Documentation
Quantum Gaussian Information Toolbox

Igor Brandão*

Department of Physics, Pontifical Catholic University of Rio de Janeiro, Brazil

June 8, 2021

The numerical simulations for this projects will be carried out with the custom MATLAB Toolbox, available at gihub: `https://github.com/IgorBrandao42/Quantum-Gaussian-Information-Toolbox/`. It has evolved from its initial version to describe any multimode gaussian state, and general unconditional open quantum dynamics of gaussian states evolving under gaussian preserving Langevin and Lypaunov equations [1]. The toolbox is divided into two classes: one simulating gaussian states with methods for extracting information from them, and another for calculating the time evolution of a given gaussian state, drift and diffusion matrices.

1 gaussian_state class

An instance of this class simulates a multimode gaussian state.

Definitions

Gaussian states are continuous variable state, whose Wigner function representation in phase-space is gaussian [1]. Let us briefly lay out some definitions:

For a N-modes continuous variable state, each mode is described by the annihilation ($\hat{a}_j$) and creation ($\hat{a}_j^\dagger$) operators, obeying bosonic commutation relations. These can be conventionally arranged in a $2N$-dimensional vectorial operator $\hat{\boldsymbol{b}} = (\hat{a}_1, \hat{a}_1^\dagger, \hat{a}_2, \hat{a}_2^\dagger, \ldots)^T$ whose commutation relations can be expressed as $\left[\hat{b}_j, \hat{b}_k\right] = \Omega_{jk}$ where $j,k = 1,\ldots,2M$ and Ω is the $2N \times 2N$ symplectic form matrix given by

$$\Omega = \bigoplus_{k=1}^{M} \Omega_k \quad , \quad \Omega_k = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} . \tag{1.1}$$

From these bosonic operators, we can define the corresponding quadrature operators $\hat{x}_j = \hat{a}_j^\dagger + \hat{a}_j$ and $\hat{p}_j = i(\hat{a}_j^\dagger - \hat{a}_j)$ and once again suitably arrange them into a $2M$-dimensional vectorial operator $\hat{\boldsymbol{X}} = (\hat{x}_1, \hat{p}_1, \hat{x}_2, \hat{p}_2, \ldots)^T$. It immediately follows from the bosonic commutation relations above that the quadratures must satisfy the canonical commutation relations $\left[\hat{X}_j, \hat{X}_k\right] = 2i\,\Omega_{jk}$.

From the definitions of gaussian states, they are completely characterized by their first moments,

$$\boldsymbol{R} \equiv \langle \hat{\boldsymbol{X}} \rangle = \mathrm{tr}\Big(\rho \hat{\boldsymbol{X}}\Big) ,$$

and second moments represented by the *covariance matrix* (CM), whose entries are given by

$$V_{j,k} = \frac{1}{2}\langle \hat{X}_j \hat{X}_k + \hat{X}_k \hat{X}_j \rangle - \langle \hat{X}_j \rangle \langle \hat{X}_k \rangle . \tag{1.2}$$

Class properties

Gaussian states greatly simplify our treatment of continuous variable systems, as instead of dealing with high-dimensional density matrices/phase spaces, we need only to worry about $2N$-dimensional vectors and $2N \times 2N$ matrices. Thus, the internal variables of the gaussian_state class are

*igorbrandao@aluno.puc-rio.br

[1] At some latter moment, the toolbox will also comprehend conditional dynamics!

Table 1: Properties of the gaussian_state class

Property	Description
R	Quadratures mean values
V	Covariance matrix
Omega	Symplectic form matrix
N_modes	Number of modes

Class methods

In Table 2, we present the name and description of the methods of the gaussian_state class except one: the class constructor. It has the same name of the class and must be called to create an instance of it: a variable of type gaussian_state. There are essentially three ways the user can create such instance, dictated by the kind of input arguments:

- No arguments – default constructor returns a single mode vacuum state;
- Vector and matrix – constructor returns a multimode gaussian state with respective mean quadrature vector and covariance matrix;
- Name-pair value – the user provides a string with the name of a default single mode gaussian state and respective parameter:
  1. "vacuum" , – ;
  2. "coherent" , complex amplitude;
  3. "squeezed" , squeezing parameter;
  4. "thermal" , occupation number.

Table 2: Properties of the gaussian_state class

Method	Description
displace	Applies a displacement operator on a single mode gaussian state
squeeze	Applies a squeezing operator on a single mode gaussian state
rotate	Applies a rotation operator on a single mode gaussian state
beam_splitter	Applies a beam splitter operator on a two mode gaussian state
two_mode_squeezing	Applies a two mode squeezing operator on a two mode gaussian state
tensor_product	Tensor product of two gaussian states
partial_trace	Partial trace over a some modes of a gaussian_state
only_modes	Partial trave over all but some modes of a gaussian_state
purity	Purity of a gaussian state
squeezing_degree	Degree of squeezing in a single mode gaussian state
symplectic_eigenvalues	Sympletic eigenvalues of the covariance matrix of a gaussian state
von_Neumann_Entropy	von Neumann entropy for a multipartite gaussian state
mutual_information	Mutual information for a multipartite gaussian state
wigner	Wigner function for a single mode gaussian state in some 2D grid
fidelity	Quantum Fidelity between the two multipartite gaussian states
occupation_number	Occupation number for a single mode gaussian state
logarithmic_negativity	Logarithmic negativity for a bipartition of a multipartite gaussian state
duan_criteria	LHS of the Duan criteria for a bipartition of a multipartite gaussian state

2 time_evolution class

An instance of this class perform a time evolution on some initial gaussian_state according to a unconditional open quantum dynamics dictated by a set of quantum Langevin and Lyapunov equations.

