




---

# Simulating the Photon Statistics of Gaussian States Employing Automatic Differentiation from PyTorch

---

## Technical Report

Erik Fitzke , Florian Niederschuh , and Thomas Walther\* 

\*thomas.walther@physik.tu-darmstadt.de

Technical University of Darmstadt, Institute for Applied Physics

January 2023



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



---

Funded by

**DFG** Deutsche  
Forschungsgemeinschaft  
German Research Foundation

---

---

# Simulating the Photon Statistics of Gaussian States Employing Automatic Differentiation from PyTorch

Technical Report

Erik Fitzke, Florian Niederschuh, and Thomas Walther  
Technical University of Darmstadt  
Institute for Applied Physics

January 6, 2023

URN: urn:nbn:de:tuda-tuprints-230615

License: CC BY 4.0 Attribution International



---

## Abstract

---

Many common photonic states are so-called Gaussian states. In a recent manuscript, we have shown how the photon statistics of such states can be obtained by constructing and differentiating generating functions [1]. In this technical report, we demonstrate the straightforward application of the framework PyTorch to compute the required multivariate higher-order derivatives by automatic differentiation. Its implementation requires only a few lines of Python code corroborating the strength of our approach based on generating functions for the computation of photon statistics.

---

## Contents

---

<b>1. Background: Obtaining the Photon Statistics from a Generating Function</b>	<b>4</b>
<b>2. Python Code Example</b>	<b>5</b>
<b>A. Covariance Matrix and Displacement Vector – Construction and Examples</b>	<b>10</b>

---

## Introduction

---

Important quantum-optical states such as vacuum, coherent states, squeezed states, thermal states, or two-mode squeezed states generated by spontaneous parametric down-conversion are *Gaussian states* (GSs). When the photon statistics of such states shall be analyzed and the states are transformed and detected in complex optical setups exhibiting e.g. losses, partial interference, and noisy detectors, numerical simulations are required. GSs and their transformations by optical setups can be modeled conveniently by using the covariance formalism, encoding the complete information about the photon statistics of the state in a *covariance matrix*  $\Gamma$  and a *displacement vector*  $d$  [2–7]. Transformations of the Gaussian state correspond to simple matrix operations on  $\Gamma$  and  $d$ .

The simulation of the photon number distribution of GSs is known as Gaussian boson sampling. A simulation approach based on a matrix function called Hafnian has been extensively discussed in the literature concerning the computational complexity and to investigate the computational advantage of quantum computers over classical computers [8–14]. The software library `The Walrus` [15] provides several functions related to Gaussian boson sampling and methods to evaluate the Hafnian function.

In Ref. [1] we have chosen a different approach to evaluate the photon statistics of Gaussian states: we have derived a generating function for the photon statistics, from which not only the photon number distribution but also its moments or factorial moments can be obtained by differentiation. Furthermore, we have shown that the method can also be applied to certain non-Gaussian states, such as photon-added and photon-subtracted Gaussian states.

Our simulation method consists of two steps. First, the covariance matrix and displacement vector of the quantum state to be detected are constructed. More details about setting up the covariance matrix, displacement vector, and transformation matrices are presented in appendix A. The matrix operations necessary for this step can easily be implemented with software, e.g. by using the Python package

NumPy [16] or the software library QuGIT providing specialized function for the simulation of Gaussian state dynamics [17].

In the second step, the covariance matrix and displacement vector are inserted into the generating function, which is subsequently differentiated. In this technical report, we demonstrate the implementation of the second step with a simple example. Automatic differentiation (AD) software allows evaluating the appearing multivariate higher-order derivatives with high numerical accuracy and requires only relatively little implementation effort from the user. For AD, the elementary mathematical operations necessary to compute the value of a function, such as addition, multiplication, or the sine function, are tracked. For the computation of the derivative, the known derivatives of the elementary operations are evaluated and combined to the derivative of the total function according to the chain rule [18, 19]. For example, instead of approximating the derivative of the sine function numerically, the cosine function is evaluated in this step. Thereby, the results obtained by AD are accurate to the working precision. A variety of AD software tools exists and the website [autodiff.org](http://autodiff.org) provides an overview of some of them. AD is often used e.g. in machine learning, which is employed in a consistently growing number of applications. We expect that software libraries for machine learning will be further extended and maintained for the next couple of years and we have therefore chosen to implement AD for our purpose by using the popular machine learning software framework PyTorch [20, 21].

