to evaluate graphsFetVec() results for four graphs where the first two are actually the same (sub-figure 35a) to examine what the similarity calculation would do.


(a)


(b)

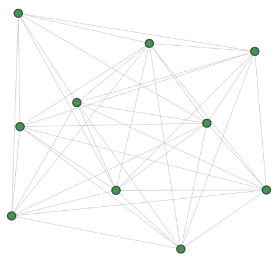

(c)

Figure 35: Random graphs to test similarity with described algorithm. The first two (G1 and G2) in sub-figures 35a and 35b are generated with the same parameters using our function for graph generation graphGenerator() with arguments $(10, 0.1)$. Last sub-figure 35c graph (G3) is generated by the same function but with edge probability 0.9.

In figure 36 we have plotted the results from using graphsFetVec() for the four graphs described. The return for those four graphs ([G1, G1', G2, G3]) was $[0.3126, 0.1154, 0.0142]$, $[0.3158, 0.1178, 0.0122]$, $[0.3082, 0.125, 0.0194]$, $[0.2106, 0.0738, 0.013]$ and each Euclidean distance to G1 given by $d_i = d(G_i, G_1)$ was successively $d_1' \approx 0.0045$, $d_2 \approx 0.0118$ and $d_3 \approx 0.1102$. As depicted, graphs that are more similar are closer to each other. This closeness highly depends on the number of samples and given the similarity of G1 and G2, we obtained several times this feature vectors closer than the two generated from G1.

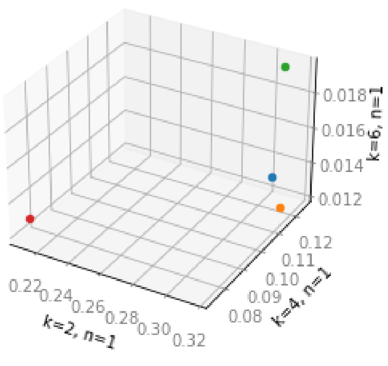


Figure 36: Plot of feature vector for four graphs described. Meta-orbits were characterized by k= $[2, 4, 6]$ and n= 1 and mean photon number and number of samples was the default attributed values (size= 7 and times= 5000). Blue and orange dots are results from graph G1, green dot is from graph G2 and red dot is representation of feature vector for G3.

In appendix for code we added two extra tests for this algorithm under very similar conditions: we used the same graphs but with different meta-orbits ($n = 2$) and more samples. This is plotted in appendix figure 39.

5.4 future perspectives

Applications including graphs covers a wide range of problems as mentioned. Some of these problems studied have an efficient classical algorithm such as dense subgraph identification without size restriction and for those cases, the GBS device could present an advantage only by a polynomial factor. Besides, in our simulation we tested for unweighted and undirected graphs - described by positive adjacency matrices - that according to algorithm by Quesada and Arrazola (2020), if σ_Q is both non-negative and a proper quantum covariance matrix, then matrix A is also non-negative and the probabilities in equation 54 can be approximated efficiently so our tests could too. The example of dense subgraph identification with size constraints does not have an efficient algorithm and for weighted or directed graphs, the GBS algorithm can not be approximated to a polynomial time algorithm. We can conclude once more that evolution of classical algorithms forces quantum devices to improve and show exponential advantages only under certain conditions. To apply this to real world problems, we need to additionally determine if these conditions for quantum advantage are the usual ones or hardly used.

In this section we review a few of the other applications of GBS that have been proposed in the literature.

5.4.1 Other applications with qumodes

Another application that can be run in GBS devices with encoding similar to graphs is computing molecular vibronic spectra. This problem consists of analysing the pattern of frequencies at which light is more strongly absorbed by a certain molecule. This absorption spectra of molecules is important in determining, for instance, their usage in photovoltaics or monitoring global warming. Classically, the vibronic spectrum of a molecule is given by Franck-Condon profile (FCP) - a function that determines the probability of generating a transition at a given vibrational frequency and some efficient classical algorithms have been found to calculate the FCP. However, to be considered an efficient algorithm, the requirement is polynomial run-time which can still grow considerably and consequently this has become difficult to calculate for larger molecules. Here enters GBS: for a linear-optical quantum computer, in Bourassa et al. (2021) we have the full description for codification and once more the device provides an output that naturally solves the problem. This is, we incorporate the molecule characteristic into the device and only need to rearrange the returned samples.

Another application is for neural networks. Killoran et al. (2019a) analysed the connection between classical neural networks and the processing part with CV quantum computation. Essentially, neural networks are based on a multi-layer technique where each layer is a linear transformation. Each layer is composed by weight matrix W and the bias vector b and a function that combines them with the input via $\varphi(Wx + b)$. These layers are applied sequentially where the final network is translated into function composition for each layer and they are followed by a non-linear part. At this point, encoding this into a GBS is somewhat intuitive: the weight matrix W and the bias vector b translate into the covariance matrix and displacement vector; linear transformation to our linear interferometer and nonlinear elements are non-Gaussian elements, namely photo-detection. The function composition for layers is also intrinsic to this system.

It is worth mentioning that quantum optical systems are also important for applications other than computation. Photons are excellent information carriers, and can be used in quantum communication, teleportation Killoran et al. (2019b), and quantum key distribution (QKD) Zhang et al. (2019).

5.4.2 Optical qubits

The description of the Boson Sampling model allows to adapt and make a connection to processing 2-level quantum information, this is, the usual qubit approach for quantum computing (more detail in appendix section A.2). Note that we mentioned before that Boson Sampling was mainly studied to show quantum supremacy, not real-life applications.

Since qubits can be built out of any quantum system having two distinguishable states to fulfill the requirement for 2-level system for state representation of $|0\rangle$ and $|1\rangle$ (see section A.2), we can associate Boson Sampling to this model with an approach towards scalable photonic quantum computing proposed by Knill et al. (2001). State preparation now is a bit different from the usual models we have reviewed so far because a qubit will be described by two qumodes and here is why: if a quantum state is $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$, we can define $|0\rangle$ and $|1\rangle$ with Fock states as written in next equation:

$$|0\rangle = |0\rangle \otimes |1\rangle = |01\rangle\,; \quad |1\rangle = |1\rangle \otimes |0\rangle = |10\rangle \tag{62}$$

The initial state is vacuum for both modes so of course we need a single photon source to produce these input states which we considered hard to implement physically; for now this is not exactly an efficient proposal for qubit given the reasons explored in chapter 3. However, we expect that technology to evolve in time and other proposals for qubits also have some drawback but for Knill et al. (2001) proposal it is sufficient to be able to prepare these states non-deterministically.

After state preparation, we need to define state evolution describing the unitary matrix U. With the model in consideration, U is an interferometer made up by combinations of beam splitters BS and phase shifters P so here we want to turn the usual qubit operations and describe them with U_{BS} and U_P. From introduction to qubit preparation and unitaries in appendix section A.2 we can start by making equivalence between our unitaries - U_{BS} and U_P - and matrices in table 3 with help of the Bloch sphere representation. Using power series expansion and equation 63 and knowing that Pauli matrices are involutory ($\sigma_y^2 = \hat{I}$), a simple phase shifter applies a rotation on z axis as $U_P(\phi) = \exp(-i\sigma_z\phi/2)$ and a beam splitter $U_{BS}(\theta) = \exp(-i\sigma_y\theta)$:

$$\begin{aligned} U_{BS}(\theta, 0) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} &= \cos\theta\hat{I} - i\sin\theta\hat{\sigma}_y = \\ &= \hat{I}\left(1 - \frac{\theta^2}{2!}\cdots\right) - i\hat{\sigma}_y\left(\theta - \frac{\theta^3}{3!}\cdots\right) = \\ &= \exp(-i\theta\hat{\sigma}_y) \end{aligned}$$

We can conclude that all qubit rotations can be implemented with linear optics and we have transformations for one qubit. From the universal set of gates, to achieve full power of quantum computation we still need an equivalent operation for CNOT, i.e. two-qubit gates capable of producing entangled states. One option to implement this requires strong photon-photon interactions which are hard to engineer which leads to the necessity of nonlinear components into the quantum network, for instance, the Kerr gate. Other solution is proposed in the cited article by Knill et al. (2001) which is to effectively create photon-photon interactions via the measurement process; in their work it is implemented a conditional sign flip on one mode using two ancilla modes making use of three beam splitters and one phase shifter and measurement of these two ancilla modes in the end.

Another approach to represent qubits is using CV cluster states, this is, the elements studied in the GBS model. This new approach only requires displacement, squeezing, linear interferometry, Gaussian measurements and photo-detection, all seen in previous chapters. An obvious advantage is the scalability of the physical implementation. These qubits are called GKP qubits after Gottesman, Kitaev and Preskill who proposed them in article Gottesman et al. (2001); they are superposition of Gaussian eigenstates centered in a specified grid in phase space.

The idea behind this proposal is to prepare non-Gaussian states efficiently (as opposed to single photon generation) combining displacement and squeezing with an efficient non-Gaussian operation: photo-detection. Recall in chapter 2 where we discussed measurements, measuring a mode in a Gaussian system with photo-detection, if the outcome is $n \neq 0$, then the remaining modes become non-Gaussian. If for a system of N modes, we prepare a multi-mode input state of squeezed and displaced vacuum $\hat{\mathcal{D}}(\alpha)\hat{S}(\zeta)\left|0\right\rangle$ and measure $N-1$ modes, we obtain a GKP state for a specific pattern of detection. The Wigner function of these states in the ideal form is a sum of delta functions with specific spacing determined by the preparation (see article by Tzitrin et al. (2020) for details) and is fixed at $2\sqrt{\pi}$. These spacings and geometry in the phase space is modelled by a symplectic matrix and, for computational terms, it is used a rectangular shape geometry. For this reason, these states are often referred to as grid states.

In the article by Bourassa et al. (2021) we have a description of GKP states where the two level states $\left|0\right\rangle_{gkp}$ and $\left|1\right\rangle_{gkp}$ are written as the following dimensionless combination in the position axis:

$$\left|\mu\right\rangle_{gkp} = \sum_{n} \left|(2n+\mu)\sqrt{\pi}\right\rangle_{q}$$

So a qubit can be written with the usual form as in description of Bloch sphere and now we need to define the other components for a universal model. The position axis encodes the logical 0 and 1 values associated with Z; with similarity to Z axis for logical 0 and 1, the momentum axis encodes the X eigenstates according to Tzitrin et al. (2020) as:

$$|+\rangle_{gkp} = \sum_{n=-\infty}^{\infty} |n\sqrt{\pi}\rangle_p \text{ and } |-\rangle_{gkp} \equiv Z(\sqrt{\pi})\,|+\rangle_{gkp} = \hat{\mathcal{D}}(i\sqrt{\pi}/\sqrt{2}\,|+\rangle_{gkp}$$

We know that the representation for the qubits uses to orthogonal states and we can conclude this for our GKP states. The spacing in phase space and use of delta functions as peaks or considering that $|1\rangle_{gkp}$ are complementary states of $|0\rangle_{gkp}$ it leads to the following inner product:

$$\langle 1|0\rangle_{gkp} = \sum_n \langle (2n+1)\sqrt{\pi}|2n\sqrt{\pi}\rangle = \delta((2n+1-2n)\sqrt{\pi}) = 0$$

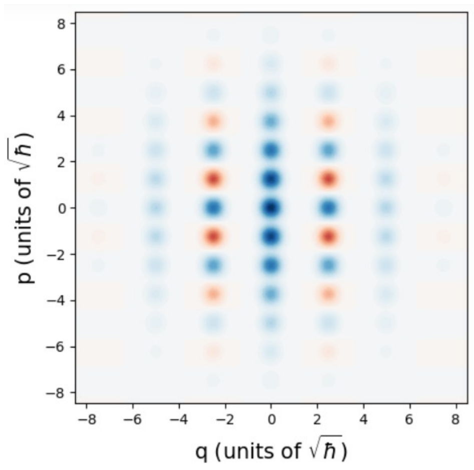


Figure 37: Wigner function representation of $|0\rangle_{gkp}$ with $\epsilon = 0.1$ that stands for the finite energy parameter.

