MDI-QRNG SDP 求解器逻辑流程与函数调用详解

本文档详细描述 `src/mdiqrng_sdp_solver.py` 的程序逻辑、SDP求解流程以及各函数的调用顺序。

1. 整体架构概览

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                           mdiqrng_sdp_solver.py                             │
├─────────────────────────────────────────────────────────────────────────────┤
│  类: MDIQRNG_SDP_Solver                                                      │
│  ├── __init__()           # 初始化求解器                                      │
│  ├── _setup_discretization_boundaries()  # 设置离散化边界                      │
│  ├── _compute_input_states()             # 计算量子态                          │
│  ├── _compute_conditional_probabilities() # 计算条件概率                       │
│  ├── _compute_conditional_probability()   # 单个概率计算                       │
│  ├── get_joint_state()                   # 获取联合态向量                      │
│  ├── get_joint_density_matrix()          # 获取密度矩阵                        │
│  ├── use_entangled_measurement_probabilities()  # 切换到纠缠测量              │
│  └── solve()                             # 核心SDP求解                         │
│                                                                             │
│  便捷函数:                                                                   │
│  ├── compute_guessing_probability()      # 一键计算G                          │
│  └── optimize_mu()                       # 参数优化                           │
└─────────────────────────────────────────────────────────────────────────────┘
```

2. 完整执行流程图

```
用户代码
    │
    ▼
┌─────────────────────────────────────────────────────────────────┐
│  Step 1: 创建求解器实例                                          │
│  solver = MDIQRNG_SDP_Solver(n=2, mu=0.15, boundary=3.0)        │
└─────────────────────────────────────────────────────────────────┘
    │
    │  触发 __init__()
    │
    ▼
┌─────────────────────────────────────────────────────────────────┐
│  __init__(n, mu, boundary, verbose)                             │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ 1. 存储参数: n, mu, boundary                              │   │
│  │ 2. 计算派生参数:                                          │   │
│  │    - delta = e^(-2μ)  (相干态内积)                        │   │
│  │    - n_e = n²         (Eve输出数)                         │   │
│  │    - dim = 4          (希尔伯特空间维度)                  │   │
│  └──────────────────────────────────────────────────────────┘   │
│                          │                                       │
│                          ▼                                       │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ 调用 _setup_discretization_boundaries()                   │   │
│  │ 设置 c_bounds 和 d_bounds                                 │   │
│  └──────────────────────────────────────────────────────────┘   │
│                          │                                       │
│                          ▼                                       │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ 调用 _compute_input_states()                              │   │
│  │ 计算 psi_A, psi_B, s1_map, s2_map                         │   │
│  └──────────────────────────────────────────────────────────┘   │
│                          │                                       │
│                          ▼                                       │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ 调用 _compute_conditional_probabilities()                 │   │
│  │ 填充 p_ab_given_xy 数组 (高斯概率)                        │   │
│  └──────────────────────────────────────────────────────────┘   │
└─────────────────────────────────────────────────────────────────┘
    │
```

```
        │  (可选) 切换到纠缠测量概率
        │
        ▼
┌────────────────────────────────────────────────────────────┐
│  Step 2: 选择概率模型 (可选)                                 │
│  solver.use_entangled_measurement_probabilities(noise=0.2)  │
└────────────────────────────────────────────────────────────┘
        │
        │  覆盖 p_ab_given_xy
        │
        ▼
┌────────────────────────────────────────────────────────────┐
│  Step 3: 求解SDP                                             │
│  results = solver.solve(x_star=0, y_star=0, solver="MOSEK")  │
└────────────────────────────────────────────────────────────┘
        │
        │  进入 solve() 函数
        │
        ▼
┌────────────────────────────────────────────────────────────┐
│  solve(x_star, y_star, solver, epsilon)                      │
│  ┌──────────────────────────────────────────────────────┐  │
│  │ Phase A: 定义决策变量                                  │  │
│  │   - M_tilde[(a,b,e)]: n×n×n² 个 4×4 PSD矩阵           │  │
│  │   - M_A[(a,e)]: n×n² 个 2×2 PSD矩阵                   │  │
│  │   - M_B[(b,e)]: n×n² 个 2×2 PSD矩阵                   │  │
│  │   - p_e: n² 维非负向量                                 │  │
│  └──────────────────────────────────────────────────────┘  │
│                           │                                  │
│                           ▼                                  │
│  ┌──────────────────────────────────────────────────────┐  │
│  │ Phase B: 添加约束条件 (7类)                            │  │
│  │   1. 观测一致性约束                                    │  │
│  │   2. POVM正定性 (PSD声明)                              │  │
│  │   3. 无信号约束 (Alice→Bob)                            │  │
│  │   4. 无信号约束 (Bob→Alice)                            │  │
│  │   5. Bob局部POVM归一化                                 │  │
│  │   6. Alice局部POVM归一化                               │  │
│  │   7. Eve概率归一化                                     │  │
│  └──────────────────────────────────────────────────────┘  │
│                           │                                  │
│                           ▼                                  │
│  ┌──────────────────────────────────────────────────────┐  │
```