## 1. Background: Obtaining the Photon Statistics from a Generating Function

A GS with  $S$  modes is a quantum state with a *characteristic function*  $\chi_{\hat{\rho}}(\boldsymbol{\xi}) = \text{tr}(\hat{\rho} \exp(i\boldsymbol{\xi}^\top \hat{\mathbf{q}}))$  given by a Gaussian function

$$\chi_{\hat{\rho}}(\boldsymbol{\xi}) = \exp\left(-\frac{1}{4}\boldsymbol{\xi}^\top \boldsymbol{\Gamma} \boldsymbol{\xi} + i\mathbf{d}^\top \boldsymbol{\xi}\right) \quad (1)$$

of  $2S$  variables  $\boldsymbol{\xi}^\top = (\xi_{x_1}, \dots, \xi_{x_S}, \xi_{p_1}, \dots, \xi_{p_S})$ . Here, the vector  $\hat{\mathbf{q}} = (\hat{x}_1, \dots, \hat{x}_S, \hat{p}_1, \dots, \hat{p}_S)^\top$  is a collection of the quadrature operators  $\hat{x}_s = (\hat{a}_s + \hat{a}_s^\dagger)/\sqrt{2}$  and  $\hat{p}_s = (\hat{a}_s - \hat{a}_s^\dagger)/(i\sqrt{2})$  for  $s = 1, \dots, S$ . The state is characterized by the covariance matrix  $\boldsymbol{\Gamma}$  of dimension  $2S \times 2S$  and by the displacement vector  $\mathbf{d}$  with  $2S$  elements. The *generating function* for the photon statistics we have derived in Ref. [1] reads<sup>1</sup>

$$G(\mathbf{w}) = \frac{\exp(-\mathbf{d}^\top \boldsymbol{\Lambda}^{-1} \mathbf{W} \mathbf{d}/2)}{\sqrt{\det \boldsymbol{\Lambda}}} \quad \text{with} \quad \boldsymbol{\Lambda} = \mathbb{1} + \frac{1}{2} \mathbf{W} (\boldsymbol{\Gamma} - \mathbb{1}). \quad (2)$$

The matrix  $\mathbf{W}$  consists of two diagonal blocks  $\text{diag}(\mathbf{w})$  for the  $x$  and  $p$  components, with  $\mathbf{w} = (w_1, \dots, w_S)^\top$ :

$$\mathbf{W} = \text{diag}(\mathbf{w}) \oplus \text{diag}(\mathbf{w}). \quad (3)$$

Each mode of the state is detected by one of  $d = 1, \dots, D$  detectors associated with a *differentiation parameter*  $y_d$ . To obtain the photon statistics, the generating function will be differentiated w.r.t. these parameters. The  $m$ -th mode entering detector  $d$  is denoted  $m_d$  and detected with efficiency  $\eta_{m_d}$ . The

<sup>1</sup>The generating function derived in Ref. [1] depends on two further parameters  $\mathbf{u}$  and  $\mathbf{v}$  which are not relevant for the example discussed in this report. They are therefore omitted.

index  $s$  runs over all mode indices, i.e. enumerates  $m_d = 1_d, \dots, M_d$  for all  $D$  detectors in the order  $1_1, 2_1, \dots, M_1, 1_2, \dots, M_2, \dots, M_D$ . To obtain the photon number distribution, the parameters  $w_s$  are set such that

$$w_s = (1 - y_d)\eta_{m_d}, \quad (4)$$

i.e. each mode is detected with an individual efficiency  $\eta_{m_d}$  and all modes entering the same detector  $d$  share the same parameter  $y_d$ . By slightly modifying the generating function and the definition of  $\mathbf{w}$  in eq. (4), the cumulative probabilities or the moments of the photon number distribution, the falling factorial moments, or the rising factorial moments can be obtained [1].