With tools and a tutorial from Strawberry Fields library, figure 37 has a plot of the Wigner function representation for these grid states. The difference in spacing for momentum and position axis is explicitly represented in the figure while observing the peaks of delta functions where red dots represent negative functions and blue dots represents positive ones.

To sum up state preparation, the states $|0\rangle_{gkp}$ and $|1\rangle_{gkp}$ are superposition states in the position axis periodically spaced by $2\sqrt{\pi}$ and the two states are separated with spacing $\sqrt{\pi}$. The similar happens for $|+\rangle_{gkp}$ and $|-\rangle_{gkp}$ in momentum axis. This implies that error correction codes must be able to correct up to $\sqrt{\pi}/2$ to distinguish logical basis states.

Having the states defined, we are left with operations. The Hadamard gate (see section A.2) when applied to a basis state, creates superposition so in linear-optics context, a state $|0\rangle_{gkp}$ represented in position axis is turned into $|+\rangle_{gkp}$ with a simple phase-shifter of $\phi = \pi/2$. Pauli operations are now a bit clearer: for instance, take the gate X as a NOT gate, it can be implemented with a displacement by $\sqrt{\pi}$ since $|0\rangle_{gkp}$ and $|1\rangle_{gkp}$ differ by this displacement in the position axis. For universal computation, it is required a CNOT gate or any other with non-zero entangling power; this can be translated into a pair of single-mode squeezers between two beam splitters as we have indication from previous chapters. Some of the most important operations have their translation explicit in table 2.

Here we have non-Gaussian states right in the system preparation to obtain the logical qubits and gates are translated into Gaussian linear-optical operations. The missing step is measurement; for the first time in this dissertation, it is referred homodyne detection in a model that is hard to simulate classically. Let us take a look at what measurement implies

Quantum Gate	CV operation
X	$\hat{\mathcal{D}}(\sqrt{\pi})$
Z	$\hat{\mathcal{D}}(i\sqrt{\pi})$
R_ϕ	$\hat{R}(\phi)$
H	$\hat{R}(\pi/2)$
CNOT	$\hat{BS}(\theta) \cdot (\hat{S}(r) \otimes \hat{S}(-r))\hat{BS}(\theta)$

Table 2: Conversion of CV gates to a 2-level quantum gates.

in Z eigenstates, this is, position axis. Theoretically we have delta functions in phase space hence measuring $|\mu\rangle_{gkp}$ we obtain $n\sqrt{\pi}$ and the parity of n indicates state $|0\rangle$ if n is even and $|1\rangle$ otherwise. In practice, we do not have delta functions so the detected value might not be associated with a specific n. In this case, this number is approximated to the closest $n\sqrt{\pi}$. Thus, the importance of quantum correction codes up to $\sqrt{\pi}/2$.

The fact that they are orthogonal along with being robust and allowing manipulation with Gaussian optical elements as described in table 2 motivates to study these states.

This was a brief description of GKP states and they are still an application being studied at the moment; this is in fact the most promising application for devices with components that essentially use GBS as a building block. This is the approach to quantum computation put forth by Canadian company Xanadu. For simulation purposes, Xanadu has a backend to simulate this new approach since the physical requirements are close to GBS and this is a promising application. The backend used to this simulation is a different one than Fock and Gaussian used in previous chapters due to their nature. It is the 'Bosonic' backend that simulates quantum optical circuits by representing states as linear combinations of Gaussian functions in phase space.

5.5 summary

Applications for this chapter were focused on graph problems so we started the chapter with an introduction to graph theory where we reviewed the some basic facts necessary for the comprehension of following sections. The key concepts were graph representation (classical algorithms use edge lists to save memory but adjacency matrices A are easily encoded into quantum computers) and matchings in a graph. These matchings can be found in several conditions and for arbitrary graphs but our interest is for perfect matchings that happens when, for a graph with even n number of nodes, all vertices are matched, this is, there a $n/2$ number of matchings. The Permanent and the Hafnian of matrices that represent graphs are strongly related to the calculation of these matchings; the Hafnian is the general formula and it counts the number of perfect matchings in a graph.

To make use of Haf(A) where A is the adjacency matrix, we need to prepare it to be a valid matrix to physical implementation. From equation 61 we have the covariance matrix from an adjacency matrix that has the following form:

$$\sigma_Q = (\mathbb{1}_{2N} - X_{2N}A)^{-1} - \frac{\mathbb{1}_{2N}}{2}$$

From chapter 2 we know it has some constraints and we made the conditions A needs to lead to a σ_Q that corresponds to a physical Gaussian state. To solve symmetry and $\sigma_Q > 0$ we have that:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \Rightarrow \begin{cases} A_{12} = A_{21} \\ A_{22} = A_{11}^{\intercal} \end{cases} ; \qquad 0 < c < 1/\lambda_{max} \rightarrow A' = cA$$

where λ_{max} is the largest eigenvalue of A. A matrix with these two conditions/modifications is Hermitian and positive definite. A last consideration is to make sure the commutation relations are satisfied which implies that sub-matrix $A_{12} \geq 0$. One option is to use the doubling process and another is to ensure that A has no negative values which most do (unweighted matrices only have values 0 and 1 and weighted usually has positive values). Decomposition of A into interferometer unitaries and squeezing parameters are made via Strawberry Fields functions.

With this encoding ready, we start to study graph algorithms. The first was dense subgraph identification. The density of a subgraph is the fraction of number of edges in the graph over the total possible number of edges and finding this can be a difficult problem. If the size is not specified, Charikar (2000) presents a polynomial classical algorithm that finds the subgraph of maximum density but finding the denser subgraph with size constraints is NP-hard Khuller and Saha (2009). This can be solved as a SAT problem but it is highly inefficient so we discussed a few classical alternatives and some particularities the algorithm might find that complicates the subgraph selection. Then we proceeded to the quantum approach. Arrazola and Bromley (2018) showed that the number of perfect matchings in a graph G with $2m$ vertices is strongly related to the number of edges. This connection becomes relevant because if the probability of outputting a sample S representing a subgraph is proportional to number of perfect matchings - Haf(A_S) - and this number is related to the number of edges, then the GBS device encoded with a graph G naturally outputs denser subgraphs. So using a nanophotonic chip with right encoding we need to generate multiple samples, calculate the density of each and, given a size, output the corresponding subgraph with highest density. A similar algorithm is the Maximum clique; if the Hafnian of a graph returns denser subgraphs, finding a clique should also be a high probability sample. Both these algorithms work as a seed for the actual problem which is a nice advantage. For the maximum clique, probabilistic samples means we do not always find what we are looking for so it was described a classical post-processing portion of the algorithm. This portion can be both used for classical and quantum algorithms except that starting points differ: quantum part has samples and classical algorithm can start with a clique of minimum size (a node or two connected nodes).

The last graph problem was graph similarity. We described how to measure and represent the similarity between graphs. A proposal to solve it is by using feature vectors. A classical algorithm defines what features it is looking for, rearranges those properties and uses a kernel to return a value for the similarity; a quantum algorithm also uses feature vectors and uses a technique called coarse-graining to organize the samples according to their occupation number. When the final feature vectors are constructed, the inner product is an example of measuring the similarity and for better visualization, we can plot them in the Euclidean space.

Before testing the algorithms, we showed how we implemented the generation of random graphs and took the opportunity to see if the encoding of these arbitrary graphs can be done with the current available hardware. We conclude that yes, we can! Even in extreme conditions, the maximum values can be implemented although most likely in present hardware only on no integrated technologies but the usual values (squeezing in interval $[0, 1.2]$ focused on the smaller requirements) where accomplished by Arrazola et al. (2021). The numerical experiments on dense subgraph identification and graph similarity were both satisfactory. In the first case we compared classical and quantum random generators and observed that the latter outputs much better samples, as expected. We also identified denser regions in graphs of size 16 and 25 in figures 33 and 34 respectively and in all tests we found the denser regions for the sizes required. For similarity, we tested the algorithm for three graphs were two of them were generated with the same conditions (number of nodes and edge probability). The results were obtained and plotted for four graphs were the first two were the same. As expected, similar graphs were closer.

In the last section we focused on other applications for which we did not make numerical experiments. Vibronic spectra and neural networks can be encoded into a GBS devices similar to graphs were some parts of the system is given by a matrix that can either be represented by a matrix A or σ_Q and make use of photodetection in the end. Vibronic spectra can be solved in a polynomial time in a classical computer so implementing in a quantum hardware is not necessarily better or required. This an open discussion as it is still being studied and many other proposals may arise in a near future for potentially useful applications.

We ended the chapter with applications fundamentally different than what was considered previously. Quantum teleportation and QKD are two good examples since photons are excellent information carriers. However, we still focused on quantum computation and considered describing qubits with BS and GBS. With a device similar to BS we have the Knill et al. (2001) dual rail encoding scheme where $|0\rangle$ and $|1\rangle$ are defined in equation 62 with Fock states:

$$|0\rangle = |0\rangle \otimes |1\rangle = |01\rangle \,; \quad |1\rangle = |1\rangle \otimes |0\rangle = |10\rangle$$

Single mode operations are implemented with beam splitters and phase shifters; two-mode operations are generally hard to engineer due to strong photon-photon interactions but Knill et al. (2001) suggested another solution that makes use of ancilla modes and measurement on them linear optics operations only.

Using a device with same nature as GBS, it is possible to encode qubits with the following construction:

$$|0\rangle_{gkp} = \sum_{n} |(2n)\sqrt{\pi}\rangle_q \quad |+\rangle_{gkp} = \sum_{n=-\infty}^{\infty} |n\sqrt{\pi}\rangle_p$$

This periodicity and geometry is a consequence of the state preparation: it combines Gaussian input states with Fock measurements, all in the first stage. Single qubit operations in these states become intuitive where a NOT gate is easily applied with a displacement gate since for instance $|0\rangle$ and $|1\rangle$ only differ by a displacement of $\sqrt{\pi}$. Table 2 summarizes the equivalence of CV gates to qubit gates necessary for arbitrary quantum computation. This is the only model in all dissertation that does not require photon detection in the end of the circuit, even though the full scalable quantum photonic architecture employing these GKP encoded qubits will naturally also require measurement read-out.