```
│   │ Phase C: 定义目标函数                                   │   │
│   │   G = Σ_{a,b} Tr[M_tilde(a,b,e=ab) · ρ_{x*,y*}]        │   │
│   └────────────────────────────────────────────────────────┘   │
│                            │                                    │
│                            ▼                                    │
│   ┌────────────────────────────────────────────────────────┐   │
│   │ Phase D: 调用求解器                                     │   │
│   │   problem.solve(solver=MOSEK/SCS/CVXOPT)               │   │
│   └────────────────────────────────────────────────────────┘   │
│                            │                                    │
│                            ▼                                    │
│   ┌────────────────────────────────────────────────────────┐   │
│   │ Phase E: 返回结果                                       │   │
│   │   {'status', 'optimal_value', 'M_tilde', 'p_e', ...}   │   │
│   └────────────────────────────────────────────────────────┘   │
└────────────────────────────────────────────────────────────────┘
    │
    ▼
┌────────────────────────────────────────────────────────────────┐
│  Step 4: 处理结果                                                │
│  G = results['optimal_value']                                   │
│  H_min = -log2(G)  # 可认证随机性                                │
└────────────────────────────────────────────────────────────────┘
```

3. 函数调用详解

3.1 初始化阶段

`__init__(n, mu, boundary, verbose)`

调用时机：创建 `MDIQRNG_SDP_Solver` 实例时

功能：

1. 存储用户参数
2. 计算派生物理量
3. 依次调用三个初始化子函数

调用链：

```
__init__()
    ├── _setup_discretization_boundaries()
    ├── _compute_input_states()
    └── _compute_conditional_probabilities()
            └── _compute_conditional_probability() × (n×n×2×2) 次
```

`_setup_discretization_boundaries()`

调用时机： `__init__` 内部自动调用

功能：根据参数 `n` 和 `boundary` 设置测量离散化边界

算法逻辑：

```python
if n == 2:
    # 二分: [-∞, 0, +∞]
    c_bounds = [-∞, 0, +∞]
    d_bounds = [-∞, 0, +∞]
else:
    # n分: [-∞, -B, ..., +B, +∞]
    inner = linspace(-boundary, +boundary, n-1)
    c_bounds = [-∞] + inner + [+∞]
    d_bounds = [-∞] + inner + [+∞]
```

输出：

- `self.c_bounds`：X_+ 测量的边界数组
- `self.d_bounds`：P_- 测量的边界数组

为什么需要：CV Bell测量产生连续输出，需要离散化为 n 个区间才能进行SDP分析。

`_compute_input_states()`

调用时机： `__init__` 内部自动调用

功能：根据 tex 文件 Section 4 计算4种输入态的向量表示

数学公式：

```
δ = e^(-2μ)