Detection noise is modeled by multiplying  $G$  with the generating function for the noise statistics. For example, if the detectors exhibit Poissonian noise with a mean number of  $\nu_d$  noise counts the  $d$ -th detector, the function  $G$  is multiplied by the generating function of the Poissonian statistics  $\exp(\sum_d (y_d - 1)\nu_d)$ . The probability  $p(\mathbf{n}, \boldsymbol{\nu}, \boldsymbol{\eta})$  to detect  $n_1$  photons in detector 1,  $n_2$  photons in detector 2 etc. is then obtained by differentiating the generating function  $n_d$  times w.r.t.  $y_d$  for each detector [1]:

$$p(\mathbf{n}, \boldsymbol{\nu}, \boldsymbol{\eta}) = \left( \prod_{d=1}^D \frac{1}{n_d!} \frac{d^{n_d}}{dy^{n_d}} \right) \exp\left( \sum_{d=1}^D (y_d - 1)\nu_d \right) G(\mathbf{w}) \Big|_{\mathbf{y}=0} \quad (5)$$

## 2. Python Code Example

By using PyTorch, the multivariate higher-order derivatives in eq. (5) can be evaluated conveniently. The implementation requires only a few lines of code with the following general strategy:

1. Arrays for covariance matrix  $\boldsymbol{\Gamma}$  and displacement vector  $\mathbf{d}$  are constructed, e.g. as NumPy arrays, and they converted to PyTorch *tensors*.
2. The differentiation parameters  $\mathbf{y}$  are set up as PyTorch tensors. The value of the tensors is the point at which the derivative is evaluated, i.e.  $\mathbf{y} = 0$  (cf. eq. (5)). The option `requires_grad=True` is set so that PyTorch will track the further operations involving these parameters for the subsequent computation of derivatives.
3. The matrices  $\mathbf{W}$  and  $\boldsymbol{\Lambda}$  and the generating function  $G$  are constructed according to eq. (2). Mathematical operations involving tensors depending on the differentiation variables such as `sqrt`, `exp`, or `det` are implemented with PyTorch functions so that these operations are tracked for the computation of the derivative.
4. A function is defined to calculate the detection probability for  $\mathbf{n} = (n_1, \dots, n_D)$  photons from the multivariate higher-order derivatives. PyTorch hides the whole complexity of the derivative computation from the user so that this function consists of only six lines of code.

We have set up the example codes with the parameters used in figure 3c of Ref. [1]. The state considered is a two-mode squeezed vacuum (TMSV) state with 16 independent, equally strong two-mode squeezers and a total mean photon pair number of  $\mu = 3$ . The squeezing angle is set to zero, such that the squeezing parameter is given by  $\chi = r$  (cf. table 1). The signal modes of all squeezers are detected

by the first detector with efficiency  $\eta = 0.8$  and noise  $\nu = 1$  and the idler modes are detected by the second detector with  $\eta = 0.9$  and  $\nu = 2$ .

Listing 1 shows the main part of the code required to calculate the bivariate photon number distribution for the two detectors. With the code below in listing 2, a 3D bar chart of the detection probabilities for up to 4 photons in each detector can be generated. Clearly, this part is not necessary for the actual computation of the derivatives. The resulting bar chart is shown in fig. 1.

When copying and pasting the code, the indentation may be lost. In this case please try to open the file with a different PDF viewer or restore the indentation manually. The code has been tested with the software versions Python 3.10, Matplotlib 3.5.2, NumPy 1.21.6, SciPy 1.8.1, and Torch 1.11.0.