6

CONCLUSIONS

In this dissertation, we started by reviewing the fundamental theory of physical elements for the models of quantum computation using linear optics. By the end of chapter 2, we had reviewed the basic description of preparation of either Fock states or Gaussian states of light, their linear-optical evolution, and a few elements that could render the classical simulation inefficient.

Throughout the dissertation we focused on studying the representation and complexity of two linear-optical models for quantum computation in order to demonstrate quantum supremacy. It had some challenges because of two central reasons: physical instruments for quantum hardware can be hard to implement (photon loss, preparation...) and classical algorithms are always evolving which makes it difficult to show the exponential advantage for an universal model. More particularly, better classical algorithms not only can make them closer to computational complexity of quantum algorithm but demands for better devices; one example is photon distinguishability where for this case the permanent calculation is efficient which requires sources for indistinguishable photons. This is an example of the interplay between quantum hardware improvements and improvements in the classical simulation of these processes.

An interesting problem we did not address in this dissertation was weak simulation namely the Clifford and Clifford (2017) algorithm that is one of the most efficient simulations for the Boson Sampling model: one is able to generate a sample calculating only approximately two permanents. Although the complexity can be lowered significantly, it still requires computing the permanent, a computationally intractable matrix function. In physical implementations for Boson Sampling there is Wang et al. (2019) which could be a good proposal for encoding qubits in quantum oscillators, by Knill et al. (2001) even though boson sampling is not an universal model. The biggest obstacle here was the state preparation (that we expect to be developed in a near future) which served as a motivation to look at variations of the boson sampling model. As for Gaussian Boson Sampling, Zhong et al. (2021) made an experiment for many modes that classical algorithms could not solve in polynomial time but their interferometer is random and also not programmable; it is focused on low loss and high transmission rate and their techniques are useful for future implementations. This implementation was able to sample in 200 seconds a system that they estimate it would take 2.5 billion years to calculate on a supercomputer. Another proposal is the programmable chip by Arrazola et al. (2021) which is the first to demonstrate to be ready for scalable hardware with efficient state preparation and dynamically programmable interferometer at the cost of being a small device that classical computers can still simulate.

With proper algorithms and these photonic devices, there is the problem of validating the models. In chapter 3 we discussed a method that requires calculating the probability of output by the nanochip which we saw to be related to the Permanent or Hafnian, both functions hard to compute. For small chips such as the proposal by Arrazola et al. (2021), we can still simulate but a device like the one demonstrated by Zhong et al. (2021) proposal can not. To do so, we briefly mentioned a technique where one divides the system in two separate parts, i.e. disconnect the interferometer in the middle so that there are no interactions and calculate the two probabilities for the output state, one for each half. Validation methods can also be improved.

Although BS and GBS can show quantum supremacy in a near future using a programmable chip with all conditions required for quantum computation, direct applications for those models are not very promising at the moment comparing to classical methods. Applications to graph theory could bring several advantages due to how many problems can be encoded into adjacency matrices. However, some algorithms in chapter 5 are also efficient in classical simulation and in those specific cases the best we could hope is a polynomial speed up. Using these models as a system for representing qubits is the application we think would be most promising.

From this work, we see that there are many prospects for future work: in physical implementation for photonic chips, in classical algorithms, in validation, applications and many others. It is expected that physical experiments significantly improve in a near future and it then becomes impossible to classically simulate them, as has already arguably happened in the case of the Gaussian Boson Sampling experiment. Most importantly, for applications, programmable devices, some of which already available on the cloud Xanadu, promise to showcase advantage in different applications, and use for encoding information as necessary for the longer-term goal of scalable quantum computation.

BIBLIOGRAPHY

Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, volume 9 of STOC '11, page 143–252, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450306911. doi: 10.4086/toc.2013.v009a004. URL http://dx.doi.org/10.4086/toc.2013.v009a004.

Scott Aaronson and Travis Hance. Generalizing and derandomizing gurvits's approximation algorithm for the permanent. Quantum Information and Computation, 14(7-8):541–559, May 2014. doi: 10.26421/QIC14.7-8-1. URL https://doi.org/10.26421/QIC14.7-8-1.

Gerardo Adesso, Sammy Ragy, and Antony R. Lee. Continuous variable quantum information: Gaussian states and beyond. Open Systems & Information Dynamics, 21(1-2), Mar 2014. ISSN 1793-7191. doi: 10.1142/s1230161214400010. URL http://dx.doi.org/10.1142/S1230161214400010.

J. M. Arrazola, V. Bergholm, K. Brádler, T. R. Bromley, M. J. Collins, I. Dhand, A. Fumagalli, T. Gerrits, A. Goussev, L. G. Helt, and et al. Quantum circuits with many photons on a programmable nanophotonic chip. Nature, 591(7848):54–60, Mar 2021. ISSN 1476-4687. doi: 10.1038/s41586-021-03202-1. URL http://dx.doi.org/10.1038/s41586-021-03202-1.

Juan Miguel Arrazola and Thomas R. Bromley. Using gaussian boson sampling to find dense subgraphs. Physical Review Letters, 121(3), Jul 2018. ISSN 1079-7114. doi: 10.1103/physrevlett.121.030503. URL http://dx.doi.org/10.1103/PhysRevLett.121.030503.

A. M. Barnett and P. M. Radmore. Methods in Theoretical Quantum Optics. Oxford science publications. Clarendon Press, 1997. ISBN 9780198563624. URL https://books.google.pt/books?id=Hp5z09gPmC8C.

Stephen D. Bartlett, Barry C. Sanders, Samuel L. Braunstein, and Kae Nemoto. Efficient classical simulation of continuous variable quantum information processes. Physical Review Letters, 88(9):097904, Nov 2002. ISSN 1079-7114. doi: 10.1103/physrevlett.88.097904. URL http://dx.doi.org/10.1103/PhysRevLett.88.097904.

Marco Bentivegna, Nicolò Spagnolo, Chiara Vitelli, Fulvio Flamini, Niko Viggianiello, Ludovico Latmiral, Paolo Mataloni, Daniel Brod, Ernesto Galvão, Andrea Crespi, R. Ramponi, Roberto Osellame, and Fabio Sciarrino. Experimental scattershot boson sampling. Science Advances, 1(3), May 2015. ISSN 2375-2548. doi: 10.1126/sciadv.1400255. URL http://dx.doi.org/10.1126/sciadv.1400255.

Andreas Björklund, Brajesh Gupt, and Nicolás Quesada. A faster hafnian formula for complex matrices and its benchmarking on a supercomputer. Journal of Experimental Algorithmics, 24(1):1–17, Jun 2019. doi: 10.1145/3325111. URL https://doi.org/10.1145/3325111.

J. Eli Bourassa, Rafael N. Alexander, Michael Vasmer, Ashlesha Patil, Ilan Tzitrin, Takaya Matsuura, Daiqin Su, Ben Q. Baragiola, Saikat Guha, Guillaume Dauphinais, and et al. Blueprint for a scalable photonic fault-tolerant quantum computer. Quantum, 5:392, Feb 2021. ISSN 2521-327X. doi: 10.22331/q-2021-02-04-392. URL http://dx.doi.org/10.22331/q-2021-02-04-392.

Daniel J. Brod, Ernesto F. Galvão, Andrea Crespi, Roberto Osellame, Nicolò Spagnolo, and Fabio Sciarrino. Photonic implementation of boson sampling: a review. Advanced Photonics, 1(3):1 – 14, 2019. doi: 10.1117/1.AP.1.3.034001. URL https://doi.org/10.1117/1.AP.1.3.034001.

Daniel Jost Brod. Bosons vs. fermions – a computational complexity perspective. Revista Brasileira de Ensino de Física, 43, Jan 2021. ISSN 1806-9126. doi: 10.1590/1806-9126-rbef-2020-0403.

Thomas R Bromley, Juan Miguel Arrazola, Soran Jahangiri, Josh Izaac, Nicolás Quesada, Alain Delgado Gran, Maria Schuld, Jeremy Swinarton, Zeid Zabaneh, and Nathan Killoran. Applications of near-term photonic quantum computers: software and algorithms. Quantum Science and Technology, 5(3):034010, May 2020. ISSN 2058-9565. doi: 10.1088/2058-9565/ab8504. URL http://dx.doi.org/10.1088/2058-9565/ab8504.

Kamil Brádler, Pierre-Luc Dallaire-Demers, Patrick Rebentrost, Daiqin Su, and Christian Weedbrook. Gaussian boson sampling for perfect matchings of arbitrary graphs. Physical Review A, 98(3), Sep 2018. ISSN 2469-9934. doi: 10.1103/physreva.98.032310. URL http://dx.doi.org/10.1103/PhysRevA.98.032310.

Kamil Brádler, Shmuel Friedland, Josh Izaac, Nathan Killoran, and Daiqin Su. Graph isomorphism and gaussian boson sampling. Special Matrices, 9(1):166–196, Jan 2021. ISSN 2300-7451. doi: 10.1515/spma-2020-0132. URL http://dx.doi.org/10.1515/spma-2020-0132.

Gianfranco Cariolaro and Gianfranco Pierobon. Bloch-messiah reduction of gaussian unitaries by takagi factorization. Physical Review A, 94:062109, Dec 2016. doi: 10.1103/PhysRevA.94.062109. URL https://link.aps.org/doi/10.1103/PhysRevA.94.062109.

Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In Klaus Jansen and Samir Khuller, editors, Approximation Algorithms for Combinatorial Optimization, volume 1913, pages 84–95, Berlin, Heidelberg, Sep 2000. Springer Berlin Heidelberg. ISBN 978-3-540-44436-7. doi: 10.1007/3-540-44436-X_10.

William R. Clements, Peter C. Humphreys, Benjamin J. Metcalf, W. Steven Kolthammer, and Ian A. Walmsley. Optimal design for universal multiport interferometers. Optica, 3(12):1460–1465, Dec 2016. doi: 10.1364/OPTICA.3.001460. URL http://www.osapublishing.org/optica/abstract.cfm?URI=optica-3-12-1460.

Peter Clifford and Raphaël Clifford. The classical complexity of boson sampling. CoRR, abs/1706.01260, 2017. URL http://arxiv.org/abs/1706.01260.

Thomas M. Cover and Joy A. Thomas. Information Theory and Statistics, chapter 11, pages 347–408. John Wiley Sons, Ltd, 2005. ISBN 9780471748823. doi: https://doi.org/10.1002/047174882X.ch11. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/047174882X.ch11.

A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? Phys. Rev., 47:777–780, May 1935. doi: 10.1103/PhysRev.47.777. URL https://link.aps.org/doi/10.1103/PhysRev.47.777.

David G. Glynn. Permanent formulae from the veronesean. Designs, Codes and Cryptography, 68(1-3):39–47, Jul 2013. doi: 10.1007/s10623-012-9618-1. URL https://doi.org/10.1007/s10623-012-9618-1.

Daniel Gottesman, Alexei Kitaev, and John Preskill. Encoding a qubit in an oscillator. Physical Review A, 64(1), Jun 2001. ISSN 1094-1622. doi: 10.1103/physreva.64.012310. URL http://dx.doi.org/10.1103/PhysRevA.64.012310.

