Numerical Simulation of Quantum Squeezing in Parametric Downconversion

Orson Kirsch

June 3, 2025

Abstract

This project presents a computational framework for simulating the generation of quantum squeezed states via parametric downconversion (PDC). We implement and compare two prominent phase-space methods: the Truncated Wigner (TW) representation and the Positive P-representation (+P). The simulations model the quantum dynamics of the interacting signal and pump modes, allowing for the analysis of quadrature squeezing, photon number evolution, and pump depletion under various physical regimes. The software is structured to facilitate configurable simulation runs, robust data management, and automated generation of publication-quality plots for key physical observables. This work provides insights into the efficacy and computational characteristics of the TW and +P methods for modeling non-classical light generation in nonlinear optical systems.

Contents

1 Introduction

The generation and manipulation of non-classical states of light are cornerstones of modern quantum optics and quantum technologies. Among these, squeezed states of light, characterized by a reduction in quantum noise for one field quadrature below the vacuum level (Heisenberg uncertainty limit), are of paramount importance. Parametric downconversion (PDC) in nonlinear optical media is a widely used and effective process for generating such states.

This project focuses on the theoretical modeling and numerical simulation of squeezed state generation in PDC. Due to the inherently quantum nature and often complex dynamics of these systems, numerical simulations provide an invaluable tool for understanding system behavior, optimizing experimental parameters, and exploring regimes that may be analytically intractable.

1.1 Parametric Downconversion and Squeezed States

In parametric downconversion, a strong pump photon interacts with a nonlinear crystal, leading to its annihilation and the creation of two lower-frequency photons, typically referred to as signal and idler photons. Energy and momentum are conserved in this process. When the signal and idler modes are degenerate, the process is known as degenerate parametric downconversion, which is particularly effective for generating single-mode squeezed vacuum or bright squeezed states.

The interaction Hamiltonian for degenerate PDC (considering a quantized pump mode $\hat{a}_2$ and a signal mode $\hat{a}_1$) involves a nonlinear coupling. The strength of this interaction is characterized by the parameter κ (defined under `simulation_params.kappa` in the configuration file `simulation_project/config.yaml`).

Field quadratures are defined as $\hat{X}_1 = (\hat{a} + \hat{a}^\dagger)/2$ and $\hat{X}_2 = (\hat{a} - \hat{a}^\dagger)/(2i)$. A state is squeezed if the variance of one quadrature, $V(\hat{X}_i) = \langle \hat{X}_i^2 \rangle - \langle \hat{X}_i \rangle^2$, is less than the vacuum variance (typically normalized to 0.25, as specified by `vacuum_noise_quad_variance` in `simulation_project/config.yaml`).

1.2 Simulation Methodologies

To model these quantum systems, this project employs two powerful phase-space representations:

1.2.1 Truncated Wigner (TW) Representation

The Wigner representation maps quantum states to a quasi-probability distribution in phase space. For many systems, the evolution equation for the Wigner function can be approximated, leading to classical-like stochastic differential equations (SDEs) or, as implemented here, deterministic Ordinary Differential Equations (ODEs) for complex phase-space variables α_1, α_2 (see `simulation_project/simulation_core/tw_simulation.py`). Quantum expectation values are obtained by averaging functions of these variables over many trajectories. Initial conditions are sampled according to the Wigner distribution of the initial state, for instance, by adding specific noise terms for vacuum states (handled by `generate_complex_gaussian_noise_tw` in `simulation_project/simulation_core/utils.py`).

1.2.2 Positive P-Representation (+P)

The Positive P-representation is a more general phase-space method. It maps quantum master equations to SDEs for a positive and normalizable probability distribution by doubling the phase-space dimension, representing each mode with two complex variables (e.g., α and α^+). This project implements the +P method for the PDC system, solving SDEs for variables $(\alpha_1, \alpha_1^+, \alpha_2, \alpha_2^+)$ (detailed in `simulation_project/simulation_core/pp_simulation.py`).

2 Project Structure

The project is organized as follows (based on the provided file structure):

```
quantum_squeezing/
 README.md
 simulation_project/
     config.yaml
     run_simulations.py
```

```
data_handling/
   storage.py
simulation_core/
   pp_simulation.py
   tw_simulation.py
   utils.py
plotting/
    plot_max_squeezing_vs_n.py
    plot_photon_tracker.py
    plot_pump_depletion.py
    plot_seed_variance.py
    plot_signal_photons.py
    plot_squeezing.py
```

- `README.md`: A brief project overview.
- `simulation_project/config.yaml`: Central configuration file.
- `simulation_project/run_simulations.py`: Main script for execution.
- `simulation_project/data_handling/storage.py`: Data saving/loading utilities.
- `simulation_project/simulation_core/`: Contains the physics simulation engines (`tw_simulation.py`, `pp_simulation.py`, `utils.py`).
- `simulation_project/plotting/`: Scripts for generating plots from simulation data.