Listing 1: Example code to generate the joint photon number distribution between two detectors. One detector receives all signal modes with efficiency  $\eta_1 = 0.8$  and noise  $\nu_1 = 1$  and the other detector receives all idler modes with  $\eta_2 = 0.9$  and  $\nu_2 = 2$ . The state is a two-mode squeezed vacuum with 16 two-mode squeezers.

```

from scipy.special import factorial
import torch
from torch.autograd import grad
from torch import tensor, zeros, ones, sqrt, exp, diag, stack, det, inverse
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import block_diag

# Set default type of torch variables and tensors to 64-bit floating point
torch.set_default_dtype(torch.float64)
torch.set_default_tensor_type(torch.DoubleTensor)

#####
#####      Set up covariance matrix of TMSV with squeezing angle = zero      #####
#####

mu = 3                # Total mean photon pair number
num_squeezers = 16   # Number of equally strong squeezers
etas = [0.8, 0.9]    # Detector efficiencies
nus = [1, 2]         # Noise parameters

r = np.arcsinh(np.sqrt(mu / num_squeezers)) # TMSV squeezing parameter
c, s = np.cosh(2*r), np.sinh(2*r)

covmat_TMSV = np.array([[c,  s,  0,  0],
                        [s,  c,  0,  0],
                        [0,  0,  c, -s],
                        [0,  0, -s,  c]]) # Covariance of a two-mode squeezer

# Set up block-diagonal covariance matrix with multiple two-mode squeezers
Gamma = tensor(block_diag(*([covmat_TMSV] * num_squeezers)))

dim = num_squeezers * covmat_TMSV.shape[0] # Dimension of the covariance matrix
d = zeros(dim)                             # The displacement vector is zero.

```

```

#####
#####          Set up differentiation parameters and Lambda-matrix          #####
#####
#####

ys = [tensor(0., requires_grad=True) for i in (0, 1)]           # Parameters y

# Set up Diagonal W matrix and Lambda matrix
W = diag(stack([eta * (1-y) for eta, y in zip(etas, ys)] * 2 * num_squeezers))
Lambda = diag(ones(dim)) + W @ (Gamma - diag(ones(dim))) / 2

#####
#####          Set up generating function          #####
#####

G = exp(- d @ (inverse(Lambda) @ (W @ d)) / 2) / sqrt(det(Lambda))
G = G * exp(sum([(y-1) * nu for nu, y in zip(nus, ys)])) # Multiply noise term

### Function to compute probability from multivariate higher-order derivatives.
### Each photon number n in the list ns corresponds to one y-parameter.
def p(ns):
    deriv = G
    for param, order in zip(ys, ns):
        for _ in range(order):
            deriv = grad(deriv, param, create_graph=True)[0]
    return deriv.detach() / np.prod(factorial(np.array(ns)))

```

Listing 2: Code to generate a 3D bar chart with the joint detection probabilities up to photon numbers of 4.

```
#####  
##### Compute probabilities on a square grid of photon numbers #####  
#####  
  
n_max = 4 # Maximum photon number to compute  
n_space = np.arange(0, n_max+1, 1)  
n1grid, n2grid = np.meshgrid(n_space, n_space)  
n1grid_fl, n2grid_fl = n1grid.flatten(), n2grid.flatten()  
  
probs = []  
for i in range(len(n1grid_fl)):  
    probs.append(p([n1grid_fl[i], n2grid_fl[i]]))  
    print(f'p({int(n1grid_fl[i])}, {int(n2grid_fl[i])}) = {probs[-1]:.3e}')  
  
#####  
##### Plot a 3D bar chart #####  
#####  
  
bar_width = 0.5  
  
fig = plt.figure(figsize=(6, 6))  
ax = fig.add_subplot(111, projection='3d')  
  
ax.bar3d(n1grid_fl - bar_width / 2, n2grid_fl - bar_width / 2,  
          np.zeros(len(n1grid_fl)), bar_width, bar_width, probs,  
          edgecolor='black')  
  
# plot cosmetics  
ax.set_xticks(n_space)  
ax.set_yticks(n_space)  
ax.set_xlabel("$n_1$")  
ax.set_ylabel("$n_2$")  
ax.set_zlabel("$p(n_1, n_2)$")  
ax.grid()  
  
plt.show()
```