Brajesh Gupt, Juan Miguel Arrazola, Nicolás Quesada, and Thomas R. Bromley. Classical benchmarking of gaussian boson sampling on the titan supercomputer. Quantum Information Processing, 19(8), Jul 2020. ISSN 1573-1332. doi: 10.1007/s11128-020-02713-6. URL http://dx.doi.org/10.1007/s11128-020-02713-6.

Craig S. Hamilton, Regina Kruse, Linda Sansoni, Sonja Barkhofen, Christine Silberhorn, and Igor Jex. Gaussian boson sampling. Physical Review Letters, 119(17), Oct 2017. ISSN 1079-7114. doi: 10.1103/PhysRevLett.119.170501. URL http://dx.doi.org/10.1103/PhysRevLett.119.170501.

C. K. Hong, Z. Y. Ou, and L. Mandel. Measurement of subpicosecond time intervals between two photons by interference. Phys. Rev. Lett., 59:2044–2046, Nov 1987. doi: 10.1103/

PhysRevLett.59.2044. URL https://link.aps.org/doi/10.1103/PhysRevLett.59.2044.

Anatole Kenfack and Karol Zyczkowski. Negativity of the wigner function as an indicator of non-classicality. Journal of Optics B: Quantum and Semiclassical Optics, 6(10):396, Aug 2004. ISSN 1741-3575. doi: 10.1088/1464-4266/6/10/003. URL http://dx.doi.org/10.1088/1464-4266/6/10/003.

Samir Khuller and Barna Saha. On finding dense subgraphs. In Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I, volume 5555 of ICALP '09, pages 597—-608, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 9783642029264. doi: 10.1007/978-3-642-02927-1_50. URL https://doi.org/10.1007/978-3-642-02927-1_50.

Nathan Killoran, Thomas R. Bromley, Juan Miguel Arrazola, Maria Schuld, Nicolás Quesada, and Seth Lloyd. Continuous-variable quantum neural networks. Physical Review Research, 1(3), Oct 2019a. ISSN 2643-1564. doi: 10.1103/physrevresearch.1.033063. URL http://dx.doi.org/10.1103/PhysRevResearch.1.033063.

Nathan Killoran, Josh Izaac, Nicolás Quesada, Ville Bergholm, Matthew Amy, and Christian Weedbrook. Strawberry fields: A software platform for photonic quantum computing. Quantum, 3, Mar 2019b. doi: 10.22331/q-2019-03-11-129. URL http://dx.doi.org/10.22331/q-2019-03-11-129.

M. S. Kim, W. Son, V. Bužek, and P. L. Knight. Entanglement by a beam splitter: Nonclassicality as a prerequisite for entanglement. Physical Review A, 65(3), Feb 2002. ISSN 1094-1622. doi: 10.1103/physreva.65.032323. URL http://dx.doi.org/10.1103/PhysRevA.65.032323.

Emmanuel Knill, Raymond Laflamme, and Gerard James Milburn. A scheme for efficient quantum computation with linear optics. Nature, 409(6816):46–52, 2001.

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), Sep 2019. ISSN 2469-9934. doi: 10.1103/physreva.100.032326. URL http://dx.doi.org/10.1103/PhysRevA.100.032326.

Leonard Mandel and Emil Wolf. Optical coherence and Quantum Optics. Cambridge University Press, 1995. doi: 10.1017/CBO9781139644105.

Alexandra E Moylett, Raúl García-Patrón, Jelmer J Renema, and Peter S Turner. Classically simulating near-term partially-distinguishable and lossy boson sampling. Quantum Science and Technology, 5(1):015001, Nov 2019. ISSN 2058-9565. doi: 10.1088/2058-9565/ab5555. URL http://dx.doi.org/10.1088/2058-9565/ab5555.

Michael A. Nielsen and Isaac L. Chuang. Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press, 2010. doi: 10.1017/CBO9780511976667.

H.C. Ohanian. Principles of Quantum Mechanics. Prentice Hall, 1990. ISBN 9780137127955. URL https://books.google.pt/books?id=2qvvAAAAMAAJ.

Nicolás Quesada and Juan Miguel Arrazola. Exact simulation of gaussian boson sampling in polynomial space and exponential time. Physical Review Research, 2(2), Apr 2020. ISSN 2643-1564. doi: 10.1103/physrevresearch.2.023005. URL http://dx.doi.org/10.1103/PhysRevResearch.2.023005.

M. Reck, A. Zeilingerand H. J. Bernstein, and P. Bertani. Experimental realization of any discrete unitary operator. 73(1):58, 1994.

Bahaa E. A. Saleh and Malvin Carl Teich. Fundamentals of Photonics. John Wiley & Sons, Inc., 1991. ISBN 9780471213741. doi: 10.1002/0471213748. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/0471213748.

S Scheel. Permanents in linear optical networks. Arxiv.org, pages 1–6, 2005. URL http://arxiv.org/abs/quant-ph/?0406127.

Maria Schuld, Kamil Brádler, Robert Israel, Daiqin Su, and Brajesh Gupt. Measuring the similarity of graphs with a gaussian boson sampler. Phys. Rev. A, 101:032314, Mar 2020. doi: 10.1103/PhysRevA.101.032314. URL https://link.aps.org/doi/10.1103/PhysRevA.101.032314.

Ilan Tzitrin, J. Eli Bourassa, Nicolas C. Menicucci, and Krishna Kumar Sabapathy. Progress towards practical qubit computation using approximate gottesman-kitaev-preskill codes. Physical Review A, 101(3), Mar 2020. ISSN 2469-9934. doi: 10.1103/physreva.101.032315. URL http://dx.doi.org/10.1103/PhysRevA.101.032315.

Leslie G. Valiant. The complexity of computing the permanent. Theoretical Computer Science, 8(2):189–201, 1979.

Niko Viggianiello, Fulvio Flamini, Luca Innocenti, Daniele Cozzolino, Marco Bentivegna, Nicolò Spagnolo, Andrea Crespi, Daniel J Brod, Ernesto F Galvão, Roberto Osellame, and et al. Experimental generalized quantum suppression law in sylvester interferometers. New Journal of Physics, 20(3):033017, Mar 2018. ISSN 1367-2630. doi: 10.1088/1367-2630/aaad92. URL http://dx.doi.org/10.1088/1367-2630/aaad92.

Hui Wang, Jian Qin, Xing Ding, Ming-Cheng Chen, Si Chen, Xiang You, Yu-Ming He, Xiao Jiang, L. You, Z. Wang, and et al. Boson sampling with 20 input photons and a 60-mode interferometer in a 10^{14}-dimensional hilbert space. Physical Review Letters, 123(25), Dec 2019. ISSN 1079-7114. doi: 10.1103/physrevlett.123.250503. URL http://dx.doi.org/10.1103/PhysRevLett.123.250503.

Christian Weedbrook, Stefano Pirandola, Raúl García-Patrón, Nicolas J. Cerf, Timothy C. Ralph, Jeffrey H. Shapiro, and Seth Lloyd. Gaussian quantum information. Rev. Mod. Phys., May 2012. doi: 10.1103/RevModPhys.84.621. URL https://link.aps.org/doi/10.1103/RevModPhys.84.621.

Yichen Zhang, Zhengyu Li, Ziyang Chen, Christian Weedbrook, Yijia Zhao, Xiangyu Wang, Yundi Huang, Chunchao Xu, Xiaoxiong Zhang, Zhenya Wang, Mei Li, Xueying Zhang, Ziyong Zheng, Binjie Chu, Xinyu Gao, Nan Meng, Weiwen Cai, Zheng Wang, Gan Wang, Song Yu, and Hong Guo. Continuous-variable QKD over 50 km commercial fiber. Quantum Science and Technology, 4(3):035006, May 2019. ISSN 2058-9565. doi: 10.1088/2058-9565/ab19d1. URL http://dx.doi.org/10.1088/2058-9565/ab19d1.

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, 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, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. Science, 370(6523):1460–1463, 2020. doi: 10.1126/science.abe8770. URL https://www.science.org/doi/abs/10.1126/science.abe8770.

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. doi: 10.1103/PhysRevLett.127.180502. URL https://link.aps.org/doi/10.1103/PhysRevLett.127.180502.

A

FUNDAMENTALS OF QUANTUM COMPUTING

a.1 bra-ket notation and hilbert space

The bra-ket notation introduced and named after by Paul Dirac was created to describe physical quantum states by a state vector $|\Psi\rangle$ named ket which refers to the symbol $|\ \rangle$ the following way:

$$|\Psi\rangle = (c_0, c_1, c_2, ..., c_N)^T$$

where x^T denotes the transpose of the $N \times 1$ column vector. Each bra corresponds to a ket and vice-versa and represents its dual such that

$$\langle\Psi| = (c_0^*, c_1^*, c_2^*, ..., c_N^*).$$

These vectors are represented in the Hilbert space $\mathcal{H}$ (usually a complex one), an abstract vector space in which is defined a scalar product and is complete. The dimension of $\mathcal{H}$ is the cardinality of the orthonormal basis representing the states.

For a system and states with this description we have a collection of mathematical operations that are relevant. Some of them are the following:

- The scalar product is denoted as $\langle\Phi|\Psi\rangle$. This determines how linear decomposed $\langle\Phi|$ is into $|\Psi\rangle$. If $|\Psi\rangle$ is the state of a system, the probability that a measurement finds the system in state $|\Phi\rangle$ is $|\langle\Phi|\Psi\rangle|^2$. This implies that the sum for all possible outcomes $\langle q_n|$ (the eigenstates of a defined operator) must equal one:

  $$\sum_n |\langle q_n|\Psi\rangle|^2 = 1$$

  and the states satisfy the normalization condition.

- The outer product is written as $|\Phi\rangle\langle\Psi|$ and produces a matrix from those two vectors with dimension $N \times N$ equal to length of both states $|\Phi\rangle$ and $|\Psi\rangle$. This can also be called a projection.

- The tensor product is denoted as $|\Phi\rangle \otimes |\Psi\rangle$. This operation is used to represent a system composed of several subsystems. This is, analysing a system with states $|\Phi\rangle$ and $|\Psi\rangle$ followed by a transformation on both, it is convenient to see them together and $|\Phi\rangle \otimes |\Psi\rangle$ represents this union. In mathematical terms, this is equivalent to create a new state with dimension N^2 where N is length of $|\Phi\rangle$ (and $|\Psi\rangle$). Usually this notation is simplified to just $|\Phi\rangle |\Psi\rangle$ or $|\Phi, \Psi\rangle$. This separation of states or trying to write a system based on its subsystems is an essential detail for quantum mechanics as we can see by studying entanglement.

The evolution of a system is described in the general form by the Schrödinger equation:

$$i\hbar \frac{d}{dt} |\Psi(t)\rangle = \hat{H} |\Psi(t)\rangle$$

where $\hat{H}$ is the Hamiltonian of the system. Since these vectors represent a state, operations over them can represent its evolution in a particular aspect. These operations transform a vector into another vector and are $N \times N$ matrices. So the operator A acting on state $|\Psi\rangle$ is a ket $A|\Psi\rangle$ and is computed via the multiplication of both.