Definitions

Langevin equation – We assume that the time evolution of the quadrature vector are dictated by a set of quantum Langevin equations of the form

$$\dot{\hat{\boldsymbol{X}}} = A(t)\hat{\boldsymbol{X}} + \hat{\boldsymbol{N}}(t)\,,$$

where $A(t)$ is the drift matrix and D is the noise vector operator. We use take the expectation value of this equation to calculate the of the first moments of time evolved gaussian state dictated by

$$\dot{\boldsymbol{R}} = A(t)\boldsymbol{R} + \boldsymbol{N}(t)\,,$$

where N is the mean noise vector.

Lyapunov equation – A direct consequence of the quantum Langevin equations above is that the time evolution of the covariance matrix of the initial state is given by a Lyapunov equation, with D the diffusion matrix,

$$\dot{V} = A(t)V + VA(t)^T + D\,,$$

Semi-classical Langevin equation – We may consider a semi-classical description of the system by performing a Monte Carlos simulation of the Langevin equations for the mean quadratures. At each iteration of this simulation, we perform a Euler-Maryuama integration of the semi-classical stochastic differential Langevin equation. The mean values of the noises are encompassed in $\boldsymbol{N}$ and its correlations (noise amplitude) are described in the diagonal elements of the diffusion matrix D. Analogously, the initial conditions are obtained from the mean values of the initial state $\boldsymbol{R}_0$ and the variances of their distribution in phase-space are given by the diagonal elements of its covariance matrix V_0.

Class properties

Table 3: Properties of the time_evolution class

Property	Description
A	Drift matrix
D	Diffusion matrix
N	Mean noise vector
t	Array with timestamps for the simulation
R	Array with mean quadratures for each timestamp
V	Cell with covariance matrix for each timestamp
state	Gaussian state for each timestamp
R_semi_classical	Array with semi-classical mean quadratures
is_stable	Boolean telling if the system is stable or not
N_time	Length of time array
Size_matrices	Size of covariance, diffusion and drift matrices

Class methods

In Table 4, we present the name and description of the methods of the time_evolution class except one: the class constructor. It has the same name of the class and must be called to create an instance of it: a variable of type time_evolution. There is only one way to call the constructor: passing as arguments the parameters for the time evolution described above:

1. A – the drift matrix

2. D – the diffusion matrix
3. $\boldsymbol{N}$ – the mean noise vector
4. Initial gaussian state (gaussian_state)

Table 4: Properties of the time_evolution class

Method	Description
langevin	Solves the Langevin equation for the time evolved mean quadratures
lyapunov	Solves the Lyapunov equation for the time evolved covariance matrix
build_states	Builds the array of gaussian_state with time evolved mean values and covariance matrices
run	Calls the three previous methods in order
langevin_semi_classical	Solve the semi-classical Langevin equation for the time evolved mean quadratures
steady_state	Calculates the steady state gaussian_state (at the moment it only works for constant drift matrix)

2.1 Example of usage

Let us see a basic example of usage of the Toolbox. Consider a harmonic oscillator in a thermal state, we can easilly define its gaussian state through

```
nbar_0    = 1.0577e+05;                          % Initial particle occupation number
initial_state = gaussian_state("thermal", nbar_0); % Initial state
```

We may apply gaussian unitaries to this state. Maybe we want to squeeze and rotate it in phase space, which are done through

```
initial_state.squeeze(3);                        % Squeeze gaussian state
initial_state.rotate(-pi/4);                     % Rotate gaussian state
```

Now that we have defined some state for our particle, we may wonder what will happen to is once it starts to evolve in time. We now define the dynamics for the particle

```
omega = 2*pi*197e+3;                             % Particle natural frequency [Hz]
gamma = 2*pi*881.9730;                           % Damping constant [Hz] at 1.4 mbar pressure
nbar_env = 3.1731e+07;                           % Environmental    occupation number

A =[[     0    ,  +omega ];                      % Drift matrix for harmonic potential
    [ -omega ,  -gamma ]];

D = diag([0, 2*gamma*(2*nbar_env+1)]);           % Diffusion matrix
N = zeros(2,1);                                  % Mean noise vector
```

Now we can proceed to perform the numerical simulation of the time evolution for some timestamps

```
t = linspace(0, 2*pi/omega, 1e4);                % Timestamps for simulation

simulation = time_evolution(A, D, N, initial_state); % Create simulation instance!
states = simulation.run(t);                      % Simulate and retrieve time evolved states
```

The variable 'states' is an array of gassian_state with the time evolution of our system. From it we can use the methods described in Table 2 and study, for example, how the mean number of photon has evolved in time:

```
nbar = zeros(size(t));                           % Variable to store occupation numbers

for i=1:1:length(states)                         % For each time evolved state,
nbar(i) = states(i).occupation_number();         % calculate its occupation number
end
```

References

[1] 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.*, 84:621–669, May 2012.