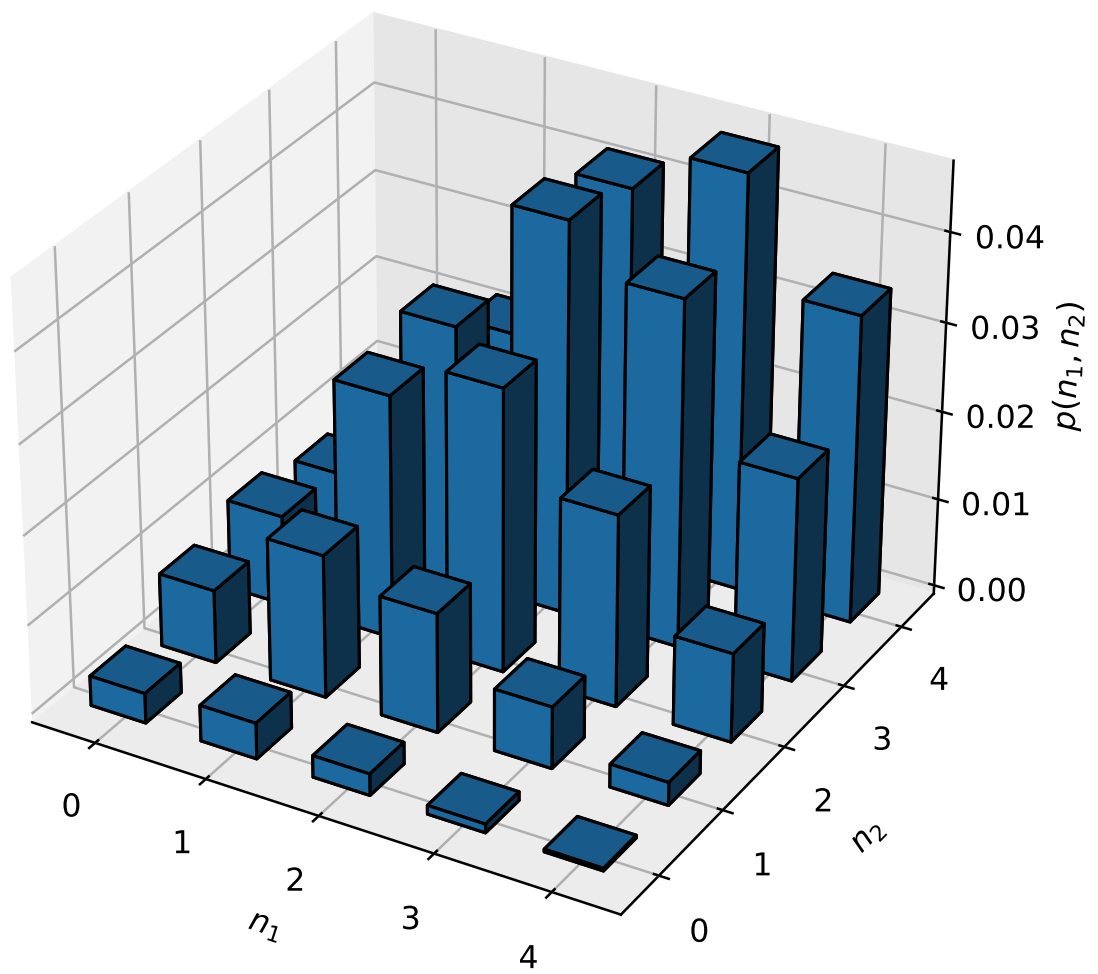


Figure 1: Joint photon number distribution generated by using the python code from listings 1 and 2.