Given a generic operator $\hat{P}$ with eigenstates $|\Psi\rangle$ and eigenvalues p_n, one can define an operator with the form $e^{\hat{P}}$ that has the same eigenstates $|\Psi\rangle$ and with eigenvalues λ_n such that $\lambda_n = e^{p_n}$. This conclusion can be retrieved from expanding the exponential with Taylor series:

$$e^{\hat{P}} = \hat{I} + \hat{P} + \frac{1}{2}\hat{P}^2 + \frac{1}{6}\hat{P}^3 + ... \quad \rightarrow \quad e^{\hat{P}} |\Psi\rangle = \lambda_n |\Psi\rangle \tag{63}$$

and calculate its eigenvalues of a state:

$$(\hat{I} + \hat{P} + \frac{1}{2}\hat{P}^2 + \frac{1}{6}\hat{P}^3 + ...) |\Psi\rangle = (1 + p_n + \frac{p_n^2}{2} + \frac{p_n^3}{6} + ...) |\Psi\rangle = e^{p_n} |\Psi\rangle = \lambda_n |\Psi\rangle$$

Trace

The mean value of A is given by

$$\bar{A} = \langle \hat{A} \rangle = Tr(\rho \hat{A})$$

where Tr denotes the trace operations which is carried out by summing the diagonal matrix elements of the operator $\rho\hat{A}$. Note that the trace is independent of the basis of the operator as long as it consists in a complete orthonormal set of states.

a.2 2-level quantum information

The fundamental unit for classical computation is a bit that has either value 0 or 1. Everything is encoded into these bits. The quantum analog is a qubit (short for quantum bit) that not only can be 0 or 1 but also a combination - superposition state - of the following qubit state $|\psi\rangle$:

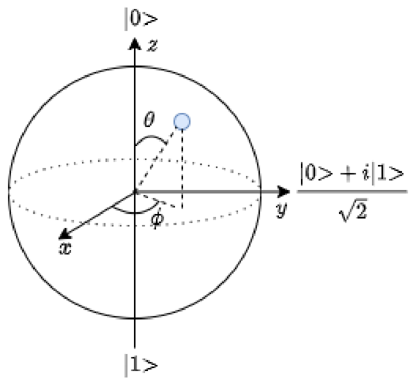
$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \quad \text{where } |0\rangle = \begin{pmatrix}1\\0\end{pmatrix};\ |1\rangle = \begin{pmatrix}0\\1\end{pmatrix} \tag{64}$$

where α and β are complex numbers that must fulfill $|\alpha|^2 + |\beta|^2 = 1$ which each value corresponds to probability of measuring the respective state. This normalization and property of α and β allows a different representation that leads to a different illustration good for visual terms. This is, the qubit state can be written using the complex numbers angles θ and ϕ where the state always has norm one:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \sin\left(\frac{\theta}{2}\right)\exp^{i\phi}|1\rangle$$

Translating this into a visual representation we obtain figure 38. In this figure we represented an arbitrary state but this can be generalized to many however visualizing purposes turns less effective.

(a)



(b)

Figure 38: Bit versus qubit illustration using Bloch sphere. A bit - represented in sub-figure 38a - is restricted to 0 or 1 where one excludes the other. Contrary to this approach, a qubit - represented in sub-figure 38b - is a state with norm 1 defined in the Bloch sphere by angles θ and ϕ. Visualizing we can easily see that to a direct measurement, phase exchange (in parameter ϕ) has no effect. The written states are the extreme values for the corresponding axis in the sphere limit.

With sub-figure 38b we can comprehend easily some importance aspects. The previously used variable α with this notation is equivalent to $\alpha = \cos(\theta/2)$ hence the probability of measuring $|0\rangle$ is $|\alpha|^2 = |\cos(\theta/2)|^2$ where a measure collapses two one of the two poles in that direction (projection onto a basis vector). This last part becomes important because we are not restricted to measuring in z axis but can do the same to x and y. We have then the Pauli operators defined as $\hat{\sigma}_x$, $\hat{\sigma}_y$ and $\hat{\sigma}_z$ that leads to measurements in this 3 directions that are characterized by detecting the states (for short) in Pauli basis $|+\rangle$ or $|-\rangle$, $|+i\rangle$ or $|-i\rangle$ and $|0\rangle$ or $|1\rangle$ each pair respectively to Pauli operators.

For interest of purely computational applications, we have the liberty to choose whatever appropriate system to be $|0\rangle$ or $|1\rangle$. Typical and most common sources are two level systems, this is, physical systems where we can easily distinguish two states or isolate from the other possible. Examples are the following:

- Photon polarization - $|0\rangle$ represents horizontal and $|1\rangle$ represents vertical polarization;
- Superconducting qubits using Josephson juntion;
- Electronic spin - Up and down represent $|0\rangle$ and $|1\rangle$ respectively;
- Nuclear spin by Nuclear Magnetic Resonance (NMR).

There are many other proposals for encoding qubits and they are chosen according to their efficiency since scaling systems to use a great number of qubits becomes complicated for some problems mentioned in chapter 3 and 4. Some issues are extremely common such as decoherence and lost of information can manifest in several ways.

Having defined the states, we should describe the operations we can apply to them. In table 3 we have some of the most used quantum logic gates. The first matrix represented is identity I; this operation does not modify the system and is only represented for mathematical and simplification purposes. In second place is the rotation gate R_ϕ; as the name indicates it rotates the state by an angle ϕ. For instance, applying to state $|0\rangle$ we have $\mathrm{R}_\phi |0\rangle = |0\rangle$ and for $|1\rangle$ we obtain $e^{i\phi} |1\rangle$. This gate does not interfere in measuring a state before of after it is applied; it is a phase rotation - see Bloch sphere for better intuition. The 3 following gates are Pauli gates which are the Pauli matrices; they correspond to a rotation of π over the respective axis. For this reason, X is often called NOT gate since $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$, Z a phase-flip for $|1\rangle$ ($|0\rangle \to |0\rangle$ and $|1\rangle \to -|1\rangle$) and Y simultaneously phase-flips (for $|1\rangle$) and bit-flips while applying i to the state. Then we have Hadamard gate H that creates superposition given a basis state; this is, $|0\rangle \to |+\rangle$ and $|1\rangle \to |-\rangle$. Last but not least, we have Controlled NOT gate or CNOT for short. This is the first gate mentioned that acts on two modes or, more particularly, on more than one mode. The controlled qubit in this case is the first draw with a dot and the target is the second qubit with a circle. The operation on the target happens when the controlled qubit is $|1\rangle$ and CNOT specifically flips the target qubit if this is the case.

Many other operations can be implemented from this. An example is controlled gates where we can use a control qubit to apply conditionally a unitary to other target qubit or qubits even. The same goes for measurements and we can use several qubits as control too.

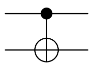
Quantum Gate	Unitary	Symbol
I	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$-\boxed{I}-$
R_ϕ	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$	$-\boxed{R}-$
X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$-\boxed{X}-$
Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$-\boxed{Z}-$
Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$-\boxed{Y}-$
H	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$-\boxed{H}-$
CNOT	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	

Table 3: 2-level quantum gates.

Generating new arbitrary matrices to implement algorithms makes universality necessary and decomposition is required as we needed here in interferometer in this thesis models. This decomposition for this approach can be done in polynomial time using gates from the polynomial set (composed by rotations over two axis and CNOT).

B

CODE IMPLEMENTATION AND SIMULATION

b.1 strawberry fields library

The Walrus is a library for the calculation of hafnians, GBS and other related problems with optimized algorithms. Functions used in Strawberry Fields use functions from this library to write faster algorithms. The company Xanadu developed for the Strawberry fields library a language to work with programming a GBS device and some more photonic-related topics allowing to run the program in a actual device or simulating where the latter makes use of The Walrus library.

Strawberry Fields is a cross-platform Python library focused on problem solving, execution and simulation of quantum photonic hardware. To start with, in order to do such simulation, it is required to perform operations, prepare states and realize measurements as described in chapter 2 along with a platform capable of such classical description. This and encoding into a quantum processor is detailed in Killoran et al. (2019b). Strawberry Fields is not restricted to linear optics; they also implement other functionalities outside the propose in this thesis (for instance, Kerr gate and Cubic Phase gate).

To write and run a Strawberry Fields program, it is essentially divided in three parts:

1. Program initialization. The program is created and initialized with the number of modes and a name (optional).

2. Program physical description. This is where preparation and operations are made.

3. Run program. To run the program we first have to define the engine (where the program will run, see later notes) and only then run.

The first part is a simple line of code that only requires the import of this library. The second part is realized within a context manager for the program, this is, in our classical simulation code we use a program direct to calculation (one for BS and another for GBS) but Strawberry Fields is more general and define these conditions within the context manager: along with the with statement, the context allows to allocate and manage resources according to the needs the programmer wants. With the with statement and the program context, we prepare the input states and implement the proper operations to our qumodes. With the purpose of defining our system, Strawberry Fields offers several operations and, for the conditions in this thesis, we consider the following where parameters within square brackets are optional:

- State preparation. The main and most used in our code are the following:
  1. Vacuum(). This state is already the default initialization.
  2. Coherent($[r, \phi]$) where r and ϕ are the polar representation of α being respectively displacement amplitude and phase angle.
  3. Squeezed($[r, p]$) where r and p are the squeezing parameter and angle of ζ. This prepares the so called squeezed vacuum.
  4. Fock($[n]$) where n is the photon number for the particles in the associated qumode.

  They also offer other ways such as DensityMatrix($state$) that prepares a state given a density matrix in Fock basis and Gaussian($V[, r, decomp, tol]$) prepares a Gaussian state given a valide covariance matrix V. Default values for parameters are zero or None except for $tol = 1 \times 10^{-6}$ (the tolerance used when checking if the matrix is symmetric) and $decomp = True$ for decompose operation into a sequence of elementary gates.

- Application of gates.
  1. Rgate($theta$) where $theta$ is the angle ϕ in equation 34.
  2. BSgate($[theta, phi]$) where default values are $theta = \pi/4$ and $phi = 0$ and they represent respectively θ and ϕ for the beam splitter in equation 36.
  3. S2gate($r[, phi]$) and Sgate($r[, phi]$) in equations 40 and 37 respectively. Within this settings, they also have Dgate($r[, phi]$) where they all are similar to state preparation but in contrary to that, here is required the squeezing parameter and displacement amplitude for squeezing and displacement gates in function arguments.
  4. Interferometer($U[, mesh, drop\_identity, tol]$) where U is the matrix describing the interferometer and default values are $mesh$ ='rectangular', $drop\_identity$ =True and $tol = 1 \times 10^{-6}$.

  Other gates we won't present because are outside of the BS and GBS applications but inside the linear optics are Xgate(x) and Zgate(p) that are positium and momentum displacement gate (specific cases for Dgate($r[, phi]$)) and also CXgate($[s]$) and CZgate($[s]$) that are controlled addition or sum gate in the position basis and controlled phase gate in the position basis.