Alice态 (2维):
  |ψ_A(x=0)⟩ = |α⟩  = (1, 0)ᵀ
  |ψ_A(x=1)⟩ = |-α⟩ = (δ, √(1-δ²))ᵀ

Bob态 (2维):
  |ψ_B(y=0)⟩ = |α⟩  = (1, 0)ᵀ
  |ψ_B(y=1)⟩ = |-α⟩ = (δ, √(1-δ²))ᵀ
```

输出：

- `self.psi_A`：Alice的2个单模态 {0: array, 1: array}
- `self.psi_B`：Bob的2个单模态 {0: array, 1: array}
- `self.s1_map`：$x \to s_1$ 映射 {0: +1, 1: -1}
- `self.s2_map`：$y \to s_2$ 映射 {0: +1, 1: -1}

为什么需要：SDP约束和目标函数需要量子态的密度矩阵表示。

`_compute_conditional_probabilities()`

调用时机： `__init__` 内部自动调用

功能：计算所有条件概率 p(a,b|x,y) 并存入4维数组

调用：内部调用 `_compute_conditional_probability()` 共 n×n×2×2 次

输出：

- `self.p_ab_given_xy`：shape=(n, n, 2, 2) 的概率数组

`_compute_conditional_probability(k, l, s1, s2)`

调用时机：被 `_compute_conditional_probabilities()` 调用

功能：计算单个条件概率 $P((k,l)|s_1,s_2)$

数学公式 (tex Eq.107)：

```
P((k,l)|s₁,s₂) = ¼ × [erf((c_k/√2) - s₁√μ - s₂√μ) - erf((c_{k-1}/√2) - s₁√μ - s₂√μ)]
                  × [erf(d_l/√2) - erf(d_{l-1}/√2)]
```

特殊处理：

- 边界为 ±∞ 时，erf(±∞) = ±1

3.2 辅助函数

`get_joint_state(x, y)`

功能：计算联合态向量 |ψ_x⟩ ⊗ |ψ_y⟩

实现：

```
return np.kron(self.psi_A[x], self.psi_B[y])  # 4维向量
```

返回：4维 numpy 数组

`get_joint_density_matrix(x, y)`

功能：计算联合密度矩阵 ρ_{xy} = |ψ⟩⟨ψ|

实现：

```
psi = self.get_joint_state(x, y)
return np.outer(psi, psi.conj())  # 4×4 矩阵
```

返回：4×4 numpy 数组

调用场景：

- `solve()` 中构建观测一致性约束
- `solve()` 中构建目标函数
- `use_entangled_measurement_probabilities()` 中计算POVM概率

3.3 概率模型切换

`use_entangled_measurement_probabilities(noise_param)`

调用时机：用户在 `solve()` 之前手动调用（可选）

功能：用纠缠测量概率替换默认的高斯概率

为什么需要：

- 高斯概率具有乘积结构 → G = 1（无随机性）
- 纠缠测量打破乘积结构 → G < 1（可认证随机性）

算法流程：

```
1. 定义Bell态基底
   |Φ⁺⟩ = (|00⟩ + |11⟩)/√2
   |Φ⁻⟩ = (|00⟩ - |11⟩)/√2
   |Ψ⁺⟩ = (|01⟩ + |10⟩)/√2
   |Ψ⁻⟩ = (|01⟩ - |10⟩)/√2

2. 构造POVM元素
   if n == 2:
       # 使用Bell态投影器（效果最好）
       E₀ = (1-ε)|Φ⁺⟩⟨Φ⁺| + ε|Ψ⁺⟩⟨Ψ⁺|
       E₁ = (1-ε)|Φ⁻⟩⟨Φ⁻| + ε|Ψ⁻⟩⟨Ψ⁻|
   else:
       # 使用SIC-POVM-like构造
       生成n²个多样化纠缠态

3. 归一化确保 Σ E_{ab} = I

4. 计算新概率
   p(a,b|x,y) = Tr[E_{ab} · ρ_{xy}]

5. 更新 self.p_ab_given_xy
```

输出：

- 覆盖 `self.p_ab_given_xy`
- 存储 `self._entangled_POVM`（供调试）

3.4 核心求解函数

`solve(x_star, y_star, solver, epsilon)`

调用时机：用户手动调用

功能：构建并求解SDP问题，返回最大猜测概率 G

详细流程：

Phase A: 定义决策变量

变量	维度	数量	说明
`M_tilde[(a,b,e)]`	4×4 PSD	$n^2 \times n^2$	Eve的联合POVM
`M_A[(a,e)]`	2×2 PSD	$n \times n^2$	Alice的边缘POVM
`M_B[(b,e)]`	2×2 PSD	$n \times n^2$	Bob的边缘POVM
`p_e`	标量	n^2	Eve的概率分布

代码：