3 Configuration (`config.yaml`)

The simulation behavior is extensively controlled by the file `simulation_project/config.yaml`. Key parameters include:

- `simulation_params`:
  - `kappa`: Nonlinear coupling strength.
  - `num_trajectories_main`: Trajectories per simulation run.
  - `global_seeds_plot4`, `n_scan_seeds`: Seeds for reproducibility and statistical analysis.
  - `regime_physical_t_end`, `n_scan_t_ends`: Simulation end times.
  - `num_simulation_points`: Output time points.
- `photon_regimes`: Initial pump photon numbers for 'Low', 'Medium', 'High' scenarios.
- `data_handling`: Settings for data storage (directory, re-run policy).
- `plotting`: Flags for plot generation and aesthetic parameters (limits, styles).

This centralized configuration facilitates systematic parameter scans.

4 Core Simulation Logic

The quantum dynamics of the PDC process are simulated via two distinct methods implemented in the `simulation_project/simulation_core/` directory.

4.1 Truncated Wigner (TW) Simulation

Implemented in `simulation_project/simulation_core/tw_simulation.py`.

4.1.1 Governing Equations

The evolution of complex phase-space variables for signal (α_1) and pump (α_2) modes is described by:

$$\frac{d\alpha_1}{dt} = \kappa\alpha_1^*\alpha_2 \tag{1}$$

$$\frac{d\alpha_2}{dt} = -\frac{\kappa}{2}\alpha_1^2 \tag{2}$$

These ODEs are defined in the Python function `pdc_full_system_tw` and solved using `scipy.integrate.solve_ivp`.

4.1.2 Initial Conditions

Initial conditions incorporate vacuum fluctuations:

$$\alpha_1(0) = \eta_1 \tag{3}$$

$$\alpha_2(0) = \sqrt{n} + \eta_2 \tag{4}$$

where n is the initial mean pump photon number (from `config.yaml`), and η_1, η_2 are complex Gaussian noises with variance 0.5 (from `generate_complex_gaussian_noise_tw` in `simulation_project/simulation_core/utils.py`).

4.1.3 Observables

Expectation values are averaged over `num_trajectories_main` (from `config.yaml`). Quadratures (X_1, X_2) are calculated as per `calc_quad_tw` in `simulation_project/simulation_core/utils.py`. Variances are $V(X_i)_{\text{TW}} = \langle X_i^2\rangle - \langle X_i\rangle^2$. Average photon number for mode j: $\langle\hat{n}_j\rangle_{\text{TW}} = \langle|\alpha_j|^2 - 1/2\rangle$.

4.2 Positive P-Representation (+P) Simulation

Implemented in `simulation_project/simulation_core/pp_simulation.py`.

4.2.1 Governing Stochastic Differential Equations (SDEs)

Dynamics for $\alpha_1, \alpha_1^+, \alpha_2, \alpha_2^+$ are (from `pdc_pplus_step_numba`):

$$d\alpha_1 = (\kappa\alpha_1^+\alpha_2)dt + \sqrt{\kappa\alpha_2}dW_1(t) \tag{5}$$

$$d\alpha_1^+ = (\kappa\alpha_1\alpha_2^+)dt + \sqrt{\kappa\alpha_2^+}dW_2(t) \tag{6}$$

$$d\alpha_2 = -\frac{\kappa}{2}\alpha_1^2 dt \tag{7}$$

$$d\alpha_2^+ = -\frac{\kappa}{2}(\alpha_1^+)^2 dt \tag{8}$$

where $dW_1(t), dW_2(t)$ are independent real Wiener increments. The complex diffusion terms $\sqrt{\kappa\alpha_j}$ are handled as $\sqrt{\kappa\alpha_j + 0j}$.

4.2.2 Initial Conditions

Signal mode in vacuum, pump in a coherent state (mean photon n):

$$\alpha_1(0) = 0, \quad \alpha_1^+(0) = 0 \tag{9}$$

$$\alpha_2(0) = \sqrt{n}, \quad \alpha_2^+(0) = \sqrt{n} \tag{10}$$

(as set in the function `run_pplus_simulation_parallel`).

4.2.3 Observables

Averages over trajectories. For operator $\hat{A}(\hat{a}, \hat{a}^\dagger)$, its +P average is $\langle A(\alpha, \alpha^+)\rangle$. Quadrature variances are derived from normally ordered moments. Average photon number: $\langle\hat{n}_j\rangle_{+P} = \text{Re}(\langle\alpha_j\alpha_j^+\rangle)$.