- Performing measurements.
  1. MeasureFock($[select, dark\_counts]$) measures the qumodes returning $n \in \mathbb{N}$.
  2. MeasureThreshold($[select]$) - avalanche detectors that returns $n \in [0, 1]$.

3. MeasureHomodyne($phi[, select]$) performs a homodyne measurement on a mode as introduced the first basis in equation 44 returning $x_\theta \in \mathbb{R}$. For specific values of θ or argument phi we are measuring one of the quadratures and they can be performed via MeasureX and MeasureP.
4. MeasureHeterodyne($[select]$) performs a heterodyne measurement on a mode also introduced the basis in equation 44 returning $\alpha \in \mathbb{C}$. One can also use MeasureHD for short when no arguments are specified (MeasureHeterodyne() =MeasureHD)

The $select$ arguments are used for post-selection if needed.

In the end of circuit preparation, one needs to select the appropriate engine where the program should run. Here is the first step to distinguish simulators from quantum processors where the former is via the Strawberry Fields engines and the latter is via a remote engine. For simulation it is now available four backends: Gaussian, Fock, TensorFlow and Bosonic. Let us consider here only the first two since those are the ones used in the core of dissertation.

The Fock and Gaussian backend treat the system with fundamentally different descriptions. As the names suggest, the first uses the approach used for Fock states in chapter 2 with countable infinite-dimensional Hilbert space and the second uses the symplectic formalism storing quantum information in displacement vectors and covariance matrices. Regardless of the backend choice, if it is being used to simulate an output in the Fock basis, it is necessary to define a cutoff dimension $cutoff\_dim$ or impose some limit in the dimension of the system. The number $cutoff\_dim$ represents the numerical truncation of the Fock space used by the underlying state. If cutoff is defined by D then it corresponds to the Fock states $\{|0\rangle, \cdots, |D-1\rangle\}$. As for the limit in Gaussian backend, it does not have an equivalent function to all_fock_probs() and to return a probability from simulation we need to input the state we intend to analyse, one by one. For generation of samples they also have a function to return them accessible via sf.apps.sample.sample().

So far we used programs to run with a simulator hence the choice of a backend as an Engine. However, there are already quantum computers based in this model and Xanadu allows access to theirs. To do so, one must configure in their computer with an authentication token given by Xanadu once they accept the request for their use. In this case, the program will run in an actual device where the engine now is the name of the device to chose and results are obtained using eng.run(prog[, shots]) and shots is by default 1 but can be set as any arbitrary integer.

Xanadu and Strawberry Fields simulators both use the assembly language called Blackbird as the quantum instruction set of lower level for this type of quantum computation. Given that Strawberry Fields is open-source, we can see the generated Blackbird code from our program with the function sf.io.to_blackbird(prog, version $='$ 1.0$'$).

b.2 linear optics programs

```
prog = sf.Program(1)
with prog.context as q:
    Vac | q[0]

eng = sf.Engine('gaussian')
state = eng.run(prog).state

fig = plt.figure()
X = np.linspace(-5, 5, 100)
P = np.linspace(-5, 5, 100)
Z = state.wigner(0, X, P)
X, P = np.meshgrid(X, P)
ax = fig.add_subplot(111, projection="3d")
ax.plot_surface(X, P, Z, cmap="RdYlGn", lw=0.5, rstride=1, cstride=1)
fig.set_size_inches(4.8, 5)
```

Listing B.1: Code to plot Wigner function of vacuum state.

```
prog = sf.Program(1)
with prog.context as q:
    Fock(1) | q[0]

eng = sf.Engine('fock', backend_options={"cutoff_dim": 10})
state = eng.run(prog).state

fig = plt.figure()
X = np.linspace(-5, 5, 100)
P = np.linspace(-5, 5, 100)
Z = state.wigner(0, X, P)
X, P = np.meshgrid(X, P)
ax = fig.add_subplot(111, projection="3d")
ax.plot_surface(X, P, Z, cmap="RdYlGn", lw=0.5, rstride=1, cstride=1)
fig.set_size_inches(4.8, 5)
#ax.set_axis_off()
```

Listing B.2: Plotting Wigner function of Fock state $|1\rangle$.

```
prog = sf.Program(1)
with prog.context as q:
    D = Dgate(1.5)
    D | q[0]
```

```
eng=sf.Engine('gaussian')
state = eng.run(prog).state
fig = plt.figure()
X = np.linspace(-5, 5, 100)
P = np.linspace(-5, 5, 100)
Z = state.wigner(0, X, P)
X, P = np.meshgrid(X, P)
ax = fig.add_subplot(111, projection="3d")
fig.set_size_inches(4.8, 5)
ax.plot_surface(X, P, Z, cmap="RdYlGn", lw=0.5, rstride=1, cstride=1)
ax.set_xlabel('X')
ax.set_ylabel('P')
```

Listing B.3: Plotting Wigner function of coherent state $|\alpha = 1.5\rangle$.

```
prog = sf.Program(1)
with prog.context as q:
    S = Sgate(1)
    S | q[0]

eng.reset()
state = eng.run(prog).state
fig = plt.figure()
X = np.linspace(-5, 5, 100)
P = np.linspace(-5, 5, 100)
Z = state.wigner(0, X, P)
X, P = np.meshgrid(X, P)
ax = fig.add_subplot(111, projection="3d")
fig.set_size_inches(4.8, 5)
ax.plot_surface(X, P, Z, cmap="RdYlGn", lw=0.5, rstride=1, cstride=1)
ax.set_xlabel('X')
ax.set_ylabel('P')
#ax.set_axis_off()
```

Listing B.4: Plotting Wigner function of squeezed state $|\zeta = 1\rangle$.

```
U=haar_measure(4)
boson_sampling = sf.Program(4)

with boson_sampling.context as q:
    Interferometer(U) | q

```

```
boson_sampling.compile(compiler="fock").print()
boson_sampling.draw_circuit(write_to_file=False)
```

Listing B.5: Haar random matrix decomposition from interferometer description to beam splitters and phase shifters.

b.3 boson sampling

```
def permanent(U):
    m=len(U)
    if (m==2): perm=U[0][0]*U[1][1]+U[0][1]*U[1][0]
    elif (m==1): perm=U[0][0]
    else:
        cp=np.zeros((m-1, m-1), dtype=complex)
        perm=0; k=0; l=0

        for n in range(m):
            for i in range(m-1):
                for j in range(m-1):
                    if (k==0 and l==n): k+=1; l+=1
                    elif (k==0): k+=1
                    elif (l==n): l+=1
                    cp[i][j]=U[k][l]; l+=1
                k+=1; l=0
            k=1
            perm+=U[0][n]*permanent( cp)
    return perm
```

Listing B.6: Permanent algorithm.

```
def probC(U, S, T):
    '''probC takes as arguments the elements that fully define the system
        (in the final equation for Boson Sampling formula):
          U - numpy matrix [m,m] for the linear interferometer
          S - list [m] input Fock state
          T - list [m] output Fock state
        returns: numpy float for probability Pr(S,T)'''

    #initialize necessary variables
    k=0; summ=0; m=len(U)
    for i in range(m): summ+=S[i];
    UST=np.zeros((summ, summ), dtype=complex)
    rows = [i for sublist in [[idx] * j for idx, j in enumerate(T)] for i
        in sublist]
    columns = [i for sublist in [[idx] * j for idx, j in enumerate(S)]
        for i in sublist]
    UST=U[:, columns][rows]

    permt=perm(UST)
    PR=pow(abs(permt), 2)
    factorial=1
    for i in range(m):
        factorial=factorial*math.factorial(S[i])*math.factorial(T[i])

    return PR/factorial
```

Listing B.7: BS classical algorithm.

```
def probQ(U, inputQ, states):
    '''probQ takes as arguments the elements that fully define the system
        :
        U - numpy matrix [m,m] for the linear interferometer
        inputQ - list [m] input Fock state
        states - list ([m]) output Fock states to return probs
        returns: list[float] for probabilities Pr(inputQ,state) for state
            in states'''

    m=len(inputQ)
    boson_sampling = sf.Program(m)

    with boson_sampling.context as q:
        for i in range(m):            # prepare the input fock states
            if inputQ[i]==0: continue   #or Vac | q[i]
            else: Fock( inputQ[i] ) | q[i]
        Interferometer(U) | q         # apply the matrix

    eng = sf.Engine(backend="fock", backend_options={"cutoff_dim": m+1})
    results = eng.run(boson_sampling)

    probs = results.state.all_fock_probs()
    list_probs = [probs[st] for st in states]
    return list_probs
```

Listing B.8: BS quantum algorithm with Strawberry Fields functions.

```
def states_giverIn(m,f):
    """ returns 1 photon per mode on first f modes and 0 in the remaining
            list(tuple) - [(1,..,1,0,..,0)]
     returns int 0 if number of photons is > number of modes
    """
    if f>m: return 0
    return (1, )*f + (0,)*(m-f)

def states_giverOut(m, f):
    # returns all combinations of f photons in m modes (for each n_i for i in m , sum_i (n_i) =f)
    #          list(tuple*B_m^f)

    s=list(itertools.product(*[range(f+1)]*m))                    #all states
    filtered_states=[s[i] for i in range(1,len(s)) if sum(s[i])==f]
    filtered_states.sort(key=lambda st : sum(st))                    #for better presentation only (1 photon, then 2 etc)
    return filtered_states
```

Listing B.9: State generation for input and output of BS.

```
def initializeComp (U):
    N=len(U)
    #states=list(itertools.product(range(2), repeat=N))      #states with outputs 0 and 1 only
    #states.remove((0,)*N)
    states=[(1,)*N]
    l=len(states); p=np.zeros(l)
    for i in range(l):
        p=[probC(U, states[i], states[j]) for j in range(l) if sum(states[j])==sum(states[i])]; prob=0
        temp=[j for j in states if sum(j)==sum(states[i])]
        for j in range(len(p)):
            prob+=p[j]

def mede(tam,reps,tecnica,gerador):
    tamanhos = range(1,tam)
    setup = 'from __main__ import '+tecnica+'\nfrom __main__ import '+gerador
    return [timeit(setup=setup,stmt=tecnica+"("+gerador+"("+str(i)+"))", number=reps)/reps for i in tamanhos]
```

```
mede(11,2,'initializeComp','haar_measure')
```

Listing B.10: Function to study BS complexity.

```
def states_giver(m, f):
    """arguments:
        m - nr of modes
        f - max nr total photons
    returns list[list] of states"""
    s=list(itertools.product(*[range(f+1)]*m));
    filtered_states=[s[i] for i in range(len(s)) if sum(s[i])<=f]
    filtered_states.sort(key=lambda st : sum(st))
    states=[list(elem) for elem in filtered_states]
    return states
```