```
M_tilde = {(a,b,e): cp.Variable((4,4), PSD=True)
           for a,b,e in product(range(n), range(n), range(n_e))}
M_A = {(a,e): cp.Variable((2,2), PSD=True) ...}
M_B = {(b,e): cp.Variable((2,2), PSD=True) ...}
p_e = cp.Variable(n_e, nonneg=True)
```

Phase B: 添加约束条件

约束1：观测一致性

```
对所有 x,y,a,b:
  Σ_e Tr[M_tilde(a,b,e) · ρ_{xy}] = p(a,b|x,y)

如果 epsilon > 0:
  |Σ_e Tr[...] - p(a,b|x,y)| ≤ epsilon
```

约束2：POVM正定性

```
M_tilde(a,b,e) ≽ 0  (通过 PSD=True 声明)
M_A(a,e) ≽ 0
M_B(b,e) ≽ 0
```

约束3：无信号 (Alice→Bob)

```
对所有 b,e:
  Σ_a M_tilde(a,b,e) = I_A ⊗ M_B(b,e)
```

约束4：无信号 (Bob→Alice)

```
对所有 a,e:
  Σ_b M_tilde(a,b,e) = M_A(a,e) ⊗ I_B
```

约束5：Bob局部归一化

```
对所有 e:
  Σ_b M_B(b,e) = p(e) · I_B
```

约束6：Alice局部归一化

```
对所有 e:
  Σ_a M_A(a,e) = p(e) · I_A
```

约束7：Eve概率归一化

```
Σ_e p(e) = 1
```

Phase C: 定义目标函数

数学表达式：

```
G = Σ_{a,b} Tr[M_tilde(a, b, e=(a,b)) · ρ_{x*,y*}]
```

关键点： `e = (a,b)` 表示 Eve 的猜测与实际输出一致

代码实现：

```python
objective = 0
for a in range(n):
    for b in range(n):
        e = a * n + b  # 将(a,b)映射到一维索引
        objective += cp.trace(M_tilde[(a,b,e)] @ rho_star)
```

Phase D: 调用求解器

```python
problem = cp.Problem(cp.Maximize(objective), constraints)
problem.solve(solver=cp.MOSEK)  # 或 SCS, CVXOPT
```

Phase E: 返回结果

```python
return {
    'status': problem.status,      # 'optimal', 'infeasible', etc.
    'optimal_value': problem.value, # G 值
    'M_tilde': {...},              # 最优POVM元素
    'M_A': {...},
    'M_B': {...},
    'p_e': [...]                   # Eve的概率分布
}
```

4. 数据流向图

```
                    ┌──────────────┐
                    │   用户参数    │
                    │ n, μ, boundary│
                    └──────┬───────┘
                           │
                           ▼
              ┌────────────────────────┐
              │   _setup_boundaries    │
              │ 计算 c_bounds, d_bounds│
              └────────────┬───────────┘
                           │
                           ▼
              ┌────────────────────────┐
              │ _compute_input_states  │
              │ 计算 ψ_A, ψ_B (2D态)   │
              └────────────┬───────────┘
                           │
         ┌─────────────────┼─────────────────┐
         │                 │                 │
         ▼                 ▼                 ▼
┌─────────────┐   ┌─────────────┐   ┌─────────────┐
│get_joint_   │   │get_joint_   │   │_compute_    │
│state(x,y)   │   │density_     │   │conditional_ │
│  → 4D向量   │   │matrix(x,y)  │   │probability  │
└─────────────┘   │  → 4×4 ρ    │   │ → p(a,b|x,y)│
                  └──────┬──────┘   └──────┬──────┘
                         │                 │
                         │                 ▼
                         │        ┌─────────────────┐
                         │        │ p_ab_given_xy   │
                         │        │ shape=(n,n,2,2) │
                         │        └────────┬────────┘
                         │                 │
                         │       ┌─────────┴──────────┐
                         │       │                    │
                         │       ▼                    ▼
                         │ ┌─────────────┐    ┌─────────────┐
                         │ │ 高斯概率     │    │ 纠缠测量概率 │
                         │ │ (默认)       │    │ (可选切换)   │
                         │ │ G → 1       │    │ G < 1       │
```