## A. Covariance Matrix and Displacement Vector – Construction and Examples

The vector of operators  $\hat{\mathbf{q}}$  is related to the vector  $\hat{\mathbf{a}} = (\hat{a}_1, \dots, \hat{a}_s, \hat{a}_1^\dagger, \dots, \hat{a}_s^\dagger)^\top$  by

$$\hat{\mathbf{q}} = \Omega \hat{\mathbf{a}} \quad \text{with} \quad \Omega = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbb{1} & \mathbb{1} \\ -i\mathbb{1} & i\mathbb{1} \end{pmatrix}. \quad (6)$$

Unitary transformations described by Hamiltonians that are at most quadratic in the creation and annihilation operators can be written as

$$\hat{H}_1 = i\mathbf{h}^\top \hat{\mathbf{a}} \quad \text{with} \quad \mathbf{h} = \begin{pmatrix} -\boldsymbol{\alpha}^* \\ \boldsymbol{\alpha} \end{pmatrix} \quad \text{and} \quad (7)$$

$$\hat{H}_2 = \frac{1}{2} \hat{\mathbf{a}}^\dagger \mathbf{H} \hat{\mathbf{a}} \quad \text{with} \quad \mathbf{H} = \begin{pmatrix} \mathbf{X} & \mathbf{Y} \\ \mathbf{Y}^* & \mathbf{X}^* \end{pmatrix}. \quad (8)$$

Such Hamiltonians map Gaussian states  $\hat{\rho}$  to new Gaussian states  $\hat{\rho}'$  by changing the covariance matrix and displacement vector:

$$\hat{\rho}' = e^{-i\hat{H}_1} \hat{\rho} e^{i\hat{H}_1} \quad \Rightarrow \quad \boldsymbol{\Gamma}' = \boldsymbol{\Gamma} \quad \mathbf{d}' = \mathbf{d} + \Omega \mathbf{J} \mathbf{h} \quad \text{with} \quad \mathbf{J} = \begin{pmatrix} \mathbf{0} & \mathbb{1} \\ -\mathbb{1} & \mathbf{0} \end{pmatrix} \quad \text{and} \quad (9)$$

$$\hat{\rho}' = e^{-i\hat{H}_2} \hat{\rho} e^{i\hat{H}_2} \quad \Rightarrow \quad \boldsymbol{\Gamma}' = \mathbf{S} \boldsymbol{\Gamma} \mathbf{S}^\top \quad \mathbf{d}' = \mathbf{S} \mathbf{d} \quad (10)$$

The transformation matrix  $\mathbf{S}$  is given by

$$\mathbf{S} = \Omega e^{-i\mathbf{K} \mathbf{H} \Omega^\dagger} \quad \text{with} \quad \mathbf{K} = \begin{pmatrix} \mathbb{1} & \mathbf{0} \\ \mathbf{0} & -\mathbb{1} \end{pmatrix}. \quad (11)$$

Another useful transformation of  $\boldsymbol{\Gamma}$  and  $\mathbf{d}$  is the introduction of additional vacuum modes, which is simply achieved by adding two columns and rows per mode with one on the diagonal for the new  $x$  and  $p$  components. Conversely, discarding some modes, i.e. performing the partial trace over  $\hat{\rho}$ , is achieved by deleting from  $\boldsymbol{\Gamma}$  and  $\mathbf{d}$  all rows and columns associated with the traced-out subsystem. This can be used for example to model losses  $L$  for a mode by introducing an auxiliary mode containing vacuum, coupling both modes with a beam splitter with transmission  $1 - L$  and tracing out the auxiliary mode. The overall loss transformation for a single-mode is represented by the transmission matrix  $\mathbf{T} = \sqrt{1-L} \mathbb{1}$  [7]:

$$\boldsymbol{\Gamma}' = \mathbf{T} \boldsymbol{\Gamma} \mathbf{T}^\top + L \mathbb{1}, \quad \mathbf{d}' = \mathbf{T} \mathbf{d}. \quad (12)$$