Listing B.11: Generation of possible output states for a GBS device up to f photons.

```
def helpA(photonOutput, limit_prob, U, stateIn, times):
    x=np.random.randint(0, len(photonOutput), times);
    y=np.random.uniform(0, limit_prob, times);

    accepted=np.array([photonOutput[x[i]] for i in range(len(x)) if y[i]<
        probC(U,stateIn, photonOutput[x[i]])])
    return accepted

def randomGeneratorA(U, f, times):
    """arguments:
        U - unitary matrix for interferometer in circuit
        f - number of photons for input mode (f<m)
        times - number of samples to generate
    rejection sampling method: function calculates the probabilities for
        each state and samples from this distribution dist
        random choice from all states x and random choice y from 0 to
            highest probability limitprob.
        Keep samples x[i] where y[i]<dist[x[i]]

    return: numpy.array[np.array[m]*times]
    """

    m=len(U); stateIn=states_giverIn(m,f)
    states=states_giverOut(m, f)
    limit_prob=1 #dist.max(); #print(scipy.special.binom(m,f))
    x=np.random.randint(0, len(states), times);
    y=np.random.uniform(0,limit_prob, times);
    accepted=np.array( [states[x[i]] for i in range(len(x)) if y[i]<probC
        (U,stateIn, states[x[i]])] )

    #generate new samples to compensate the ones rejected
    while len(accepted)<times:
        x=helpA(states, limit_prob, U, stateIn, times-len(accepted))

        if accepted.size==0:
            accepted=np.copy(x)
        elif x.size==0: pass
        else: accepted=np.concatenate((accepted, x), axis=0)

    return accepted
```

Listing B.12: BS sampling algorithm with rejection sampling method.

```
def randomGeneratorB(U, f, times):
    """arguments:
        U - unitary matrix for interferometer in circuit
        f - number of photons for input mode (f<m)
        times - number of samples to generate
    method: calculates all probabilities in array dist, does the cumulative
        sum in probs and samples in interval [0,1]
        samples correspond to the probability selected

    return: numpy.array[np.array[m]*times]
    """
    m=len(U); stateIn=states_giverIn(m,f)
    states=states_giverOut(m, f)
    states=np.array(states)

    dist=np.array( [probC(U, stateIn, T) for T in states] );
    probs=np.cumsum(dist)#; print('probs', probs)
    y=np.random.uniform( size=times); y=np.sort(y)#; print('y      ',y)

    x=0;
    while y[0]>probs[x]:
        x=x+1
    accepted=np.array([states[x]])      #initialize with the first one
        accepted and then concatenate the rest

    for i in range(1,times):
        while y[i]>probs[x]: x=x+1      #cycle to go trough states with
            zero probabilities
        accepted=np.concatenate((accepted, np.array([states[x]])), axis
            =0)
    return accepted
```

Listing B.13: BS sampling algorithm with brute force.

```
def randomVal(U, f, times):
    m=len(U); stateIn=states_giverIn(m,f)
    states=states_giverOut(m, f)
    states=np.array(states)

    dist=np.array( [probC(U, stateIn, T) for T in states] );
    probs=np.cumsum(dist)
    y=np.random.uniform( size=times); y=np.sort(y)

    x=0;
    while y[0]>probs[x]: x=x+1
    probBSampler=np.array([dist[x]])
    accepted=np.array([states[x]])

    for i in range(1,times):
        while y[i]>probs[x]: x=x+1
        accepted=np.concatenate((accepted,np.array([states[x]])))
        probBSampler=np.concatenate((probBSampler, np.array([dist[x]])))

    probUniSelec=np.array([1/len(states)]*times)
    return [probBSampler, probUniSelec]
```

Listing B.14: Validation of samples from a (theoretical) Boson Sampler for comparing BS with Uniform Sampler.

```
def unifGenVal(U, f, times):
    m=len(U); stateIn=states_giverIn(m,f)
    states=np.array( states_giverOut(m, f) )
    a=len(states); samples=np.random.choice(a,times)

    probBS=np.array( [probC(U, stateIn, states[i]) for i in samples] )
    probUniSelec=np.array([1/a]*times)
    return [probUniSelec, probBS]
```

Listing B.15: Generation of uniform samples for post validation with BS.

```python
def distributions(m, f, samples, interf='haar'):
    #define input conditions
    states=states_giverOut(m, f)
    states=np.array(states); stateIn=states_giverIn(m,f)
    if interf=='dft': U=DFT_matrix(m)
    elif interf=='hadamard': U=hadamard(m, dtype=complex)/np.sqrt(m)
    else: U=haar_measure(m)

    #obtain our samples
    samplesA=randomGeneratorA(U, f, samples)
    samplesB=randomGeneratorB(U, f, samples)

    n_states=len(states)
    occurencesA=np.zeros(n_states); occurencesB=np.zeros(n_states)
    for i,j in zip(samplesA,samplesB):    #run both arrays in same loop
        for k in range(n_states):
            if (np.array_equal(i,states[k])): occurencesA[k]=occurencesA[k]+1
            if (np.array_equal(j,states[k])): occurencesB[k]=occurencesB[k]+1

    #write distributions
    distT=np.array( [probC(U, stateIn, T) for T in states] )
    distA=occurencesA/(samples)
    distB=occurencesB/(samples)

    statesp=[np.array2string(state, precision=2, separator=',',
                    suppress_small=True) for state in states]

    #to remove elements whose probability is zero for better visualization
    j=0; epsilon=0.00001
    for i in range(len(statesp)):
        if distT[j]<epsilon and distA[j]<epsilon and distB[j]<epsilon:
            distT=np.delete(distT, j)
            distA=np.delete(distA, j);
            occurencesA=np.delete(occurencesA, j)
            distB=np.delete(distB, j);
            occurencesB=np.delete(occurencesB, j)
            statesp=np.delete(statesp, j)
        else: j=j+1

    return [statesp, distT, distA, distB, occurencesA, occurencesB]
```

Listing B.16: Distribution calculation for preparation of comparison in plots. Calculates the theoretical probability distribution and calculates distributions from the two sampling methods all in the same conditions.

```
def plotDistError(matrix, dists, samples):
    #calculating error for each sampling distribution computing the standard deviation for binomial distribution
    yerrA=np.sqrt(dists[4]*(1-dists[2]))/samples
    yerrB=np.sqrt(dists[5]*(1-dists[3]))/samples

    if len(dists[0])>50:
        plt.figure(figsize=(22, 4))
        indexis=np.argsort(dists[1])
        dists[1]=np.array( [ dists[1][i] for i in indexis ] )
        dists[2]=np.array( [ dists[2][i] for i in indexis ] )
        dists[3]=np.array( [ dists[3][i] for i in indexis ] )
        dists[0]=np.array( [ dists[0][i] for i in indexis ] )
        plt.tick_params( labelbottom = False, bottom = False) #labelleft=False,
        plt.xlabel( str(dists[0][0])+', '+str(dists[0][1])+'........ output states........'+ str(dists[0][ len(dists[0])-1 ]))
       #plt.ylabel('Probabilities')
       #plt.bar(dists[0], -dists[1], width=0.5, color='lightgray')
       #plt.axhline(y = -0.000001, color = 'k')

    plt.bar(dists[0], dists[1], width=0.5, color='lightgray')   # experimentar alterar apenas este para barra
    plt.errorbar(dists[0], dists[2], yerrA, fmt='bs',markersize=4)
    plt.errorbar(dists[0], dists[3], yerrB, fmt='g^',markersize=4)

    #if matrix=='dft': plt.title('DFT matrix')
    #elif matrix=='hadamard': plt.title('Hadamard matrix')
    #else: plt.title('Random haar-measured matrix')

    plt.xticks(rotation = 90)
    plt.savefig('saved_figure8m4fhadamard.png',bbox_inches='tight') #, bbox_inches='tight'
    plt.legend(['Theory', 'SamplesA', 'SamplesB'])
    plt.show()
```

Listing B.17: Function to plot boson sampling distributions from theoretical analysis, rejection sampling B.12 and brute force (method B) B.13.

b.4 gaussian boson sampling

```
a0 = 1000.
anm1 = 2.
n = 20
r = (anm1/a0)**(1./(n-1))
#nreps: number of samples for each size (smaller matrices run more times
    for better results since the computer can run them in a small amount
    of time)
nreps = [(int)(a0*(r**((i)))) for i in range(n)]

times = np.empty(n)
for ind,reps in enumerate(nreps):
    start = time.time()
    for i in range(reps):
        size = 2*(ind+1)
        nth = 1
        matrix = haar_measure(size)
        A = matrix @ matrix.T
        A = 0.5*(A+A.T)
        res = hafnian(A)
    end = time.time()
    times[ind] = (end - start)/reps
    print(2*(ind+1), times[ind])
```

Listing B.18: Generation of results to study complexity of calculating the Hafnian. Plot can be found in figure 21.

sum_trunc1() and sum_trunc2() are functions that return CDF given the same input conditions but with different states to test, this is, states_giver() and geraMF().

```
nPhot=[i for i in range(11)]
m=9; r=1
U=haar_measure(m)

plt.plot(nPhot, [sum_trunc1(U,r,i) for i in nPhot], label='states_giver',
     marker='.')
plt.plot(nPhot, [sum_trunc2(U,r,i) for i in nPhot], label='geraMF',
    marker='.')
plt.legend()
plt.xlabel('Total photon Number')
plt.ylabel('CDF')
#plt.title('Probability for different squeezing parameters ')
plt.savefig('truncation_9mMs',bbox_inches='tight')
plt.show()
```

Listing B.19: GBS CDF for same input squeezing parameter in every qumode.

```
nPhot=[i for i in range(11)]
m=9
meanV=0.5
squeezing=squeezeDist2(m,meanV)
U=haar_measure(m)

#print(sum_trunc(haar_measure(m),squeezing, 4))
plt.plot(nPhot, [sum_trunc1(U,squeezing, i) for i in nPhot], label='
    states_giver', marker='.')
plt.plot(nPhot, [sum_trunc2(U,squeezing, i) for i in nPhot], label='
    gera_MF', marker='.')

plt.legend()
plt.xlabel('Total photon Number')
plt.ylabel('CDF')
#plt.title('Probability for different squeezing parameters ')
#plt.savefig('truncation_9mMeansM1',bbox_inches='tight')
plt.show()
```

Listing B.20: GBS CDF for squeezing parameters around a mean value.

```
def helpA(photonOutput, limit_prob, U, squeeze, times):
    x=np.random.randint(0, len(photonOutput), times);
    y=np.random.uniform(0, limit_prob, times);

    accepted=np.array([photonOutput[x[i]] for i in range(len(x)) if y[i]<
        probGBS(U,squeeze, photonOutput[x[i]])])
    return accepted

def sampleGeneratorA(U, squeeze, times, limit_photons=10):
    """arguments:
        U - unitary matrix of interferometer
        squeeze - squeezing parameters (either a value and equal
            squeezing for all modes or list of N squeezing parameters)
        times - number of samples
        limit_photons - truncation number for the sum of number photons
            per mode, default value is 10

    rejection sampling method:
        random choice from all states x and random choice y from 0 to
            highest probability limit_prob.
        Keep samples x[i] where y[i]<probGBS[x[i]]

     returns np.ndarray[np.ndarray]
    """
    limit_prob=probGBS( U,squeeze, np.zeros(len(squeeze), dtype=int) )
    photonOutput=states_giverOut(len(U), limit_photons)
    x=np.random.randint(0, len(photonOutput), times);
    y=np.random.uniform(0, limit_prob, times);

    accepted=np.array([photonOutput[x[i]] for i in range(len(x)) if y[i]<
       probGBS(U,squeeze, photonOutput[x[i]])])

    #generate new samples to compensate the ones rejected
    while len(accepted)<times:
        x=helpA(photonOutput, limit_prob, U,squeeze, times-len(accepted))

        if accepted.size==0:
            accepted=np.copy(x)
        elif x.size==0: pass
        else: accepted=np.concatenate((accepted, x), axis=0)

    return accepted
```