```
               │  └──────────┘   └──────────┘
               │         │           │
               │         └─────┬─────┘
               │               │
               ▼               ▼
          ┌──────────────────────────┐
          │       solve()            │
          │  ┌────────────────────┐  │
          │  │ 构建SDP:           │  │
          │  │ - 变量: M_tilde, p_e │  │
          │  │ - 约束: 7类         │  │
          │  │ - 目标: max G       │  │
          │  └────────────────────┘  │
          │           │              │
          │           ▼              │
          │  ┌────────────────────┐  │
          │  │ MOSEK/SCS 求解器    │  │
          │  └────────────────────┘  │
          │           │              │
          │           ▼              │
          │  ┌────────────────────┐  │
          │  │ 返回: G, M_tilde, p_e │  │
          │  └────────────────────┘  │
          └──────────────────────────┘
                      │
                      ▼
          ┌──────────────────────────┐
          │ H_min = -log₂(G) bits    │
          │ 可认证随机性              │
          └──────────────────────────┘
```

5. 约束条件的物理意义

约束	数学形式	物理意义	数量
1	Tr(Σ_e M·ρ) = p	测量概率与观测数据一致	$4n^2$
2	M ≽ 0	POVM元素是正算符	n^4+2n^3
3	Σ_a M = I⊗M_B	Alice的输出不影响Bob	n^3

约束	数学形式	物理意义	数量
4	Σ_b M = M_A⊗I	Bob的输出不影响Alice	n^3
5	Σ_b M_B = p(e)I	Bob的POVM完备性	n^2
6	Σ_a M_A = p(e)I	Alice的POVM完备性	n^2
7	Σ_e p(e) = 1	Eve的猜测是概率分布	1

6. 典型使用示例

示例1：基本使用

```python
from mdiqrng_sdp_solver import MDIQRNG_SDP_Solver
import numpy as np

# 1. 创建求解器
solver = MDIQRNG_SDP_Solver(n=2, mu=0.15, boundary=3.0, verbose=False)

# 2. 切换到纠缠测量（关键！）
solver.use_entangled_measurement_probabilities(noise_param=0.2)

# 3. 求解SDP
results = solver.solve(x_star=0, y_star=0, solver="MOSEK")

# 4. 计算随机性
G = results['optimal_value']
H_min = -np.log2(G)
print(f"G = {G:.4f}, H_min = {H_min:.4f} bits")
```

示例2：参数扫描

```python
from mdiqrng_sdp_solver import MDIQRNG_SDP_Solver
import numpy as np

for mu in [0.05, 0.10, 0.15, 0.20]:
    solver = MDIQRNG_SDP_Solver(n=2, mu=mu, verbose=False)
    solver.use_entangled_measurement_probabilities()
    results = solver.solve()

    G = results['optimal_value']
    H_min = -np.log2(G) if 0 < G < 1 else 0
    print(f"μ={mu:.2f}: G={G:.4f}, H_min={H_min:.4f} bits")
```

示例3：使用便捷函数

```python
from mdiqrng_sdp_solver import optimize_mu

# 自动扫描找最优μ
results = optimize_mu(n=2, mu_range=(0.05, 1.0), n_points=20)
print(f"最优μ = {results['optimal_mu']:.4f}")
print(f"最小G = {results['min_guessing_prob']:.4f}")
```

7. 性能与复杂度

变量数量

变量	数量	n=2时	n=3时
M_tilde	$n^2 \times n^2$ 个 4×4矩阵	16×16 = 256	81×16 = 1296
M_A	$n \times n^2$ 个 2×2矩阵	8×4 = 32	27×4 = 108
M_B	$n \times n^2$ 个 2×2矩阵	8×4 = 32	27×4 = 108
p_e	n^2 个标量	4	9
总变量		**324**	**1521**

约束数量

约束类型	数量公式	n=2时	n=3时
观测一致性	$4n^2$	16	36
无信号(A)	n^3	8	27
无信号(B)	n^3	8	27
Bob归一化	n^2	4	9
Alice归一化	n^2	4	9
Eve归一化	1	1	1
总约束		**41**	**109**

求解时间参考

配置	MOSEK	SCS
n=2, mu=0.1	~0.5s	~1s
n=3, mu=0.1	~2s	~5s
n=4, mu=0.1	~10s	~30s

文档版本: 1.0
最后更新: 2025-12-03