The covariance matrix of a two-mode squeezed vacuum (TMSV) state for example can be calculated by setting up  $\mathbf{H}$  from eq. (8) for  $\exp(-i\hat{H}_2) = \exp(\chi \hat{a}_s^\dagger \hat{a}_i^\dagger - \chi^* \hat{a}_s \hat{a}_i)$ , calculating  $\mathbf{S}$  from eq. (11) and applying it from both sides to the covariance of the initial vacuum state,  $\boldsymbol{\Gamma} = \mathbb{1}$ , according to eq. (10). However, in practice, these steps are not necessary because the expression for the covariance matrix of TMSV can be looked up in the literature. Hence, the covariance formalism can be considered as a kit of building blocks: known expressions for  $\boldsymbol{\Gamma}$  and  $\mathbf{d}$  for basic states can be used and combined with known transformations  $\mathbf{S}$  to represent the setup of interest. The parameters for  $\boldsymbol{\Gamma}$  and  $\mathbf{d}$  for various Gaussian states and the most common transformation  $\mathbf{S}$  can be found in various publications, e.g. in Refs. [2–7, 22]. Table 1 shows  $\boldsymbol{\Gamma}$  and  $\mathbf{d}$  for a few Gaussian states and table 2 shows two important transformation matrices  $\mathbf{S}$ .

Table 1: Covariance representation of some common Gaussian states. The displacement operator is  $\hat{D}(\alpha) = \exp(\alpha\hat{a}^\dagger - \alpha^*\hat{a})$  and the squeezing operator is  $S(\chi) = \exp[(\chi\hat{a}^{\dagger 2} - \chi^*\hat{a}^2)/2]$ . The squeezing parameter for single- or two-mode squeezing is  $\chi = r e^{i\theta}$ . We use the abbreviations  $C = \cosh 2r$ ,  $S = \sinh 2r$ ,  $c = \cos \theta$ ,  $s = \sin \theta$ . The thermal state with mean photon number  $\mu_{\text{th}}$  is given by  $\hat{\rho}_{\text{th}}(\mu_{\text{th}}) = \sum_{m=0}^{\infty} \mu_{\text{th}}^m / (1 + \mu_{\text{th}})^{m+1} |m\rangle\langle m|$ .

	State	Covariance matrix and displacement vector
vacuum	$ 0\rangle$	$\Gamma = \mathbb{1}$ and $\mathbf{d} = 0$
coherent state	$ \alpha\rangle = \hat{D}(\alpha) 0\rangle$	$\Gamma = \mathbb{1}$ and $\mathbf{d} = \sqrt{2}[\text{Re}(\alpha), \text{Im}(\alpha)]^\top$
Displaced squeezed thermal state [2, 5]	$D(\alpha)S(\chi)\hat{\rho}_{\text{th}}(\mu_{\text{th}})S^\dagger(\chi)D^\dagger(\alpha)$	$\Gamma = (1 + 2\mu_{\text{th}}) \left[ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} C + \begin{pmatrix} c & s \\ s & -c \end{pmatrix} S \right]$ , $\mathbf{d} = \sqrt{2} \begin{pmatrix} \text{Re}(\alpha) \\ \text{Im}(\alpha) \end{pmatrix}$
Two-mode squeezed vacuum	$\exp(\chi\hat{a}_s^\dagger\hat{a}_i^\dagger - \chi^*\hat{a}_s\hat{a}_i) 0\rangle$	$\Gamma = \begin{pmatrix} C & Sc & 0 & Ss \\ Sc & C & Ss & 0 \\ 0 & Ss & C & -Sc \\ Ss & 0 & -Sc & C \end{pmatrix}$ and $\mathbf{d} = 0$

Table 2: Transformation matrices  $S$  representing phase rotations and beam splitters [7].

Transformation	Matrix $S$
Phase rotation $\hat{a} \rightarrow e^{-i\phi}\hat{a}$	$S = \begin{pmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{pmatrix}$
Beam splitter with transmission $T$ , coupling two modes	$S = \begin{pmatrix} \sqrt{T} & \sqrt{1-T} & 0 & 0 \\ -\sqrt{1-T} & \sqrt{T} & 0 & 0 \\ 0 & 0 & \sqrt{T} & \sqrt{1-T} \\ 0 & 0 & -\sqrt{1-T} & \sqrt{T} \end{pmatrix}$

---

## Acknowledgements

---

This research has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), under Grant No. SFB 1119–236615297.