Listing B.21: GBS sampling algorithm with rejection sampling method.

```
def sampleGeneratorB(U, squeeze, times, limit_photons=10):
    """arguments:
        U - unitary matrix for interferometer in circuit
        times - number of samples to generate
        limit_photons - truncation number for number photons per mode,
            default value is 10

    method: calculates all probabilities in array dist, does the cumulative
        sum in probs and samples in interval [0,1]
        samples correspond to the probability selected

    return: numpy.array[np.array[m]*times]
    """
    m=len(U);
    photonOutput=states_giverOut(m, limit\_photons)
    dist=np.array( [probGBS(U, squeeze, T) for T in photonOutput] );
    limit_prob=sum(dist)   #given limit_photons, method has to be adapted
    probs=np.cumsum(dist)
    y=np.random.uniform(0, limit_prob, size=times); y=np.sort(y)

    x=0;
    while y[0]>probs[x]: x=x+1
    accepted=np.array([photonOutput[x]])

    for i in range(1,times):
        while y[i]>probs[x]: x=x+1
        accepted=np.concatenate((accepted, np.array([photonOutput[x]])),
            axis=0)

    return accepted
```

Listing B.22: GBS sampling algorithm with brute force.

```
def distributions(U, squeeze, samples, f):
    #define input conditions
    states=outStates(len(U),f)
    states=np.array(states)

    #obtain our samples
    samplesA=sampleGeneratorA(U, squeeze, samples, limit_photons=f)
    samplesB=sampleGeneratorB(U, squeeze, samples, limit_photons=f)

    n_states=len(states)
    occurencesA=np.zeros(n_states); occurencesB=np.zeros(n_states)
    for i,j in zip(samplesA,samplesB): #run both arrays in same loop
        for k in range(n_states):
            if (np.array_equal(i,states[k])):
                occurencesA[k]=occurencesA[k]+1
            if (np.array_equal(j,states[k])):
                occurencesB[k]=occurencesB[k]+1

    #write distributions
    distT=np.array( [probGBS(U, squeeze, T) for T in states] )
    truncation=(sum_trunc(U, squeeze, f))
    distT=distT/truncation #rescaling theoretical distribution

    distA=occurencesA/(samples)#; print('distA', distA)
    distB=occurencesB/(samples)#; print('distB', distB)

    statesp=[np.array2string(state, precision=2, separator=',',
                        suppress_small=True) for state in states]

    #to remove elements whose probability is zero
    j=0; epsilon=0.00001
    for i in range(len(statesp)):
        if distT[j]<epsilon and distA[j]<epsilon and distB[j]<epsilon:
            distT=np.delete(distT,j)
            distA=np.delete(distA,j);occurencesA=np.delete(occurencesA,j)
            distB=np.delete(distB,j);occurencesB=np.delete(occurencesB,j)
            statesp=np.delete(statesp,j)
        else: j=j+1

    return [statesp, distT, distA, distB, occurencesA, occurencesB]
```

Listing B.23: Function distributions to returns the distributions for a specific configuration given in parameters for the theoretical prediction and both sampling methods.

```
def plotDistError(matrix, dists, samples):
    #calculating error for each sampling distribution computing the
        standard deviation for binomial distribution
    yerrA=np.sqrt(dists[4]*(1-dists[2]))/samples
    yerrB=np.sqrt(dists[5]*(1-dists[3]))/samples
    print('Number of states', len(dists[0]))
    if len(dists[0])>35:
        plt.figure(figsize=(9, 4))
        indexis=np.argsort(dists[1])
        dists[1]=np.array( [ dists[1][i] for i in indexis ] )
        dists[2]=np.array( [ dists[2][i] for i in indexis ] )
        dists[3]=np.array( [ dists[3][i] for i in indexis ] )
        dists[0]=np.array( [ dists[0][i] for i in indexis ] )
        plt.tick_params( labelbottom = False, bottom = False)
        plt.xlabel( str(dists[0][0])+',
            '+str(dists[0][1])+'........output states........'+
            str(dists[0][ len(dists[0])-1 ]))

    plt.bar(dists[0], dists[1], width=0.5, color='lightgray')
    plt.errorbar(dists[0], dists[2], yerrA, fmt='bs',markersize=4)
    plt.errorbar(dists[0], dists[3], yerrB, fmt='g^',markersize=4)

    plt.xticks(rotation = 90)
    plt.savefig('GBS_simulationGraphEns.png',bbox_inches='tight')
    plt.legend(['Theory', 'SamplesA', 'SamplesB'])
    plt.show()
```

Listing B.24: Calculation of error for each sampling probability distribution obtained from function distributions.

Plots for figure 24 can be found by running the next lines of code:

```
plotDistError('haar', distributions(haar_measure(2),[2,2],1000,15), 1000)
plotDistError('dft', distributions(DFT_matrix(2),[2,2],1000,15), 1000)
plotDistError('dft', distributions(DFT_matrix(2),[2,-2],1000, 15), 1000)
```

Listing B.25: Code for two-modes GBS sampler.

b.5 applications

```
def graphGenDensest(n_nodes, p1, n_densest, p2):
    '''Generates a random graph with:
            .n_nodes and edges with probability p1
            .n_densest - selects n_densest nodes to add edges with
                probability p2
    '''
    G1 = nx.Graph()
    G1.add_nodes_from([k for k in range(n_nodes-n_densest)])

    #adding edges with probability p1 for the given nodes
    for i in range(n_nodes-n_densest):
        for j in range(i+1,n_nodes-n_densest):
            r=np.random.random()
            if r<p1:
                G1.add_edge(i, j)

    #adding edges with probability p2 for selected nodes
    G2 = nx.Graph()
    G2.add_nodes_from([k for k in range(n_nodes-n_densest, n_nodes)])

    #adding edges with probability p2 for the given nodes
    for i in range(n_nodes-n_densest, n_nodes):
        for j in range(i+1, n_nodes):
            r=np.random.random()
            if r<p2:
                G2.add_edge(i, j)
    G=nx.compose(G1, G2)
    n_connect=round(n_densest/2)
    for i in range(n_connect):
        node1=np.random.randint(n_nodes-n_densest)
        node2=np.random.randint(n_nodes-n_densest,n_nodes)
        G.add_edge(node1, node2)

    #testing if there are either isolated nodes or isolated subgraphs
    if list(nx.isolates(G))!=[] or len(list(G.subgraph(c) for c in nx.
        connected_components(G)))>1:
        G1.clear(); G2.clear(); G.clear()
        G=graphGenDensest(n_nodes, p1, n_densest, p2)
    return G
```

Listing B.26: Graph generator for denser subgraphs.

```
def GBShelp(U,r,size,times):
    G_A=sampleGeneratorB(U, r, times, limit_photons=5)
    G_A=postselect(G_A,size-1,size+1)
    return G_A

def GBSGenerator(G, size, times=5000):
    n_nodes=len(G.nodes())
    (r,U)=sf.decompositions.graph_embed(nx.adjacency_matrix(G).todense(),
         mean_photon_per_mode=size/n_nodes)

    G_A=sampleGeneratorB(np.asarray(U), r, times, limit_photons=5)
    G_A=postselect(G_A,size-1,size+1)
    while len(G_A)<times:
        G_A=np.concatenate( (G_A,np.array(GBShelp(np.asarray(U), r, size,
             times))) )
    G_A=G_A[:times]

    for i in range( len(G_A) ):
        for j in range(1,n_nodes):
            if G_A[i][j]!=0: G_A[i][j]=j
    density=[]
    for subG in G_A:
        if subG[0]!=0: H=G.subgraph([0]+subG[subG != 0])
        else: H=G.subgraph(subG[subG!=0])
        density.append(2*len(H.edges)/(size*(size-1)))

    n_edges=int(size*(size-1)/2)
    n_d=np.zeros(n_edges+1)
    for i in density:
        ind=int(i*n_edges)
        n_d[ind]=n_d[ind]+1
    return n_d
```

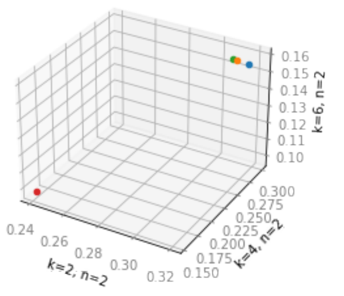
Listing B.27: Function GBSGenerator that outputs a density array of subgraphs of a given graph G from arguments. This function generates subgraphs according to a GBS sampler.

```
def graphsFetVec(graphs, k, n, size=7, times=5000):
    samplesGBS=[]
    for G in graphs:
        n_nodes=len(G.nodes())
        (r,U)=sf.decompositions.graph_embed(nx.adjacency_matrix(G).
            todense(), mean_photon_per_mode=size/n_nodes)
        G_A=sampleGeneratorB(np.asarray(U), r, times, limit_photons=size)
        samplesGBS.append(G_A)
    return [featureVec_metaOrb(samples, k, n) for samples in samplesGBS]
```

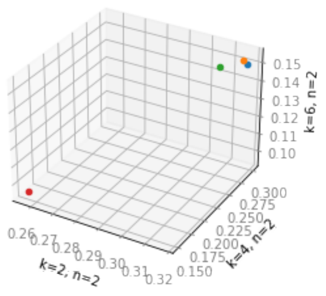
Listing B.28: Calculation of feature vectors for each graph in graphs argument for meta-orbits described by array k and number n. size and times are default arguments for mean photon number and required number of samples.

```
k= [2,4,6]; n=1
fetVecs=graphsFetVec([G1,G1,G2,G3], k, n, size=7)
fig = plt.figure() #figsize=(4,4)
ax = plt.axes(projection='3d');

lbs=['G1', 'G1', 'G2', 'G3']; j=0
for i in fetVecs:
    ax.scatter(i[0],i[1],i[2], label=lbs[j])
    j=j+1

ax.legend()
ax.grid(True)

ax.set_xlabel('k='+str(k[0])+', n='+str(n))
ax.set_ylabel('k='+str(k[1])+', n='+str(n))
ax.set_zlabel('k='+str(k[2])+', n='+str(n));
ax.tick_params(direction='out', length=6, width=2, colors='grey',
    grid_alpha=0.5)
#plt.savefig('app_similarity1.png')
plt.show()
```

Listing B.29: Generation of plot for graph similarity of G1, G1', G2 and G3.



(a)



(b)

Figure 39: Plot of feature vectors for the four graphs described and used in figure 36. Meta-orbits were characterized by k= $[2, 4, 6]$ and n= 2 and mean photon number was the default attributed value (size= 7). Number of samples was default for sub-figure 39a (times= 5000) and for sub-figure 39b was times= 10000. Blue and orange dots are results from graph G1, green dot is from graph G2 and red dot is representation of feature vector for G3.