4.2.4 Numerical Implementation

SDEs are integrated via an Euler-Maruyama scheme in `pdc_pplus_step_numba` (Numba JIT-compiled). Parallel execution is managed by `run_pplus_simulation_parallel` using Python's `multiprocessing` module.

5 Data Handling

The module `simulation_project/data_handling/storage.py` manages data:

- `get_params_hash`: MD5 hash for unique run identification.
- `generate_filename`: Descriptive filenames including parameters and hash.
- `save_simulation_data`: Saves results to compressed `.npz` files.
- `load_simulation_data`: Loads data, or returns `None` if absent/corrupt.

This avoids re-computation if `force_rerun_all` in `config.yaml` is false.

6 Execution and Workflow

The main script `simulation_project/run_simulations.py` performs:

1. **Configuration Loading**: From `config.yaml`.
2. **Regime Iteration**: Loops 'Low', 'Medium', 'High' regimes and 'TW', 'PP' methods.
3. **Seed Iteration**: Runs simulations for multiple global seeds (from `config.yaml`) for statistical analysis.
4. **N-Scan Simulation**: If `n_scan_active` (in `config.yaml`) is true, scans initial pump photon numbers (`n_scan_values`).
5. **Data Management**: Checks for existing data or runs simulation, then saves results.
6. **Plot Generation**: Calls plotting functions based on flags in `config.yaml`.

7 Plotting and Visualization

Scripts in `simulation_project/plotting/` generate Matplotlib visualizations:

- **Plot 1: Squeezing Comparison (`plot_squeezing.py`)**: Time evolution of signal quadrature variances ($V(X_1), V(X_2)$) in dB. Compares TW and +P. Can show error bands for +P $V(X_2)$ from seed runs.
- **Plot 2: Pump Depletion (`plot_pump_depletion.py`)**: Pump photon number proxy ($|\langle\alpha_2\rangle|^2$) over time.
- **Plot 3: Photon Tracker (`plot_photon_tracker.py`)**: Average actual photon numbers ($\langle\hat{n}_1\rangle, \langle\hat{n}_2\rangle$) and $2\langle\hat{n}_2\rangle + \langle\hat{n}_1\rangle$.
- **Plot 4: Seed Variance (`plot_seed_variance.py`)**: X_2 variance (dB) from multiple individual seed runs for TW and +P across regimes.
- **Signal Photon Proxy (`plot_signal_photons.py`)**: Signal photon number proxy ($|\langle\alpha_1\rangle|^2$) over time.
- **Max Squeezing vs. N (`plot_max_squeezing_vs_n.py`)**: Max X_2 squeezing (dB) vs. initial pump photon number n, typically for +P with SEM error bars. Uses `n_scan_reliability_std_threshold_linear` from `config.yaml` for data filtering.

Plot aesthetics are controlled by `config.yaml`.

8 Conclusion

This project successfully developed a Python-based simulation framework for quantum squeezing in PDC using Truncated Wigner and Positive P-representation methods. The framework allows flexible configuration, efficient execution (leveraging parallelism and Numba), and automated generation of analytical plots. It serves as a valuable tool for exploring PDC quantum dynamics and comparing theoretical methods.

9 Future Work

Potential extensions include:

- Incorporating optical loss mechanisms.
- Modeling non-degenerate PDC for entangled pair generation.
- Including spatial mode effects.
- Investigating higher-order nonlinearities.
- Exploring advanced SDE solvers.
- Applying outputs to quantum metrology models.
- Developing a GUI for enhanced usability.

10 Dependencies

- NumPy
- SciPy
- Matplotlib
- PyYAML
- Numba
- multiprocessing (standard library)

A Example Configuration Snippet (`config.yaml`)

Listing 1: Snippet from config.yaml

```yaml
simulation_params:
  kappa: 0.1
  num_trajectories_main: 2500

  num_seed_runs_plot4: 100
  global_seeds_plot4: [
    42, 123, 789, # ... more seeds
  ]

  regime_physical_t_end:
    Low: 30.0
    Medium: 15.0
    High: 15.0

  n_scan_active: true
  n_scan_values: [1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95,
      ↪ 100]
  n_scan_seeds: [2001, 2002, # ... more seeds for n_scan
  ]
  n_scan_t_ends:
    1: 30.0
    5: 25.0
    # ... more t_ends for n_scan values

photon_regimes:
```

```
    Low: 1
    Medium: 20
    High: 100

  plotting:
    plot_dir: "plots"
    generate_plot1_squeezing: true
    # ... other plot flags ...
    vacuum_noise_quad_variance: 0.25 # Corresponds to V(X_vac) = 1/4
```