---

## References

---

- [1] Erik FITZKE, Florian NIEDERSCHUH, and Thomas WALTHER. Generating Functions and Automatic Differentiation for Photon-Number-Resolved Simulations with Multimode Gaussian States. <https://arxiv.org/abs/2209.05330> (2022).
- [2] A. FERRARO, S. OLIVARES, and Matteo G. A. PARIS. Gaussian States in Quantum Information. Napoli Series on physics and Astrophysics. Bibliopolis 2005.
- [3] Xiang-Bin WANG *et al.* Quantum information with Gaussian states. *Physics Reports* **448** (2007), 1–111.
- [4] Christian WEEDBROOK *et al.* Gaussian quantum information. *Rev. Mod. Phys.* **84** (2012), 621–669.
- [5] S. OLIVARES. Quantum optics in the phase space. *The European Physical Journal Special Topics* **203** (2012), 3–24.
- [6] Gerardo ADESSO, Sammy RAGY, and Antony R. LEE. Continuous Variable Quantum Information: Gaussian States and Beyond. *Open Systems & Information Dynamics* **21** (2014), 1440001.
- [7] Masahiro TAKEOKA, Rui-Bo JIN, and Masahide SASAKI. Full analysis of multi-photon pair effects in spontaneous parametric down conversion based photonic quantum information processing. *New Journal of Physics* **17** (2015), 043030.
- [8] Craig S. HAMILTON *et al.* Gaussian Boson Sampling. *Phys. Rev. Lett.* **119** (2017), 170501.
- [9] Nicolás QUESADA, Juan Miguel ARRAZOLA, and Nathan KILLORAN. Gaussian boson sampling using threshold detectors. *Phys. Rev. A* **98** (2018), 062322.
- [10] Regina KRUSE *et al.* Detailed study of Gaussian boson sampling. *Phys. Rev. A* **100** (2019), 032326.
- [11] Andreas BJÖRKLUND, Brajesh GUPT, and Nicolás QUESADA. A Faster Hafnian Formula for Complex Matrices and Its Benchmarking on a Supercomputer. *ACM J. Exp. Algorithmics* **24** (2019).
- [12] N. QUESADA *et al.* Simulating realistic non-Gaussian state preparation. *Phys. Rev. A* **100** (2019), 022341.
- [13] Nicolás QUESADA. Franck-Condon factors by counting perfect matchings of graphs with loops. *The Journal of Chemical Physics* **150** (2019), 164113.
- [14] Nicolás QUESADA *et al.* Quadratic Speed-Up for Simulating Gaussian Boson Sampling. *PRX Quantum* **3** (2022), 010306.
- [15] Brajesh GUPT, Josh IZAAC, and Nicolás QUESADA. The Walrus: a library for the calculation of hafnians, Hermite polynomials and Gaussian boson sampling. *Journal of Open Source Software* **4** (2019), 1705.

- 
- [16] Charles R. HARRIS *et al.* Array programming with NumPy. *Nature* **585** (2020), 357–362.
  - [17] I. BRANDÃO, D. TANDEITNIK, and T. GUERREIRO. QuGIT: A numerical toolbox for Gaussian quantum states. *Computer Physics Communications* **280** (2022), 108471.
  - [18] Andreas GRIEWANK and Andrea WALTHER. *Evaluating Derivatives*. Second. Society for Industrial and Applied Mathematics 2008.
  - [19] Uwe NAUMANN. *The Art of Differentiating Computer Programs - An Introduction to Algorithmic Differentiation*. SIAM 2012.
  - [20] Adam PASZKE *et al.* Automatic differentiation in PyTorch. In: NIPS-W. 2017.
  - [21] Adam PASZKE *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. WALLACH *et al.* Curran Associates, Inc. 2019, 8024–8035.
  - [22] Jonatan Bohr BRASK. Gaussian states and operations – a quick reference. <https://arxiv.org/abs/2102.05748> (2021).